File 1 - extra.py:

```
1: (0)              from sys import version_info as _swig_python_version_info
2: (0)              if __package__ or "." in __name__:
3: (4)                  from . import _extra
4: (0)              else:
5: (4)                  import _extra
6: (0)              try:
7: (4)                  import builtins as __builtin__
8: (0)              except ImportError:
9: (4)                  import __builtin__
10: (0)             def _swig_repr(self):
11: (4)                 try:
12: (8)                     strthis = "proxy of " + self.this.__repr__()
13: (4)                 except __builtin__.Exception:
14: (8)                     strthis = ""
15: (4)                 return "<%s.%s; %s >" % (self.__class__.__module__,
self.__class__.__name__, strthis,)
16: (0)             def _swig_setattr_nondynamic_instance_variable(set):
17: (4)                 def set_instance_attr(self, name, value):
18: (8)                     if name == "this":
19: (12)                        set(self, name, value)
20: (8)                     elif name == "thisown":
21: (12)                        self.this.own(value)
22: (8)                     elif hasattr(self, name) and isinstance(getattr(type(self), name),
property):
23: (12)                        set(self, name, value)
24: (8)                     else:
25: (12)                        raise AttributeError("You cannot add instance attributes to %s" %
self)
26: (4)                 return set_instance_attr
27: (0)             def _swig_setattr_nondynamic_class_variable(set):
28: (4)                 def set_class_attr(cls, name, value):
29: (8)                     if hasattr(cls, name) and not isinstance(getattr(cls, name),
property):
30: (12)                        set(cls, name, value)
31: (8)                     else:
32: (12)                        raise AttributeError("You cannot add class attributes to %s" %
cls)
33: (4)                 return set_class_attr
34: (0)             def _swig_add_metaclass(metaclass):
35: (4)                 """Class decorator for adding a metaclass to a SWIG wrapped class - a
slimmed down version of six.add_metaclass"""
36: (4)                 def wrapper(cls):
37: (8)                     return metaclass(cls.__name__, cls.__bases__, cls.__dict__.copy())
38: (4)                 return wrapper
39: (0)             class _SwigNonDynamicMeta(type):
40: (4)                 """Meta class to enforce nondynamic attributes (no new attributes) for a
class"""
41: (4)                 __setattr__ = _swig_setattr_nondynamic_class_variable(type.__setattr__)
42: (0)             def page_merge(doc_des, doc_src, page_from, page_to, rotate, links,
copy_annots, graft_map):
43: (4)                 return _extra.page_merge(doc_des, doc_src, page_from, page_to, rotate,
links, copy_annots, graft_map)
44: (0)             def JM_merge_range(doc_des, doc_src, spage, epage, apage, rotate, links,
annots, show_progress, graft_map):
45: (4)                 return _extra.JM_merge_range(doc_des, doc_src, spage, epage, apage,
rotate, links, annots, show_progress, graft_map)
46: (0)             def FzDocument_insert_pdf(doc, src, from_page, to_page, start_at, rotate,
links, annots, show_progress, final, graft_map):
47: (4)                 return _extra.FzDocument_insert_pdf(doc, src, from_page, to_page,
start_at, rotate, links, annots, show_progress, final, graft_map)
48: (0)             def page_xref(this_doc, pno):
49: (4)                 return _extra.page_xref(this_doc, pno)
50: (0)             def _newPage(*args):
51: (4)                 return _extra._newPage(*args)
52: (0)             def JM_add_annot_id(annot, stem):
53: (4)                 return _extra.JM_add_annot_id(annot, stem)
54: (0)             def JM_get_annot_id_list(page):
```

```
 55: (4)                 return _extra.JM_get_annot_id_list(page)
 56: (0)             def _add_caret_annot(*args):
 57: (4)                 return _extra._add_caret_annot(*args)
 58: (0)             def Tools_parse_da(this_annot):
 59: (4)                 return _extra.Tools_parse_da(this_annot)
 60: (0)             def Annot_getAP(annot):
 61: (4)                 return _extra.Annot_getAP(annot)
 62: (0)             def Tools_update_da(this_annot, da_str):
 63: (4)                 return _extra.Tools_update_da(this_annot, da_str)
 64: (0)             def JM_point_from_py(p):
 65: (4)                 return _extra.JM_point_from_py(p)
 66: (0)             def Annot_rect(annot):
 67: (4)                 return _extra.Annot_rect(annot)
 68: (0)             def util_transform_rect(rect, matrix):
 69: (4)                 return _extra.util_transform_rect(rect, matrix)
 70: (0)             def Annot_rect3(annot):
 71: (4)                 return _extra.Annot_rect3(annot)
 72: (0)             def Page_derotate_matrix(*args):
 73: (4)                 return _extra.Page_derotate_matrix(*args)
 74: (0)             def JM_get_annot_xref_list(page_obj):
 75: (4)                 return _extra.JM_get_annot_xref_list(page_obj)
 76: (0)             def xref_object(*args):
 77: (4)                 return _extra.xref_object(*args)
 78: (0)             def Link_is_external(this_link):
 79: (4)                 return _extra.Link_is_external(this_link)
 80: (0)             def Page_addAnnot_FromString(*args):
 81: (4)                 return _extra.Page_addAnnot_FromString(*args)
 82: (0)             def Link_next(this_link):
 83: (4)                 return _extra.Link_next(this_link)
 84: (0)             def page_count_fz2(document):
 85: (4)                 return _extra.page_count_fz2(document)
 86: (0)             def page_count_fz(document):
 87: (4)                 return _extra.page_count_fz(document)
 88: (0)             def page_count_pdf(pdf):
 89: (4)                 return _extra.page_count_pdf(pdf)
 90: (0)             def page_count(*args):
 91: (4)                 return _extra.page_count(*args)
 92: (0)             def page_annot_xrefs(*args):
 93: (4)                 return _extra.page_annot_xrefs(*args)
 94: (0)             def Outline_is_external(outline):
 95: (4)                 return _extra.Outline_is_external(outline)
 96: (0)             def Document_extend_toc_items(*args):
 97: (4)                 return _extra.Document_extend_toc_items(*args)
 98: (0)             def ll_fz_absi(i):
 99: (4)                 return _extra.ll_fz_absi(i)
100: (0)             def JM_new_texttrace_device(out):
101: (4)                 return _extra.JM_new_texttrace_device(out)
102: (0)             def JM_char_bbox(line, ch):
103: (4)                 return _extra.JM_char_bbox(line, ch)
104: (0)             def JM_char_quad(line, ch):
105: (4)                 return _extra.JM_char_quad(line, ch)
106: (0)             def JM_print_stext_page_as_text(res, page):
107: (4)                 return _extra.JM_print_stext_page_as_text(res, page)
108: (0)             def set_skip_quad_corrections(on):
109: (4)                 return _extra.set_skip_quad_corrections(on)
110: (0)             def set_subset_fontnames(on):
111: (4)                 return _extra.set_subset_fontnames(on)
112: (0)             def set_small_glyph_heights(on):
113: (4)                 return _extra.set_small_glyph_heights(on)
114: (0)             def JM_cropbox(page_obj):
115: (4)                 return _extra.JM_cropbox(page_obj)
116: (0)             def get_cdrawings(page, extended=None, callback=None, method=None):
117: (4)                 return _extra.get_cdrawings(page, extended, callback, method)
118: (0)             def JM_make_spanlist(line_dict, line, raw, buff, tp_rect):
119: (4)                 return _extra.JM_make_spanlist(line_dict, line, raw, buff, tp_rect)
120: (0)             def extractWORDS(this_tpage, delimiters):
121: (4)                 return _extra.extractWORDS(this_tpage, delimiters)
122: (0)             def extractBLOCKS(_self):
123: (4)                 return _extra.extractBLOCKS(_self)
```

```
124: (0)            def link_uri(link):
125: (4)                return _extra.link_uri(link)
126: (0)            def page_get_textpage(_self, clip, flags, matrix):
127: (4)                return _extra.page_get_textpage(_self, clip, flags, matrix)
128: (0)            def JM_make_textpage_dict(tp, page_dict, raw):
129: (4)                return _extra.JM_make_textpage_dict(tp, page_dict, raw)
130: (0)            def pixmap_pixel(pm, x, y):
131: (4)                return _extra.pixmap_pixel(pm, x, y)
132: (0)            def pixmap_n(pixmap):
133: (4)                return _extra.pixmap_n(pixmap)
134: (0)            def JM_search_stext_page(page, needle):
135: (4)                return _extra.JM_search_stext_page(page, needle)
136: (0)            def set_pixel(pm, x, y, color):
137: (4)                return _extra.set_pixel(pm, x, y, color)
138: (0)            def fz_new_image_from_compressed_buffer(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, buffer, mask):
139: (4)                return _extra.fz_new_image_from_compressed_buffer(w, h, bpc, colorspace,
xres, yres, interpolate, imagemask, buffer, mask)
140: (0)            def rearrange_pages2(doc, new_pages):
141: (4)                return _extra.rearrange_pages2(doc, new_pages)
```

File 2 - mupdf.py:

```
1: (0)              from sys import version_info as _swig_python_version_info
2: (0)              if __package__ or "." in __name__:
3: (4)                  from . import _mupdf
4: (0)              else:
5: (4)                  import _mupdf
6: (0)              try:
7: (4)                  import builtins as __builtin__
8: (0)              except ImportError:
9: (4)                  import __builtin__
10: (0)             def _swig_repr(self):
11: (4)                 try:
12: (8)                     strthis = "proxy of " + self.this.__repr__()
13: (4)                 except __builtin__.Exception:
14: (8)                     strthis = ""
15: (4)                 return "<%s.%s; %s >" % (self.__class__.__module__,
self.__class__.__name__, strthis,)
16: (0)             def _swig_setattr_nondynamic_instance_variable(set):
17: (4)                 def set_instance_attr(self, name, value):
18: (8)                     if name == "this":
19: (12)                        set(self, name, value)
20: (8)                     elif name == "thisown":
21: (12)                        self.this.own(value)
22: (8)                     elif hasattr(self, name) and isinstance(getattr(type(self), name),
property):
23: (12)                        set(self, name, value)
24: (8)                     else:
25: (12)                        raise AttributeError("You cannot add instance attributes to %s" %
self)
26: (4)                 return set_instance_attr
27: (0)             def _swig_setattr_nondynamic_class_variable(set):
28: (4)                 def set_class_attr(cls, name, value):
29: (8)                     if hasattr(cls, name) and not isinstance(getattr(cls, name),
property):
30: (12)                        set(cls, name, value)
31: (8)                     else:
32: (12)                        raise AttributeError("You cannot add class attributes to %s" %
cls)
33: (4)                 return set_class_attr
34: (0)             def _swig_add_metaclass(metaclass):
35: (4)                 """Class decorator for adding a metaclass to a SWIG wrapped class - a
slimmed down version of six.add_metaclass"""
36: (4)                 def wrapper(cls):
37: (8)                     return metaclass(cls.__name__, cls.__bases__, cls.__dict__.copy())
38: (4)                 return wrapper
```

```
39: (0)             class _SwigNonDynamicMeta(type):
40: (4)                 """Meta class to enforce nondynamic attributes (no new attributes) for a
class"""
41: (4)                 __setattr__ = _swig_setattr_nondynamic_class_variable(type.__setattr__)
42: (0)             import weakref
43: (0)             def cdata(ptr, nelements=1):
44: (4)                 return _mupdf.cdata(ptr, nelements)
45: (0)             def memmove(data, indata):
46: (4)                 return _mupdf.memmove(data, indata)
47: (0)             class SwigPyIterator(object):
48: (4)                 thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
49: (4)                 def __init__(self, *args, **kwargs):
50: (8)                     raise AttributeError("No constructor defined - class is abstract")
51: (4)                 __repr__ = _swig_repr
52: (4)                 __swig_destroy__ = _mupdf.delete_SwigPyIterator
53: (4)                 def value(self):
54: (8)                     return _mupdf.SwigPyIterator_value(self)
55: (4)                 def incr(self, n=1):
56: (8)                     return _mupdf.SwigPyIterator_incr(self, n)
57: (4)                 def decr(self, n=1):
58: (8)                     return _mupdf.SwigPyIterator_decr(self, n)
59: (4)                 def distance(self, x):
60: (8)                     return _mupdf.SwigPyIterator_distance(self, x)
61: (4)                 def equal(self, x):
62: (8)                     return _mupdf.SwigPyIterator_equal(self, x)
63: (4)                 def copy(self):
64: (8)                     return _mupdf.SwigPyIterator_copy(self)
65: (4)                 def next(self):
66: (8)                     return _mupdf.SwigPyIterator_next(self)
67: (4)                 def __next__(self):
68: (8)                     return _mupdf.SwigPyIterator___next__(self)
69: (4)                 def previous(self):
70: (8)                     return _mupdf.SwigPyIterator_previous(self)
71: (4)                 def advance(self, n):
72: (8)                     return _mupdf.SwigPyIterator_advance(self, n)
73: (4)                 def __eq__(self, x):
74: (8)                     return _mupdf.SwigPyIterator___eq__(self, x)
75: (4)                 def __ne__(self, x):
76: (8)                     return _mupdf.SwigPyIterator___ne__(self, x)
77: (4)                 def __iadd__(self, n):
78: (8)                     return _mupdf.SwigPyIterator___iadd__(self, n)
79: (4)                 def __isub__(self, n):
80: (8)                     return _mupdf.SwigPyIterator___isub__(self, n)
81: (4)                 def __add__(self, n):
82: (8)                     return _mupdf.SwigPyIterator___add__(self, n)
83: (4)                 def __sub__(self, *args):
84: (8)                     return _mupdf.SwigPyIterator___sub__(self, *args)
85: (4)                 def __iter__(self):
86: (8)                     return self
87: (0)             _mupdf.SwigPyIterator_swigregister(SwigPyIterator)
88: (0)             class uchar_array(object):
89: (4)                 thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
90: (4)                 __repr__ = _swig_repr
91: (4)                 def __init__(self, nelements):
92: (8)                     _mupdf.uchar_array_swiginit(self, _mupdf.new_uchar_array(nelements))
93: (4)                 __swig_destroy__ = _mupdf.delete_uchar_array
94: (4)                 def __getitem__(self, index):
95: (8)                     return _mupdf.uchar_array___getitem__(self, index)
96: (4)                 def __setitem__(self, index, value):
97: (8)                     return _mupdf.uchar_array___setitem__(self, index, value)
98: (4)                 def cast(self):
99: (8)                     return _mupdf.uchar_array_cast(self)
100: (4)                @staticmethod
101: (4)                def frompointer(t):
102: (8)                    return _mupdf.uchar_array_frompointer(t)
103: (0)            _mupdf.uchar_array_swigregister(uchar_array)
104: (0)            class vectoruc(object):
```

```
(4)                 thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
(4)                 __repr__ = _swig_repr
(4)                 def iterator(self):
(8)                     return _mupdf.vectoruc_iterator(self)
(4)                 def __iter__(self):
(8)                     return self.iterator()
(4)                 def __nonzero__(self):
(8)                     return _mupdf.vectoruc___nonzero__(self)
(4)                 def __bool__(self):
(8)                     return _mupdf.vectoruc___bool__(self)
(4)                 def __len__(self):
(8)                     return _mupdf.vectoruc___len__(self)
(4)                 def __getslice__(self, i, j):
(8)                     return _mupdf.vectoruc___getslice__(self, i, j)
(4)                 def __setslice__(self, *args):
(8)                     return _mupdf.vectoruc___setslice__(self, *args)
(4)                 def __delslice__(self, i, j):
(8)                     return _mupdf.vectoruc___delslice__(self, i, j)
(4)                 def __delitem__(self, *args):
(8)                     return _mupdf.vectoruc___delitem__(self, *args)
(4)                 def __getitem__(self, *args):
(8)                     return _mupdf.vectoruc___getitem__(self, *args)
(4)                 def __setitem__(self, *args):
(8)                     return _mupdf.vectoruc___setitem__(self, *args)
(4)                 def pop(self):
(8)                     return _mupdf.vectoruc_pop(self)
(4)                 def append(self, x):
(8)                     return _mupdf.vectoruc_append(self, x)
(4)                 def empty(self):
(8)                     return _mupdf.vectoruc_empty(self)
(4)                 def size(self):
(8)                     return _mupdf.vectoruc_size(self)
(4)                 def swap(self, v):
(8)                     return _mupdf.vectoruc_swap(self, v)
(4)                 def begin(self):
(8)                     return _mupdf.vectoruc_begin(self)
(4)                 def end(self):
(8)                     return _mupdf.vectoruc_end(self)
(4)                 def rbegin(self):
(8)                     return _mupdf.vectoruc_rbegin(self)
(4)                 def rend(self):
(8)                     return _mupdf.vectoruc_rend(self)
(4)                 def clear(self):
(8)                     return _mupdf.vectoruc_clear(self)
(4)                 def get_allocator(self):
(8)                     return _mupdf.vectoruc_get_allocator(self)
(4)                 def pop_back(self):
(8)                     return _mupdf.vectoruc_pop_back(self)
(4)                 def erase(self, *args):
(8)                     return _mupdf.vectoruc_erase(self, *args)
(4)                 def __init__(self, *args):
(8)                     _mupdf.vectoruc_swiginit(self, _mupdf.new_vectoruc(*args))
(4)                 def push_back(self, x):
(8)                     return _mupdf.vectoruc_push_back(self, x)
(4)                 def front(self):
(8)                     return _mupdf.vectoruc_front(self)
(4)                 def back(self):
(8)                     return _mupdf.vectoruc_back(self)
(4)                 def assign(self, n, x):
(8)                     return _mupdf.vectoruc_assign(self, n, x)
(4)                 def resize(self, *args):
(8)                     return _mupdf.vectoruc_resize(self, *args)
(4)                 def insert(self, *args):
(8)                     return _mupdf.vectoruc_insert(self, *args)
(4)                 def reserve(self, n):
(8)                     return _mupdf.vectoruc_reserve(self, n)
(4)                 def capacity(self):
(8)                     return _mupdf.vectoruc_capacity(self)
```

```
173: (4)                __swig_destroy__ = _mupdf.delete_vectoruc
174: (0)            _mupdf.vectoruc_swigregister(vectoruc)
175: (0)            class vectori(object):
176: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
177: (4)                __repr__ = _swig_repr
178: (4)                def iterator(self):
179: (8)                    return _mupdf.vectori_iterator(self)
180: (4)                def __iter__(self):
181: (8)                    return self.iterator()
182: (4)                def __nonzero__(self):
183: (8)                    return _mupdf.vectori___nonzero__(self)
184: (4)                def __bool__(self):
185: (8)                    return _mupdf.vectori___bool__(self)
186: (4)                def __len__(self):
187: (8)                    return _mupdf.vectori___len__(self)
188: (4)                def __getslice__(self, i, j):
189: (8)                    return _mupdf.vectori___getslice__(self, i, j)
190: (4)                def __setslice__(self, *args):
191: (8)                    return _mupdf.vectori___setslice__(self, *args)
192: (4)                def __delslice__(self, i, j):
193: (8)                    return _mupdf.vectori___delslice__(self, i, j)
194: (4)                def __delitem__(self, *args):
195: (8)                    return _mupdf.vectori___delitem__(self, *args)
196: (4)                def __getitem__(self, *args):
197: (8)                    return _mupdf.vectori___getitem__(self, *args)
198: (4)                def __setitem__(self, *args):
199: (8)                    return _mupdf.vectori___setitem__(self, *args)
200: (4)                def pop(self):
201: (8)                    return _mupdf.vectori_pop(self)
202: (4)                def append(self, x):
203: (8)                    return _mupdf.vectori_append(self, x)
204: (4)                def empty(self):
205: (8)                    return _mupdf.vectori_empty(self)
206: (4)                def size(self):
207: (8)                    return _mupdf.vectori_size(self)
208: (4)                def swap(self, v):
209: (8)                    return _mupdf.vectori_swap(self, v)
210: (4)                def begin(self):
211: (8)                    return _mupdf.vectori_begin(self)
212: (4)                def end(self):
213: (8)                    return _mupdf.vectori_end(self)
214: (4)                def rbegin(self):
215: (8)                    return _mupdf.vectori_rbegin(self)
216: (4)                def rend(self):
217: (8)                    return _mupdf.vectori_rend(self)
218: (4)                def clear(self):
219: (8)                    return _mupdf.vectori_clear(self)
220: (4)                def get_allocator(self):
221: (8)                    return _mupdf.vectori_get_allocator(self)
222: (4)                def pop_back(self):
223: (8)                    return _mupdf.vectori_pop_back(self)
224: (4)                def erase(self, *args):
225: (8)                    return _mupdf.vectori_erase(self, *args)
226: (4)                def __init__(self, *args):
227: (8)                    _mupdf.vectori_swiginit(self, _mupdf.new_vectori(*args))
228: (4)                def push_back(self, x):
229: (8)                    return _mupdf.vectori_push_back(self, x)
230: (4)                def front(self):
231: (8)                    return _mupdf.vectori_front(self)
232: (4)                def back(self):
233: (8)                    return _mupdf.vectori_back(self)
234: (4)                def assign(self, n, x):
235: (8)                    return _mupdf.vectori_assign(self, n, x)
236: (4)                def resize(self, *args):
237: (8)                    return _mupdf.vectori_resize(self, *args)
238: (4)                def insert(self, *args):
239: (8)                    return _mupdf.vectori_insert(self, *args)
240: (4)                def reserve(self, n):
```

```
241: (8)                    return _mupdf.vectori_reserve(self, n)
242: (4)                def capacity(self):
243: (8)                    return _mupdf.vectori_capacity(self)
244: (4)                __swig_destroy__ = _mupdf.delete_vectori
245: (0)            _mupdf.vectori_swigregister(vectori)
246: (0)            class vectorf(object):
247: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
248: (4)                __repr__ = _swig_repr
249: (4)                def iterator(self):
250: (8)                    return _mupdf.vectorf_iterator(self)
251: (4)                def __iter__(self):
252: (8)                    return self.iterator()
253: (4)                def __nonzero__(self):
254: (8)                    return _mupdf.vectorf___nonzero__(self)
255: (4)                def __bool__(self):
256: (8)                    return _mupdf.vectorf___bool__(self)
257: (4)                def __len__(self):
258: (8)                    return _mupdf.vectorf___len__(self)
259: (4)                def __getslice__(self, i, j):
260: (8)                    return _mupdf.vectorf___getslice__(self, i, j)
261: (4)                def __setslice__(self, *args):
262: (8)                    return _mupdf.vectorf___setslice__(self, *args)
263: (4)                def __delslice__(self, i, j):
264: (8)                    return _mupdf.vectorf___delslice__(self, i, j)
265: (4)                def __delitem__(self, *args):
266: (8)                    return _mupdf.vectorf___delitem__(self, *args)
267: (4)                def __getitem__(self, *args):
268: (8)                    return _mupdf.vectorf___getitem__(self, *args)
269: (4)                def __setitem__(self, *args):
270: (8)                    return _mupdf.vectorf___setitem__(self, *args)
271: (4)                def pop(self):
272: (8)                    return _mupdf.vectorf_pop(self)
273: (4)                def append(self, x):
274: (8)                    return _mupdf.vectorf_append(self, x)
275: (4)                def empty(self):
276: (8)                    return _mupdf.vectorf_empty(self)
277: (4)                def size(self):
278: (8)                    return _mupdf.vectorf_size(self)
279: (4)                def swap(self, v):
280: (8)                    return _mupdf.vectorf_swap(self, v)
281: (4)                def begin(self):
282: (8)                    return _mupdf.vectorf_begin(self)
283: (4)                def end(self):
284: (8)                    return _mupdf.vectorf_end(self)
285: (4)                def rbegin(self):
286: (8)                    return _mupdf.vectorf_rbegin(self)
287: (4)                def rend(self):
288: (8)                    return _mupdf.vectorf_rend(self)
289: (4)                def clear(self):
290: (8)                    return _mupdf.vectorf_clear(self)
291: (4)                def get_allocator(self):
292: (8)                    return _mupdf.vectorf_get_allocator(self)
293: (4)                def pop_back(self):
294: (8)                    return _mupdf.vectorf_pop_back(self)
295: (4)                def erase(self, *args):
296: (8)                    return _mupdf.vectorf_erase(self, *args)
297: (4)                def __init__(self, *args):
298: (8)                    _mupdf.vectorf_swiginit(self, _mupdf.new_vectorf(*args))
299: (4)                def push_back(self, x):
300: (8)                    return _mupdf.vectorf_push_back(self, x)
301: (4)                def front(self):
302: (8)                    return _mupdf.vectorf_front(self)
303: (4)                def back(self):
304: (8)                    return _mupdf.vectorf_back(self)
305: (4)                def assign(self, n, x):
306: (8)                    return _mupdf.vectorf_assign(self, n, x)
307: (4)                def resize(self, *args):
308: (8)                    return _mupdf.vectorf_resize(self, *args)
```

```
309: (4)                def insert(self, *args):
310: (8)                    return _mupdf.vectorf_insert(self, *args)
311: (4)                def reserve(self, n):
312: (8)                    return _mupdf.vectorf_reserve(self, n)
313: (4)                def capacity(self):
314: (8)                    return _mupdf.vectorf_capacity(self)
315: (4)                __swig_destroy__ = _mupdf.delete_vectorf
316: (0)            _mupdf.vectorf_swigregister(vectorf)
317: (0)            class vectord(object):
318: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
319: (4)                __repr__ = _swig_repr
320: (4)                def iterator(self):
321: (8)                    return _mupdf.vectord_iterator(self)
322: (4)                def __iter__(self):
323: (8)                    return self.iterator()
324: (4)                def __nonzero__(self):
325: (8)                    return _mupdf.vectord___nonzero__(self)
326: (4)                def __bool__(self):
327: (8)                    return _mupdf.vectord___bool__(self)
328: (4)                def __len__(self):
329: (8)                    return _mupdf.vectord___len__(self)
330: (4)                def __getslice__(self, i, j):
331: (8)                    return _mupdf.vectord___getslice__(self, i, j)
332: (4)                def __setslice__(self, *args):
333: (8)                    return _mupdf.vectord___setslice__(self, *args)
334: (4)                def __delslice__(self, i, j):
335: (8)                    return _mupdf.vectord___delslice__(self, i, j)
336: (4)                def __delitem__(self, *args):
337: (8)                    return _mupdf.vectord___delitem__(self, *args)
338: (4)                def __getitem__(self, *args):
339: (8)                    return _mupdf.vectord___getitem__(self, *args)
340: (4)                def __setitem__(self, *args):
341: (8)                    return _mupdf.vectord___setitem__(self, *args)
342: (4)                def pop(self):
343: (8)                    return _mupdf.vectord_pop(self)
344: (4)                def append(self, x):
345: (8)                    return _mupdf.vectord_append(self, x)
346: (4)                def empty(self):
347: (8)                    return _mupdf.vectord_empty(self)
348: (4)                def size(self):
349: (8)                    return _mupdf.vectord_size(self)
350: (4)                def swap(self, v):
351: (8)                    return _mupdf.vectord_swap(self, v)
352: (4)                def begin(self):
353: (8)                    return _mupdf.vectord_begin(self)
354: (4)                def end(self):
355: (8)                    return _mupdf.vectord_end(self)
356: (4)                def rbegin(self):
357: (8)                    return _mupdf.vectord_rbegin(self)
358: (4)                def rend(self):
359: (8)                    return _mupdf.vectord_rend(self)
360: (4)                def clear(self):
361: (8)                    return _mupdf.vectord_clear(self)
362: (4)                def get_allocator(self):
363: (8)                    return _mupdf.vectord_get_allocator(self)
364: (4)                def pop_back(self):
365: (8)                    return _mupdf.vectord_pop_back(self)
366: (4)                def erase(self, *args):
367: (8)                    return _mupdf.vectord_erase(self, *args)
368: (4)                def __init__(self, *args):
369: (8)                    _mupdf.vectord_swiginit(self, _mupdf.new_vectord(*args))
370: (4)                def push_back(self, x):
371: (8)                    return _mupdf.vectord_push_back(self, x)
372: (4)                def front(self):
373: (8)                    return _mupdf.vectord_front(self)
374: (4)                def back(self):
375: (8)                    return _mupdf.vectord_back(self)
376: (4)                def assign(self, n, x):
```

```
377: (8)                    return _mupdf.vectord_assign(self, n, x)
378: (4)                def resize(self, *args):
379: (8)                    return _mupdf.vectord_resize(self, *args)
380: (4)                def insert(self, *args):
381: (8)                    return _mupdf.vectord_insert(self, *args)
382: (4)                def reserve(self, n):
383: (8)                    return _mupdf.vectord_reserve(self, n)
384: (4)                def capacity(self):
385: (8)                    return _mupdf.vectord_capacity(self)
386: (4)                __swig_destroy__ = _mupdf.delete_vectord
387: (0)            _mupdf.vectord_swigregister(vectord)
388: (0)            class vectors(object):
389: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
390: (4)                __repr__ = _swig_repr
391: (4)                def iterator(self):
392: (8)                    return _mupdf.vectors_iterator(self)
393: (4)                def __iter__(self):
394: (8)                    return self.iterator()
395: (4)                def __nonzero__(self):
396: (8)                    return _mupdf.vectors___nonzero__(self)
397: (4)                def __bool__(self):
398: (8)                    return _mupdf.vectors___bool__(self)
399: (4)                def __len__(self):
400: (8)                    return _mupdf.vectors___len__(self)
401: (4)                def __getslice__(self, i, j):
402: (8)                    return _mupdf.vectors___getslice__(self, i, j)
403: (4)                def __setslice__(self, *args):
404: (8)                    return _mupdf.vectors___setslice__(self, *args)
405: (4)                def __delslice__(self, i, j):
406: (8)                    return _mupdf.vectors___delslice__(self, i, j)
407: (4)                def __delitem__(self, *args):
408: (8)                    return _mupdf.vectors___delitem__(self, *args)
409: (4)                def __getitem__(self, *args):
410: (8)                    return _mupdf.vectors___getitem__(self, *args)
411: (4)                def __setitem__(self, *args):
412: (8)                    return _mupdf.vectors___setitem__(self, *args)
413: (4)                def pop(self):
414: (8)                    return _mupdf.vectors_pop(self)
415: (4)                def append(self, x):
416: (8)                    return _mupdf.vectors_append(self, x)
417: (4)                def empty(self):
418: (8)                    return _mupdf.vectors_empty(self)
419: (4)                def size(self):
420: (8)                    return _mupdf.vectors_size(self)
421: (4)                def swap(self, v):
422: (8)                    return _mupdf.vectors_swap(self, v)
423: (4)                def begin(self):
424: (8)                    return _mupdf.vectors_begin(self)
425: (4)                def end(self):
426: (8)                    return _mupdf.vectors_end(self)
427: (4)                def rbegin(self):
428: (8)                    return _mupdf.vectors_rbegin(self)
429: (4)                def rend(self):
430: (8)                    return _mupdf.vectors_rend(self)
431: (4)                def clear(self):
432: (8)                    return _mupdf.vectors_clear(self)
433: (4)                def get_allocator(self):
434: (8)                    return _mupdf.vectors_get_allocator(self)
435: (4)                def pop_back(self):
436: (8)                    return _mupdf.vectors_pop_back(self)
437: (4)                def erase(self, *args):
438: (8)                    return _mupdf.vectors_erase(self, *args)
439: (4)                def __init__(self, *args):
440: (8)                    _mupdf.vectors_swiginit(self, _mupdf.new_vectors(*args))
441: (4)                def push_back(self, x):
442: (8)                    return _mupdf.vectors_push_back(self, x)
443: (4)                def front(self):
444: (8)                    return _mupdf.vectors_front(self)
```

```
445: (4)                def back(self):
446: (8)                    return _mupdf.vectors_back(self)
447: (4)                def assign(self, n, x):
448: (8)                    return _mupdf.vectors_assign(self, n, x)
449: (4)                def resize(self, *args):
450: (8)                    return _mupdf.vectors_resize(self, *args)
451: (4)                def insert(self, *args):
452: (8)                    return _mupdf.vectors_insert(self, *args)
453: (4)                def reserve(self, n):
454: (8)                    return _mupdf.vectors_reserve(self, n)
455: (4)                def capacity(self):
456: (8)                    return _mupdf.vectors_capacity(self)
457: (4)                __swig_destroy__ = _mupdf.delete_vectors
458: (0)            _mupdf.vectors_swigregister(vectors)
459: (0)            class map_string_int(object):
460: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
461: (4)                __repr__ = _swig_repr
462: (4)                def iterator(self):
463: (8)                    return _mupdf.map_string_int_iterator(self)
464: (4)                def __iter__(self):
465: (8)                    return self.iterator()
466: (4)                def __nonzero__(self):
467: (8)                    return _mupdf.map_string_int___nonzero__(self)
468: (4)                def __bool__(self):
469: (8)                    return _mupdf.map_string_int___bool__(self)
470: (4)                def __len__(self):
471: (8)                    return _mupdf.map_string_int___len__(self)
472: (4)                def __iter__(self):
473: (8)                    return self.key_iterator()
474: (4)                def iterkeys(self):
475: (8)                    return self.key_iterator()
476: (4)                def itervalues(self):
477: (8)                    return self.value_iterator()
478: (4)                def iteritems(self):
479: (8)                    return self.iterator()
480: (4)                def __getitem__(self, key):
481: (8)                    return _mupdf.map_string_int___getitem__(self, key)
482: (4)                def __delitem__(self, key):
483: (8)                    return _mupdf.map_string_int___delitem__(self, key)
484: (4)                def has_key(self, key):
485: (8)                    return _mupdf.map_string_int_has_key(self, key)
486: (4)                def keys(self):
487: (8)                    return _mupdf.map_string_int_keys(self)
488: (4)                def values(self):
489: (8)                    return _mupdf.map_string_int_values(self)
490: (4)                def items(self):
491: (8)                    return _mupdf.map_string_int_items(self)
492: (4)                def __contains__(self, key):
493: (8)                    return _mupdf.map_string_int___contains__(self, key)
494: (4)                def key_iterator(self):
495: (8)                    return _mupdf.map_string_int_key_iterator(self)
496: (4)                def value_iterator(self):
497: (8)                    return _mupdf.map_string_int_value_iterator(self)
498: (4)                def __setitem__(self, *args):
499: (8)                    return _mupdf.map_string_int___setitem__(self, *args)
500: (4)                def asdict(self):
501: (8)                    return _mupdf.map_string_int_asdict(self)
502: (4)                def __init__(self, *args):
503: (8)                    _mupdf.map_string_int_swiginit(self, _mupdf.new_map_string_int(*args))
504: (4)                def empty(self):
505: (8)                    return _mupdf.map_string_int_empty(self)
506: (4)                def size(self):
507: (8)                    return _mupdf.map_string_int_size(self)
508: (4)                def swap(self, v):
509: (8)                    return _mupdf.map_string_int_swap(self, v)
510: (4)                def begin(self):
511: (8)                    return _mupdf.map_string_int_begin(self)
512: (4)                def end(self):
```

```
513: (8)                    return _mupdf.map_string_int_end(self)
514: (4)                def rbegin(self):
515: (8)                    return _mupdf.map_string_int_rbegin(self)
516: (4)                def rend(self):
517: (8)                    return _mupdf.map_string_int_rend(self)
518: (4)                def clear(self):
519: (8)                    return _mupdf.map_string_int_clear(self)
520: (4)                def get_allocator(self):
521: (8)                    return _mupdf.map_string_int_get_allocator(self)
522: (4)                def count(self, x):
523: (8)                    return _mupdf.map_string_int_count(self, x)
524: (4)                def erase(self, *args):
525: (8)                    return _mupdf.map_string_int_erase(self, *args)
526: (4)                def find(self, x):
527: (8)                    return _mupdf.map_string_int_find(self, x)
528: (4)                def lower_bound(self, x):
529: (8)                    return _mupdf.map_string_int_lower_bound(self, x)
530: (4)                def upper_bound(self, x):
531: (8)                    return _mupdf.map_string_int_upper_bound(self, x)
532: (4)                __swig_destroy__ = _mupdf.delete_map_string_int
533: (0)            _mupdf.map_string_int_swigregister(map_string_int)
534: (0)            class vectorq(object):
535: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
536: (4)                __repr__ = _swig_repr
537: (4)                def iterator(self):
538: (8)                    return _mupdf.vectorq_iterator(self)
539: (4)                def __iter__(self):
540: (8)                    return self.iterator()
541: (4)                def __nonzero__(self):
542: (8)                    return _mupdf.vectorq___nonzero__(self)
543: (4)                def __bool__(self):
544: (8)                    return _mupdf.vectorq___bool__(self)
545: (4)                def __len__(self):
546: (8)                    return _mupdf.vectorq___len__(self)
547: (4)                def __getslice__(self, i, j):
548: (8)                    return _mupdf.vectorq___getslice__(self, i, j)
549: (4)                def __setslice__(self, *args):
550: (8)                    return _mupdf.vectorq___setslice__(self, *args)
551: (4)                def __delslice__(self, i, j):
552: (8)                    return _mupdf.vectorq___delslice__(self, i, j)
553: (4)                def __delitem__(self, *args):
554: (8)                    return _mupdf.vectorq___delitem__(self, *args)
555: (4)                def __getitem__(self, *args):
556: (8)                    return _mupdf.vectorq___getitem__(self, *args)
557: (4)                def __setitem__(self, *args):
558: (8)                    return _mupdf.vectorq___setitem__(self, *args)
559: (4)                def pop(self):
560: (8)                    return _mupdf.vectorq_pop(self)
561: (4)                def append(self, x):
562: (8)                    return _mupdf.vectorq_append(self, x)
563: (4)                def empty(self):
564: (8)                    return _mupdf.vectorq_empty(self)
565: (4)                def size(self):
566: (8)                    return _mupdf.vectorq_size(self)
567: (4)                def swap(self, v):
568: (8)                    return _mupdf.vectorq_swap(self, v)
569: (4)                def begin(self):
570: (8)                    return _mupdf.vectorq_begin(self)
571: (4)                def end(self):
572: (8)                    return _mupdf.vectorq_end(self)
573: (4)                def rbegin(self):
574: (8)                    return _mupdf.vectorq_rbegin(self)
575: (4)                def rend(self):
576: (8)                    return _mupdf.vectorq_rend(self)
577: (4)                def clear(self):
578: (8)                    return _mupdf.vectorq_clear(self)
579: (4)                def get_allocator(self):
580: (8)                    return _mupdf.vectorq_get_allocator(self)
```

```
(4)                 def pop_back(self):
(8)                     return _mupdf.vectorq_pop_back(self)
(4)                 def erase(self, *args):
(8)                     return _mupdf.vectorq_erase(self, *args)
(4)                 def __init__(self, *args):
(8)                     _mupdf.vectorq_swiginit(self, _mupdf.new_vectorq(*args))
(4)                 def push_back(self, x):
(8)                     return _mupdf.vectorq_push_back(self, x)
(4)                 def front(self):
(8)                     return _mupdf.vectorq_front(self)
(4)                 def back(self):
(8)                     return _mupdf.vectorq_back(self)
(4)                 def assign(self, n, x):
(8)                     return _mupdf.vectorq_assign(self, n, x)
(4)                 def resize(self, *args):
(8)                     return _mupdf.vectorq_resize(self, *args)
(4)                 def insert(self, *args):
(8)                     return _mupdf.vectorq_insert(self, *args)
(4)                 def reserve(self, n):
(8)                     return _mupdf.vectorq_reserve(self, n)
(4)                 def capacity(self):
(8)                     return _mupdf.vectorq_capacity(self)
(4)                 __swig_destroy__ = _mupdf.delete_vectorq
(0)             _mupdf.vectorq_swigregister(vectorq)
(0)             class vector_search_page2_hit(object):
(4)                 thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
(4)                 __repr__ = _swig_repr
(4)                 def iterator(self):
(8)                     return _mupdf.vector_search_page2_hit_iterator(self)
(4)                 def __iter__(self):
(8)                     return self.iterator()
(4)                 def __nonzero__(self):
(8)                     return _mupdf.vector_search_page2_hit___nonzero__(self)
(4)                 def __bool__(self):
(8)                     return _mupdf.vector_search_page2_hit___bool__(self)
(4)                 def __len__(self):
(8)                     return _mupdf.vector_search_page2_hit___len__(self)
(4)                 def __getslice__(self, i, j):
(8)                     return _mupdf.vector_search_page2_hit___getslice__(self, i, j)
(4)                 def __setslice__(self, *args):
(8)                     return _mupdf.vector_search_page2_hit___setslice__(self, *args)
(4)                 def __delslice__(self, i, j):
(8)                     return _mupdf.vector_search_page2_hit___delslice__(self, i, j)
(4)                 def __delitem__(self, *args):
(8)                     return _mupdf.vector_search_page2_hit___delitem__(self, *args)
(4)                 def __getitem__(self, *args):
(8)                     return _mupdf.vector_search_page2_hit___getitem__(self, *args)
(4)                 def __setitem__(self, *args):
(8)                     return _mupdf.vector_search_page2_hit___setitem__(self, *args)
(4)                 def pop(self):
(8)                     return _mupdf.vector_search_page2_hit_pop(self)
(4)                 def append(self, x):
(8)                     return _mupdf.vector_search_page2_hit_append(self, x)
(4)                 def empty(self):
(8)                     return _mupdf.vector_search_page2_hit_empty(self)
(4)                 def size(self):
(8)                     return _mupdf.vector_search_page2_hit_size(self)
(4)                 def swap(self, v):
(8)                     return _mupdf.vector_search_page2_hit_swap(self, v)
(4)                 def begin(self):
(8)                     return _mupdf.vector_search_page2_hit_begin(self)
(4)                 def end(self):
(8)                     return _mupdf.vector_search_page2_hit_end(self)
(4)                 def rbegin(self):
(8)                     return _mupdf.vector_search_page2_hit_rbegin(self)
(4)                 def rend(self):
(8)                     return _mupdf.vector_search_page2_hit_rend(self)
(4)                 def clear(self):
```

```
649: (8)                    return _mupdf.vector_search_page2_hit_clear(self)
650: (4)                def get_allocator(self):
651: (8)                    return _mupdf.vector_search_page2_hit_get_allocator(self)
652: (4)                def pop_back(self):
653: (8)                    return _mupdf.vector_search_page2_hit_pop_back(self)
654: (4)                def erase(self, *args):
655: (8)                    return _mupdf.vector_search_page2_hit_erase(self, *args)
656: (4)                def __init__(self, *args):
657: (8)                    _mupdf.vector_search_page2_hit_swiginit(self,
_mupdf.new_vector_search_page2_hit(*args))
658: (4)                def push_back(self, x):
659: (8)                    return _mupdf.vector_search_page2_hit_push_back(self, x)
660: (4)                def front(self):
661: (8)                    return _mupdf.vector_search_page2_hit_front(self)
662: (4)                def back(self):
663: (8)                    return _mupdf.vector_search_page2_hit_back(self)
664: (4)                def assign(self, n, x):
665: (8)                    return _mupdf.vector_search_page2_hit_assign(self, n, x)
666: (4)                def resize(self, *args):
667: (8)                    return _mupdf.vector_search_page2_hit_resize(self, *args)
668: (4)                def insert(self, *args):
669: (8)                    return _mupdf.vector_search_page2_hit_insert(self, *args)
670: (4)                def reserve(self, n):
671: (8)                    return _mupdf.vector_search_page2_hit_reserve(self, n)
672: (4)                def capacity(self):
673: (8)                    return _mupdf.vector_search_page2_hit_capacity(self)
674: (4)                __swig_destroy__ = _mupdf.delete_vector_search_page2_hit
675: (0)            _mupdf.vector_search_page2_hit_swigregister(vector_search_page2_hit)
676: (0)            def new_bytes(nelements):
677: (4)                return _mupdf.new_bytes(nelements)
678: (0)            def delete_bytes(ary):
679: (4)                return _mupdf.delete_bytes(ary)
680: (0)            def bytes_getitem(ary, index):
681: (4)                return _mupdf.bytes_getitem(ary, index)
682: (0)            def bytes_setitem(ary, index, value):
683: (4)                return _mupdf.bytes_setitem(ary, index, value)
684: (0)            def new_floats(nelements):
685: (4)                return _mupdf.new_floats(nelements)
686: (0)            def delete_floats(ary):
687: (4)                return _mupdf.delete_floats(ary)
688: (0)            def floats_getitem(ary, index):
689: (4)                return _mupdf.floats_getitem(ary, index)
690: (0)            def floats_setitem(ary, index, value):
691: (4)                return _mupdf.floats_setitem(ary, index, value)
692: (0)            def internal_set_error_classes(classes):
693: (4)                return _mupdf.internal_set_error_classes(classes)
694: (0)            FZ_VERSION = _mupdf.FZ_VERSION
695: (0)            r"""This file was auto-generated by mupdfwrap.py."""
696: (0)            FZ_VERSION_MAJOR = _mupdf.FZ_VERSION_MAJOR
697: (0)            FZ_VERSION_MINOR = _mupdf.FZ_VERSION_MINOR
698: (0)            FZ_VERSION_PATCH = _mupdf.FZ_VERSION_PATCH
699: (0)            TOFU_CJK_EXT = _mupdf.TOFU_CJK_EXT
700: (0)            FZ_ENABLE_SPOT_RENDERING = _mupdf.FZ_ENABLE_SPOT_RENDERING
701: (0)            r"""
702: (4)                Enable the following for spot (and hence overprint/overprint
703: (4)                simulation) capable rendering. This forces FZ_PLOTTERS_N on.
704: (4)                Choose which plotters we need.
705: (4)                By default we build all the plotters in. To avoid building
706: (4)                plotters in that aren't needed, define the unwanted
707: (4)                FZ_PLOTTERS_... define to 0.
708: (4)                Choose which document agents to include.
709: (4)                By default all are enabled. To avoid building unwanted
710: (4)                ones, define FZ_ENABLE_... to 0.
711: (4)                Choose which document writers to include.
712: (4)                By default all are enabled. To avoid building unwanted
713: (4)                ones, define FZ_ENABLE_..._OUTPUT to 0.
714: (4)                Choose whether to enable ICC color profiles.
715: (4)                Choose whether to enable JPEG2000 decoding.
716: (4)                By default, it is enabled, but due to frequent security
```

```
(4)                 issues with the third party libraries we support disabling
(4)                 it with this flag.
(4)                 Choose whether to enable JavaScript.
(4)                 By default JavaScript is enabled both for mutool and PDF
(4)                 interactivity.
(4)                 Choose which fonts to include.
(4)                 By default we include the base 14 PDF fonts,
(4)                 DroidSansFallback from Android for CJK, and
(4)                 Charis SIL from SIL for epub/html.
(4)                 Enable the following defines to AVOID including
(4)                 unwanted fonts.
(4)                 """
(0)             FZ_PLOTTERS_N = _mupdf.FZ_PLOTTERS_N
(0)             FZ_PLOTTERS_G = _mupdf.FZ_PLOTTERS_G
(0)             FZ_PLOTTERS_RGB = _mupdf.FZ_PLOTTERS_RGB
(0)             FZ_PLOTTERS_CMYK = _mupdf.FZ_PLOTTERS_CMYK
(0)             FZ_ENABLE_PDF = _mupdf.FZ_ENABLE_PDF
(0)             FZ_ENABLE_XPS = _mupdf.FZ_ENABLE_XPS
(0)             FZ_ENABLE_SVG = _mupdf.FZ_ENABLE_SVG
(0)             FZ_ENABLE_CBZ = _mupdf.FZ_ENABLE_CBZ
(0)             FZ_ENABLE_IMG = _mupdf.FZ_ENABLE_IMG
(0)             FZ_ENABLE_HTML = _mupdf.FZ_ENABLE_HTML
(0)             FZ_ENABLE_EPUB = _mupdf.FZ_ENABLE_EPUB
(0)             FZ_ENABLE_OCR_OUTPUT = _mupdf.FZ_ENABLE_OCR_OUTPUT
(0)             FZ_ENABLE_ODT_OUTPUT = _mupdf.FZ_ENABLE_ODT_OUTPUT
(0)             FZ_ENABLE_DOCX_OUTPUT = _mupdf.FZ_ENABLE_DOCX_OUTPUT
(0)             FZ_ENABLE_JPX = _mupdf.FZ_ENABLE_JPX
(0)             FZ_ENABLE_JS = _mupdf.FZ_ENABLE_JS
(0)             FZ_ENABLE_ICC = _mupdf.FZ_ENABLE_ICC
(0)             MEMENTO_PREFILL = _mupdf.MEMENTO_PREFILL
(0)             MEMENTO_POSTFILL = _mupdf.MEMENTO_POSTFILL
(0)             MEMENTO_ALLOCFILL = _mupdf.MEMENTO_ALLOCFILL
(0)             MEMENTO_FREEFILL = _mupdf.MEMENTO_FREEFILL
(0)             def Memento_checkBlock(arg1):
(4)                 return _mupdf.Memento_checkBlock(arg1)
(0)             def Memento_checkAllMemory():
(4)                 return _mupdf.Memento_checkAllMemory()
(0)             def Memento_check():
(4)                 return _mupdf.Memento_check()
(0)             def Memento_setParanoia(arg1):
(4)                 return _mupdf.Memento_setParanoia(arg1)
(0)             def Memento_paranoidAt(arg1):
(4)                 return _mupdf.Memento_paranoidAt(arg1)
(0)             def Memento_breakAt(arg1):
(4)                 return _mupdf.Memento_breakAt(arg1)
(0)             def Memento_breakOnFree(a):
(4)                 return _mupdf.Memento_breakOnFree(a)
(0)             def Memento_breakOnRealloc(a):
(4)                 return _mupdf.Memento_breakOnRealloc(a)
(0)             def Memento_getBlockNum(arg1):
(4)                 return _mupdf.Memento_getBlockNum(arg1)
(0)             def Memento_find(a):
(4)                 return _mupdf.Memento_find(a)
(0)             def Memento_breakpoint():
(4)                 return _mupdf.Memento_breakpoint()
(0)             def Memento_failAt(arg1):
(4)                 return _mupdf.Memento_failAt(arg1)
(0)             def Memento_failThisEvent():
(4)                 return _mupdf.Memento_failThisEvent()
(0)             def Memento_listBlocks():
(4)                 return _mupdf.Memento_listBlocks()
(0)             def Memento_listNewBlocks():
(4)                 return _mupdf.Memento_listNewBlocks()
(0)             def Memento_listPhasedBlocks():
(4)                 return _mupdf.Memento_listPhasedBlocks()
(0)             def Memento_setMax(arg1):
(4)                 return _mupdf.Memento_setMax(arg1)
(0)             def Memento_stats():
(4)                 return _mupdf.Memento_stats()
```

```
786: (0)            def Memento_label(arg1, arg2):
787: (4)                return _mupdf.Memento_label(arg1, arg2)
788: (0)            def Memento_tick():
789: (4)                return _mupdf.Memento_tick()
790: (0)            def Memento_setVerbose(arg1):
791: (4)                return _mupdf.Memento_setVerbose(arg1)
792: (0)            def Memento_addBacktraceLimitFnname(fnname):
793: (4)                return _mupdf.Memento_addBacktraceLimitFnname(fnname)
794: (0)            def Memento_setAtexitFin(atexitfin):
795: (4)                return _mupdf.Memento_setAtexitFin(atexitfin)
796: (0)            def Memento_setIgnoreNewDelete(ignore):
797: (4)                return _mupdf.Memento_setIgnoreNewDelete(ignore)
798: (0)            def Memento_malloc(s):
799: (4)                return _mupdf.Memento_malloc(s)
800: (0)            def Memento_realloc(arg1, s):
801: (4)                return _mupdf.Memento_realloc(arg1, s)
802: (0)            def Memento_free(arg1):
803: (4)                return _mupdf.Memento_free(arg1)
804: (0)            def Memento_calloc(arg1, arg2):
805: (4)                return _mupdf.Memento_calloc(arg1, arg2)
806: (0)            def Memento_strdup(arg1):
807: (4)                return _mupdf.Memento_strdup(arg1)
808: (0)            def Memento_info(addr):
809: (4)                return _mupdf.Memento_info(addr)
810: (0)            def Memento_listBlockInfo():
811: (4)                return _mupdf.Memento_listBlockInfo()
812: (0)            def Memento_blockInfo(blk):
813: (4)                return _mupdf.Memento_blockInfo(blk)
814: (0)            def Memento_takeByteRef(blk):
815: (4)                return _mupdf.Memento_takeByteRef(blk)
816: (0)            def Memento_dropByteRef(blk):
817: (4)                return _mupdf.Memento_dropByteRef(blk)
818: (0)            def Memento_takeShortRef(blk):
819: (4)                return _mupdf.Memento_takeShortRef(blk)
820: (0)            def Memento_dropShortRef(blk):
821: (4)                return _mupdf.Memento_dropShortRef(blk)
822: (0)            def Memento_takeIntRef(blk):
823: (4)                return _mupdf.Memento_takeIntRef(blk)
824: (0)            def Memento_dropIntRef(blk):
825: (4)                return _mupdf.Memento_dropIntRef(blk)
826: (0)            def Memento_takeRef(blk):
827: (4)                return _mupdf.Memento_takeRef(blk)
828: (0)            def Memento_dropRef(blk):
829: (4)                return _mupdf.Memento_dropRef(blk)
830: (0)            def Memento_adjustRef(blk, adjust):
831: (4)                return _mupdf.Memento_adjustRef(blk, adjust)
832: (0)            def Memento_reference(blk):
833: (4)                return _mupdf.Memento_reference(blk)
834: (0)            def Memento_checkPointerOrNull(blk):
835: (4)                return _mupdf.Memento_checkPointerOrNull(blk)
836: (0)            def Memento_checkBytePointerOrNull(blk):
837: (4)                return _mupdf.Memento_checkBytePointerOrNull(blk)
838: (0)            def Memento_checkShortPointerOrNull(blk):
839: (4)                return _mupdf.Memento_checkShortPointerOrNull(blk)
840: (0)            def Memento_checkIntPointerOrNull(blk):
841: (4)                return _mupdf.Memento_checkIntPointerOrNull(blk)
842: (0)            def Memento_startLeaking():
843: (4)                return _mupdf.Memento_startLeaking()
844: (0)            def Memento_stopLeaking():
845: (4)                return _mupdf.Memento_stopLeaking()
846: (0)            def Memento_sequence():
847: (4)                return _mupdf.Memento_sequence()
848: (0)            def Memento_squeezing():
849: (4)                return _mupdf.Memento_squeezing()
850: (0)            def Memento_fin():
851: (4)                return _mupdf.Memento_fin()
852: (0)            def Memento_bt():
853: (4)                return _mupdf.Memento_bt()
854: (0)            FZ_PI = _mupdf.FZ_PI
```

```
855: (0)            FZ_RADIAN = _mupdf.FZ_RADIAN
856: (0)            FZ_DEGREE = _mupdf.FZ_DEGREE
857: (0)            FZ_SQRT2 = _mupdf.FZ_SQRT2
858: (0)            FZ_LN2 = _mupdf.FZ_LN2
859: (0)            HAVE_SIGSETJMP = _mupdf.HAVE_SIGSETJMP
860: (0)            r"""
861: (4)                Spot architectures where we have optimisations.
862: (4)                Some differences in libc can be smoothed over
863: (4)                """
864: (0)            EOF = _mupdf.EOF
865: (0)            SEEK_SET = _mupdf.SEEK_SET
866: (0)            SEEK_CUR = _mupdf.SEEK_CUR
867: (0)            SEEK_END = _mupdf.SEEK_END
868: (0)            class fz_point(object):
869: (4)                r"""    fz_point is a point in a two-dimensional space."""
870: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
871: (4)                __repr__ = _swig_repr
872: (4)                x = property(_mupdf.fz_point_x_get, _mupdf.fz_point_x_set)
873: (4)                y = property(_mupdf.fz_point_y_get, _mupdf.fz_point_y_set)
874: (4)                def __init__(self):
875: (8)                    _mupdf.fz_point_swiginit(self, _mupdf.new_fz_point())
876: (4)                __swig_destroy__ = _mupdf.delete_fz_point
877: (0)            _mupdf.fz_point_swigregister(fz_point)
878: (0)            class fz_rect(object):
879: (4)                r"""
880: (4)                fz_rect is a rectangle represented by two diagonally opposite
881: (4)                corners at arbitrary coordinates.
882: (4)                Rectangles are always axis-aligned with the X- and Y- axes. We
883: (4)                wish to distinguish rectangles in 3 categories; infinite, finite,
884: (4)                and invalid. Zero area rectangles are a sub-category of finite
885: (4)                ones.
886: (4)                For all valid rectangles, x0 <= x1 and y0 <= y1 in all cases.
887: (4)                Infinite rectangles have x0 = y0 = FZ_MIN_INF_RECT,
888: (4)                x1 = y1 = FZ_MAX_INF_RECT. For any non infinite valid rectangle,
889: (4)                the area is defined as (x1 - x0) * (y1 - y0).
890: (4)                To check for empty or infinite rectangles use fz_is_empty_rect
891: (4)                and fz_is_infinite_rect. To check for valid rectangles use
892: (4)                fz_is_valid_rect.
893: (4)                We choose this representation, so that we can easily distinguish
894: (4)                the difference between intersecting 2 valid rectangles and
895: (4)                getting an invalid one, as opposed to getting a zero area one
896: (4)                (which nonetheless has valid bounds within the plane).
897: (4)                x0, y0: The top left corner.
898: (4)                x1, y1: The bottom right corner.
899: (4)                We choose FZ_{MIN,MAX}_INF_RECT to be the largest 32bit signed
900: (4)                integer values that survive roundtripping to floats.
901: (4)                """
902: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
903: (4)                __repr__ = _swig_repr
904: (4)                x0 = property(_mupdf.fz_rect_x0_get, _mupdf.fz_rect_x0_set)
905: (4)                y0 = property(_mupdf.fz_rect_y0_get, _mupdf.fz_rect_y0_set)
906: (4)                x1 = property(_mupdf.fz_rect_x1_get, _mupdf.fz_rect_x1_set)
907: (4)                y1 = property(_mupdf.fz_rect_y1_get, _mupdf.fz_rect_y1_set)
908: (4)                def __init__(self):
909: (8)                    _mupdf.fz_rect_swiginit(self, _mupdf.new_fz_rect())
910: (4)                __swig_destroy__ = _mupdf.delete_fz_rect
911: (0)            _mupdf.fz_rect_swigregister(fz_rect)
912: (0)            class fz_irect(object):
913: (4)                r"""
914: (4)                fz_irect is a rectangle using integers instead of floats.
915: (4)                It's used in the draw device and for pixmap dimensions.
916: (4)                """
917: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
918: (4)                __repr__ = _swig_repr
919: (4)                x0 = property(_mupdf.fz_irect_x0_get, _mupdf.fz_irect_x0_set)
920: (4)                y0 = property(_mupdf.fz_irect_y0_get, _mupdf.fz_irect_y0_set)
```

```
921: (4)                x1 = property(_mupdf.fz_irect_x1_get, _mupdf.fz_irect_x1_set)
922: (4)                y1 = property(_mupdf.fz_irect_y1_get, _mupdf.fz_irect_y1_set)
923: (4)                def __init__(self):
924: (8)                    _mupdf.fz_irect_swiginit(self, _mupdf.new_fz_irect())
925: (4)                __swig_destroy__ = _mupdf.delete_fz_irect
926: (0)            _mupdf.fz_irect_swigregister(fz_irect)
927: (0)            class fz_matrix(object):
928: (4)                r"""
929: (8)                    fz_matrix is a row-major 3x3 matrix used for representing
930: (8)                    transformations of coordinates throughout MuPDF.
931: (8)                    Since all points reside in a two-dimensional space, one vector
932: (8)                    is always a constant unit vector; hence only some elements may
933: (8)                    vary in a matrix. Below is how the elements map between
934: (8)                    different representations.
935: (4)                a b 0
936: (8)                    | c d 0 | normally represented as [ a b c d e f ].
937: (8)                    \ e f 1 /
938: (4)                """
939: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
940: (4)                __repr__ = _swig_repr
941: (4)                a = property(_mupdf.fz_matrix_a_get, _mupdf.fz_matrix_a_set)
942: (4)                b = property(_mupdf.fz_matrix_b_get, _mupdf.fz_matrix_b_set)
943: (4)                c = property(_mupdf.fz_matrix_c_get, _mupdf.fz_matrix_c_set)
944: (4)                d = property(_mupdf.fz_matrix_d_get, _mupdf.fz_matrix_d_set)
945: (4)                e = property(_mupdf.fz_matrix_e_get, _mupdf.fz_matrix_e_set)
946: (4)                f = property(_mupdf.fz_matrix_f_get, _mupdf.fz_matrix_f_set)
947: (4)                def __init__(self):
948: (8)                    _mupdf.fz_matrix_swiginit(self, _mupdf.new_fz_matrix())
949: (4)                __swig_destroy__ = _mupdf.delete_fz_matrix
950: (0)            _mupdf.fz_matrix_swigregister(fz_matrix)
951: (0)            cvar = _mupdf.cvar
952: (0)            fz_unit_rect = cvar.fz_unit_rect
953: (0)            fz_empty_rect = cvar.fz_empty_rect
954: (0)            fz_empty_irect = cvar.fz_empty_irect
955: (0)            fz_infinite_rect = cvar.fz_infinite_rect
956: (0)            fz_infinite_irect = cvar.fz_infinite_irect
957: (0)            class fz_quad(object):
958: (4)                r"""
959: (4)                A representation for a region defined by 4 points.
960: (4)                The significant difference between quads and rects is that
961: (4)                the edges of quads are not axis aligned.
962: (4)                """
963: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
964: (4)                __repr__ = _swig_repr
965: (4)                ul = property(_mupdf.fz_quad_ul_get, _mupdf.fz_quad_ul_set)
966: (4)                ur = property(_mupdf.fz_quad_ur_get, _mupdf.fz_quad_ur_set)
967: (4)                ll = property(_mupdf.fz_quad_ll_get, _mupdf.fz_quad_ll_set)
968: (4)                lr = property(_mupdf.fz_quad_lr_get, _mupdf.fz_quad_lr_set)
969: (4)                def __init__(self):
970: (8)                    _mupdf.fz_quad_swiginit(self, _mupdf.new_fz_quad())
971: (4)                __swig_destroy__ = _mupdf.delete_fz_quad
972: (0)            _mupdf.fz_quad_swigregister(fz_quad)
973: (0)            fz_identity = cvar.fz_identity
974: (0)            FZ_VERBOSE_EXCEPTIONS = _mupdf.FZ_VERBOSE_EXCEPTIONS
975: (0)            class fz_alloc_context(object):
976: (4)                r"""    Allocator structure; holds callbacks and private data pointer."""
977: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
978: (4)                __repr__ = _swig_repr
979: (4)                user = property(_mupdf.fz_alloc_context_user_get,
_mupdf.fz_alloc_context_user_set)
980: (4)                malloc = property(_mupdf.fz_alloc_context_malloc_get,
_mupdf.fz_alloc_context_malloc_set)
981: (4)                realloc = property(_mupdf.fz_alloc_context_realloc_get,
_mupdf.fz_alloc_context_realloc_set)
982: (4)                free = property(_mupdf.fz_alloc_context_free_get,
_mupdf.fz_alloc_context_free_set)
```

```
983: (4)                def __init__(self):
984: (8)                    _mupdf.fz_alloc_context_swiginit(self, _mupdf.new_fz_alloc_context())
985: (4)                __swig_destroy__ = _mupdf.delete_fz_alloc_context
986: (0)            _mupdf.fz_alloc_context_swigregister(fz_alloc_context)
987: (0)            FZ_ERROR_NONE = _mupdf.FZ_ERROR_NONE
988: (0)            FZ_ERROR_GENERIC = _mupdf.FZ_ERROR_GENERIC
989: (0)            FZ_ERROR_SYSTEM = _mupdf.FZ_ERROR_SYSTEM
990: (0)            FZ_ERROR_LIBRARY = _mupdf.FZ_ERROR_LIBRARY
991: (0)            FZ_ERROR_ARGUMENT = _mupdf.FZ_ERROR_ARGUMENT
992: (0)            FZ_ERROR_LIMIT = _mupdf.FZ_ERROR_LIMIT
993: (0)            FZ_ERROR_UNSUPPORTED = _mupdf.FZ_ERROR_UNSUPPORTED
994: (0)            FZ_ERROR_FORMAT = _mupdf.FZ_ERROR_FORMAT
995: (0)            FZ_ERROR_SYNTAX = _mupdf.FZ_ERROR_SYNTAX
996: (0)            FZ_ERROR_TRYLATER = _mupdf.FZ_ERROR_TRYLATER
997: (0)            FZ_ERROR_ABORT = _mupdf.FZ_ERROR_ABORT
998: (0)            FZ_ERROR_REPAIRED = _mupdf.FZ_ERROR_REPAIRED
999: (0)            class fz_locks_context(object):
1000: (4)               r"""
1001: (4)               Locking functions
1002: (4)               MuPDF is kept deliberately free of any knowledge of particular
1003: (4)               threading systems. As such, in order for safe multi-threaded
1004: (4)               operation, we rely on callbacks to client provided functions.
1005: (4)               A client is expected to provide FZ_LOCK_MAX number of mutexes,
1006: (4)               and a function to lock/unlock each of them. These may be
1007: (4)               recursive mutexes, but do not have to be.
1008: (4)               If a client does not intend to use multiple threads, then it
1009: (4)               may pass NULL instead of a lock structure.
1010: (4)               In order to avoid deadlocks, we have one simple rule
1011: (4)               internally as to how we use locks: We can never take lock n
1012: (4)               when we already hold any lock i, where 0 <= i <= n. In order
1013: (4)               to verify this, we have some debugging code, that can be
1014: (4)               enabled by defining FITZ_DEBUG_LOCKING.
1015: (4)               """
1016: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1017: (4)               __repr__ = _swig_repr
1018: (4)               user = property(_mupdf.fz_locks_context_user_get,
_mupdf.fz_locks_context_user_set)
1019: (4)               lock = property(_mupdf.fz_locks_context_lock_get,
_mupdf.fz_locks_context_lock_set)
1020: (4)               unlock = property(_mupdf.fz_locks_context_unlock_get,
_mupdf.fz_locks_context_unlock_set)
1021: (4)               def __init__(self):
1022: (8)                   _mupdf.fz_locks_context_swiginit(self, _mupdf.new_fz_locks_context())
1023: (4)               __swig_destroy__ = _mupdf.delete_fz_locks_context
1024: (0)           _mupdf.fz_locks_context_swigregister(fz_locks_context)
1025: (0)           FZ_LOCK_ALLOC = _mupdf.FZ_LOCK_ALLOC
1026: (0)           FZ_LOCK_FREETYPE = _mupdf.FZ_LOCK_FREETYPE
1027: (0)           FZ_LOCK_GLYPHCACHE = _mupdf.FZ_LOCK_GLYPHCACHE
1028: (0)           FZ_LOCK_MAX = _mupdf.FZ_LOCK_MAX
1029: (0)           FZ_STORE_UNLIMITED = _mupdf.FZ_STORE_UNLIMITED
1030: (0)           FZ_STORE_DEFAULT = _mupdf.FZ_STORE_DEFAULT
1031: (0)           class fz_string(object):
1032: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1033: (4)               __repr__ = _swig_repr
1034: (4)               refs = property(_mupdf.fz_string_refs_get, _mupdf.fz_string_refs_set)
1035: (4)               str = property(_mupdf.fz_string_str_get, _mupdf.fz_string_str_set)
1036: (4)               def __init__(self):
1037: (8)                   _mupdf.fz_string_swiginit(self, _mupdf.new_fz_string())
1038: (4)               __swig_destroy__ = _mupdf.delete_fz_string
1039: (0)           _mupdf.fz_string_swigregister(fz_string)
1040: (0)           FZ_JMPBUF_ALIGN = _mupdf.FZ_JMPBUF_ALIGN
1041: (0)           class fz_error_context(object):
1042: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1043: (4)               __repr__ = _swig_repr
1044: (4)               top = property(_mupdf.fz_error_context_top_get,
_mupdf.fz_error_context_top_set)
```

```
1045: (4)               stack = property(_mupdf.fz_error_context_stack_get,
_mupdf.fz_error_context_stack_set)
1046: (4)               padding = property(_mupdf.fz_error_context_padding_get,
_mupdf.fz_error_context_padding_set)
1047: (4)               stack_base = property(_mupdf.fz_error_context_stack_base_get,
_mupdf.fz_error_context_stack_base_set)
1048: (4)               errcode = property(_mupdf.fz_error_context_errcode_get,
_mupdf.fz_error_context_errcode_set)
1049: (4)               errnum = property(_mupdf.fz_error_context_errnum_get,
_mupdf.fz_error_context_errnum_set)
1050: (4)               print_user = property(_mupdf.fz_error_context_print_user_get,
_mupdf.fz_error_context_print_user_set)
1051: (4)               _print = property(_mupdf.fz_error_context__print_get,
_mupdf.fz_error_context__print_set)
1052: (4)               message = property(_mupdf.fz_error_context_message_get,
_mupdf.fz_error_context_message_set)
1053: (4)               def __init__(self):
1054: (8)                   _mupdf.fz_error_context_swiginit(self, _mupdf.new_fz_error_context())
1055: (4)               __swig_destroy__ = _mupdf.delete_fz_error_context
1056: (0)           _mupdf.fz_error_context_swigregister(fz_error_context)
1057: (0)           class fz_warn_context(object):
1058: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1059: (4)               __repr__ = _swig_repr
1060: (4)               print_user = property(_mupdf.fz_warn_context_print_user_get,
_mupdf.fz_warn_context_print_user_set)
1061: (4)               _print = property(_mupdf.fz_warn_context__print_get,
_mupdf.fz_warn_context__print_set)
1062: (4)               count = property(_mupdf.fz_warn_context_count_get,
_mupdf.fz_warn_context_count_set)
1063: (4)               message = property(_mupdf.fz_warn_context_message_get,
_mupdf.fz_warn_context_message_set)
1064: (4)               def __init__(self):
1065: (8)                   _mupdf.fz_warn_context_swiginit(self, _mupdf.new_fz_warn_context())
1066: (4)               __swig_destroy__ = _mupdf.delete_fz_warn_context
1067: (0)           _mupdf.fz_warn_context_swigregister(fz_warn_context)
1068: (0)           class fz_aa_context(object):
1069: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1070: (4)               __repr__ = _swig_repr
1071: (4)               hscale = property(_mupdf.fz_aa_context_hscale_get,
_mupdf.fz_aa_context_hscale_set)
1072: (4)               vscale = property(_mupdf.fz_aa_context_vscale_get,
_mupdf.fz_aa_context_vscale_set)
1073: (4)               scale = property(_mupdf.fz_aa_context_scale_get,
_mupdf.fz_aa_context_scale_set)
1074: (4)               bits = property(_mupdf.fz_aa_context_bits_get,
_mupdf.fz_aa_context_bits_set)
1075: (4)               text_bits = property(_mupdf.fz_aa_context_text_bits_get,
_mupdf.fz_aa_context_text_bits_set)
1076: (4)               min_line_width = property(_mupdf.fz_aa_context_min_line_width_get,
_mupdf.fz_aa_context_min_line_width_set)
1077: (4)               def __init__(self):
1078: (8)                   _mupdf.fz_aa_context_swiginit(self, _mupdf.new_fz_aa_context())
1079: (4)               __swig_destroy__ = _mupdf.delete_fz_aa_context
1080: (0)           _mupdf.fz_aa_context_swigregister(fz_aa_context)
1081: (0)           class fz_context(object):
1082: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1083: (4)               __repr__ = _swig_repr
1084: (4)               user = property(_mupdf.fz_context_user_get, _mupdf.fz_context_user_set)
1085: (4)               alloc = property(_mupdf.fz_context_alloc_get, _mupdf.fz_context_alloc_set)
1086: (4)               locks = property(_mupdf.fz_context_locks_get, _mupdf.fz_context_locks_set)
1087: (4)               error = property(_mupdf.fz_context_error_get, _mupdf.fz_context_error_set)
1088: (4)               warn = property(_mupdf.fz_context_warn_get, _mupdf.fz_context_warn_set)
1089: (4)               aa = property(_mupdf.fz_context_aa_get, _mupdf.fz_context_aa_set)
1090: (4)               seed48 = property(_mupdf.fz_context_seed48_get,
_mupdf.fz_context_seed48_set)
1091: (4)               icc_enabled = property(_mupdf.fz_context_icc_enabled_get,
```

```
_mupdf.fz_context_icc_enabled_set)
1092: (4)               throw_on_repair = property(_mupdf.fz_context_throw_on_repair_get,
_mupdf.fz_context_throw_on_repair_set)
1093: (4)               handler = property(_mupdf.fz_context_handler_get,
_mupdf.fz_context_handler_set)
1094: (4)               archive = property(_mupdf.fz_context_archive_get,
_mupdf.fz_context_archive_set)
1095: (4)               style = property(_mupdf.fz_context_style_get, _mupdf.fz_context_style_set)
1096: (4)               tuning = property(_mupdf.fz_context_tuning_get,
_mupdf.fz_context_tuning_set)
1097: (4)               stddbg = property(_mupdf.fz_context_stddbg_get,
_mupdf.fz_context_stddbg_set)
1098: (4)               font = property(_mupdf.fz_context_font_get, _mupdf.fz_context_font_set)
1099: (4)               colorspace = property(_mupdf.fz_context_colorspace_get,
_mupdf.fz_context_colorspace_set)
1100: (4)               store = property(_mupdf.fz_context_store_get, _mupdf.fz_context_store_set)
1101: (4)               glyph_cache = property(_mupdf.fz_context_glyph_cache_get,
_mupdf.fz_context_glyph_cache_set)
1102: (4)               def __init__(self):
1103: (8)                   _mupdf.fz_context_swiginit(self, _mupdf.new_fz_context())
1104: (4)               __swig_destroy__ = _mupdf.delete_fz_context
1105: (0)           _mupdf.fz_context_swigregister(fz_context)
1106: (0)           class fz_buffer(object):
1107: (4)               r"""
1108: (4)               fz_buffer is a wrapper around a dynamically allocated array of
1109: (4)               bytes.
1110: (4)               Buffers have a capacity (the number of bytes storage immediately
1111: (4)               available) and a current size.
1112: (4)               The contents of the structure are considered implementation
1113: (4)               details and are subject to change. Users should use the accessor
1114: (4)               functions in preference.
1115: (4)               """
1116: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1117: (4)               __repr__ = _swig_repr
1118: (4)               refs = property(_mupdf.fz_buffer_refs_get, _mupdf.fz_buffer_refs_set)
1119: (4)               data = property(_mupdf.fz_buffer_data_get, _mupdf.fz_buffer_data_set)
1120: (4)               cap = property(_mupdf.fz_buffer_cap_get, _mupdf.fz_buffer_cap_set)
1121: (4)               len = property(_mupdf.fz_buffer_len_get, _mupdf.fz_buffer_len_set)
1122: (4)               unused_bits = property(_mupdf.fz_buffer_unused_bits_get,
_mupdf.fz_buffer_unused_bits_set)
1123: (4)               shared = property(_mupdf.fz_buffer_shared_get,
_mupdf.fz_buffer_shared_set)
1124: (4)               def __init__(self):
1125: (8)                   _mupdf.fz_buffer_swiginit(self, _mupdf.new_fz_buffer())
1126: (4)               __swig_destroy__ = _mupdf.delete_fz_buffer
1127: (0)           _mupdf.fz_buffer_swigregister(fz_buffer)
1128: (0)           FZ_REPLACEMENT_CHARACTER = _mupdf.FZ_REPLACEMENT_CHARACTER
1129: (0)           FZ_UTFMAX = _mupdf.FZ_UTFMAX
1130: (0)           class fz_stream(object):
1131: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1132: (4)               __repr__ = _swig_repr
1133: (4)               refs = property(_mupdf.fz_stream_refs_get, _mupdf.fz_stream_refs_set)
1134: (4)               error = property(_mupdf.fz_stream_error_get, _mupdf.fz_stream_error_set)
1135: (4)               eof = property(_mupdf.fz_stream_eof_get, _mupdf.fz_stream_eof_set)
1136: (4)               progressive = property(_mupdf.fz_stream_progressive_get,
_mupdf.fz_stream_progressive_set)
1137: (4)               pos = property(_mupdf.fz_stream_pos_get, _mupdf.fz_stream_pos_set)
1138: (4)               avail = property(_mupdf.fz_stream_avail_get, _mupdf.fz_stream_avail_set)
1139: (4)               bits = property(_mupdf.fz_stream_bits_get, _mupdf.fz_stream_bits_set)
1140: (4)               rp = property(_mupdf.fz_stream_rp_get, _mupdf.fz_stream_rp_set)
1141: (4)               wp = property(_mupdf.fz_stream_wp_get, _mupdf.fz_stream_wp_set)
1142: (4)               state = property(_mupdf.fz_stream_state_get, _mupdf.fz_stream_state_set)
1143: (4)               next = property(_mupdf.fz_stream_next_get, _mupdf.fz_stream_next_set)
1144: (4)               drop = property(_mupdf.fz_stream_drop_get, _mupdf.fz_stream_drop_set)
1145: (4)               seek = property(_mupdf.fz_stream_seek_get, _mupdf.fz_stream_seek_set)
1146: (4)               def __init__(self):
1147: (8)                   _mupdf.fz_stream_swiginit(self, _mupdf.new_fz_stream())
```

```
1148: (4)               __swig_destroy__ = _mupdf.delete_fz_stream
1149: (0)           _mupdf.fz_stream_swigregister(fz_stream)
1150: (0)           class fz_output(object):
1151: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1152: (4)               __repr__ = _swig_repr
1153: (4)               state = property(_mupdf.fz_output_state_get, _mupdf.fz_output_state_set)
1154: (4)               write = property(_mupdf.fz_output_write_get, _mupdf.fz_output_write_set)
1155: (4)               seek = property(_mupdf.fz_output_seek_get, _mupdf.fz_output_seek_set)
1156: (4)               tell = property(_mupdf.fz_output_tell_get, _mupdf.fz_output_tell_set)
1157: (4)               close = property(_mupdf.fz_output_close_get, _mupdf.fz_output_close_set)
1158: (4)               drop = property(_mupdf.fz_output_drop_get, _mupdf.fz_output_drop_set)
1159: (4)               reset = property(_mupdf.fz_output_reset_get, _mupdf.fz_output_reset_set)
1160: (4)               as_stream = property(_mupdf.fz_output_as_stream_get,
_mupdf.fz_output_as_stream_set)
1161: (4)               truncate = property(_mupdf.fz_output_truncate_get,
_mupdf.fz_output_truncate_set)
1162: (4)               closed = property(_mupdf.fz_output_closed_get,
_mupdf.fz_output_closed_set)
1163: (4)               bp = property(_mupdf.fz_output_bp_get, _mupdf.fz_output_bp_set)
1164: (4)               wp = property(_mupdf.fz_output_wp_get, _mupdf.fz_output_wp_set)
1165: (4)               ep = property(_mupdf.fz_output_ep_get, _mupdf.fz_output_ep_set)
1166: (4)               buffered = property(_mupdf.fz_output_buffered_get,
_mupdf.fz_output_buffered_set)
1167: (4)               bits = property(_mupdf.fz_output_bits_get, _mupdf.fz_output_bits_set)
1168: (4)               def __init__(self):
1169: (8)                   _mupdf.fz_output_swiginit(self, _mupdf.new_fz_output())
1170: (4)               __swig_destroy__ = _mupdf.delete_fz_output
1171: (0)           _mupdf.fz_output_swigregister(fz_output)
1172: (0)           class fz_md5(object):
1173: (4)               r"""
1174: (4)               Structure definition is public to enable stack
1175: (4)               based allocation. Do not access the members directly.
1176: (4)               """
1177: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1178: (4)               __repr__ = _swig_repr
1179: (4)               lo = property(_mupdf.fz_md5_lo_get, _mupdf.fz_md5_lo_set)
1180: (4)               hi = property(_mupdf.fz_md5_hi_get, _mupdf.fz_md5_hi_set)
1181: (4)               a = property(_mupdf.fz_md5_a_get, _mupdf.fz_md5_a_set)
1182: (4)               b = property(_mupdf.fz_md5_b_get, _mupdf.fz_md5_b_set)
1183: (4)               c = property(_mupdf.fz_md5_c_get, _mupdf.fz_md5_c_set)
1184: (4)               d = property(_mupdf.fz_md5_d_get, _mupdf.fz_md5_d_set)
1185: (4)               buffer = property(_mupdf.fz_md5_buffer_get, _mupdf.fz_md5_buffer_set)
1186: (4)               def __init__(self):
1187: (8)                   _mupdf.fz_md5_swiginit(self, _mupdf.new_fz_md5())
1188: (4)               __swig_destroy__ = _mupdf.delete_fz_md5
1189: (0)           _mupdf.fz_md5_swigregister(fz_md5)
1190: (0)           class fz_sha256(object):
1191: (4)               r"""
1192: (4)               Structure definition is public to enable stack
1193: (4)               based allocation. Do not access the members directly.
1194: (4)               """
1195: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1196: (4)               __repr__ = _swig_repr
1197: (4)               state = property(_mupdf.fz_sha256_state_get, _mupdf.fz_sha256_state_set)
1198: (4)               count = property(_mupdf.fz_sha256_count_get, _mupdf.fz_sha256_count_set)
1199: (4)               def __init__(self):
1200: (8)                   _mupdf.fz_sha256_swiginit(self, _mupdf.new_fz_sha256())
1201: (4)               __swig_destroy__ = _mupdf.delete_fz_sha256
1202: (0)           _mupdf.fz_sha256_swigregister(fz_sha256)
1203: (0)           class fz_sha512(object):
1204: (4)               r"""
1205: (4)               Structure definition is public to enable stack
1206: (4)               based allocation. Do not access the members directly.
1207: (4)               """
1208: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
1209: (4)                __repr__ = _swig_repr
1210: (4)                state = property(_mupdf.fz_sha512_state_get, _mupdf.fz_sha512_state_set)
1211: (4)                count = property(_mupdf.fz_sha512_count_get, _mupdf.fz_sha512_count_set)
1212: (4)                def __init__(self):
1213: (8)                    _mupdf.fz_sha512_swiginit(self, _mupdf.new_fz_sha512())
1214: (4)                __swig_destroy__ = _mupdf.delete_fz_sha512
1215: (0)            _mupdf.fz_sha512_swigregister(fz_sha512)
1216: (0)            class fz_arc4(object):
1217: (4)                r"""
1218: (4)                Structure definition is public to enable stack
1219: (4)                based allocation. Do not access the members directly.
1220: (4)                """
1221: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1222: (4)                __repr__ = _swig_repr
1223: (4)                x = property(_mupdf.fz_arc4_x_get, _mupdf.fz_arc4_x_set)
1224: (4)                y = property(_mupdf.fz_arc4_y_get, _mupdf.fz_arc4_y_set)
1225: (4)                state = property(_mupdf.fz_arc4_state_get, _mupdf.fz_arc4_state_set)
1226: (4)                def __init__(self):
1227: (8)                    _mupdf.fz_arc4_swiginit(self, _mupdf.new_fz_arc4())
1228: (4)                __swig_destroy__ = _mupdf.delete_fz_arc4
1229: (0)            _mupdf.fz_arc4_swigregister(fz_arc4)
1230: (0)            class fz_aes(object):
1231: (4)                r"""
1232: (4)                Structure definitions are public to enable stack
1233: (4)                based allocation. Do not access the members directly.
1234: (4)                """
1235: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1236: (4)                __repr__ = _swig_repr
1237: (4)                nr = property(_mupdf.fz_aes_nr_get, _mupdf.fz_aes_nr_set)
1238: (4)                rk = property(_mupdf.fz_aes_rk_get, _mupdf.fz_aes_rk_set)
1239: (4)                buf = property(_mupdf.fz_aes_buf_get, _mupdf.fz_aes_buf_set)
1240: (4)                def __init__(self):
1241: (8)                    _mupdf.fz_aes_swiginit(self, _mupdf.new_fz_aes())
1242: (4)                __swig_destroy__ = _mupdf.delete_fz_aes
1243: (0)            _mupdf.fz_aes_swigregister(fz_aes)
1244: (0)            FZ_AES_DECRYPT = _mupdf.FZ_AES_DECRYPT
1245: (0)            FZ_AES_ENCRYPT = _mupdf.FZ_AES_ENCRYPT
1246: (0)            class fz_getopt_long_options(object):
1247: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1248: (4)                __repr__ = _swig_repr
1249: (4)                option = property(_mupdf.fz_getopt_long_options_option_get,
_mupdf.fz_getopt_long_options_option_set)
1250: (4)                flag = property(_mupdf.fz_getopt_long_options_flag_get,
_mupdf.fz_getopt_long_options_flag_set)
1251: (4)                opaque = property(_mupdf.fz_getopt_long_options_opaque_get,
_mupdf.fz_getopt_long_options_opaque_set)
1252: (4)                def __init__(self):
1253: (8)                    _mupdf.fz_getopt_long_options_swiginit(self,
_mupdf.new_fz_getopt_long_options())
1254: (4)                __swig_destroy__ = _mupdf.delete_fz_getopt_long_options
1255: (0)            _mupdf.fz_getopt_long_options_swigregister(fz_getopt_long_options)
1256: (0)            FZ_HASH_TABLE_KEY_LENGTH = _mupdf.FZ_HASH_TABLE_KEY_LENGTH
1257: (0)            FZ_BIDI_LTR = _mupdf.FZ_BIDI_LTR
1258: (0)            FZ_BIDI_RTL = _mupdf.FZ_BIDI_RTL
1259: (0)            FZ_BIDI_NEUTRAL = _mupdf.FZ_BIDI_NEUTRAL
1260: (0)            FZ_BIDI_CLASSIFY_WHITE_SPACE = _mupdf.FZ_BIDI_CLASSIFY_WHITE_SPACE
1261: (0)            FZ_BIDI_REPLACE_TAB = _mupdf.FZ_BIDI_REPLACE_TAB
1262: (0)            class fz_archive_handler(object):
1263: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1264: (4)                __repr__ = _swig_repr
1265: (4)                recognize = property(_mupdf.fz_archive_handler_recognize_get,
_mupdf.fz_archive_handler_recognize_set)
1266: (4)                open = property(_mupdf.fz_archive_handler_open_get,
_mupdf.fz_archive_handler_open_set)
1267: (4)                def __init__(self):
```

```
1268: (8)                    _mupdf.fz_archive_handler_swiginit(self,
_mupdf.new_fz_archive_handler())
1269: (4)                __swig_destroy__ = _mupdf.delete_fz_archive_handler
1270: (0)            _mupdf.fz_archive_handler_swigregister(fz_archive_handler)
1271: (0)            class fz_archive(object):
1272: (4)                r"""    Implementation details: Subject to change."""
1273: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1274: (4)                __repr__ = _swig_repr
1275: (4)                refs = property(_mupdf.fz_archive_refs_get, _mupdf.fz_archive_refs_set)
1276: (4)                file = property(_mupdf.fz_archive_file_get, _mupdf.fz_archive_file_set)
1277: (4)                format = property(_mupdf.fz_archive_format_get,
_mupdf.fz_archive_format_set)
1278: (4)                drop_archive = property(_mupdf.fz_archive_drop_archive_get,
_mupdf.fz_archive_drop_archive_set)
1279: (4)                count_entries = property(_mupdf.fz_archive_count_entries_get,
_mupdf.fz_archive_count_entries_set)
1280: (4)                list_entry = property(_mupdf.fz_archive_list_entry_get,
_mupdf.fz_archive_list_entry_set)
1281: (4)                has_entry = property(_mupdf.fz_archive_has_entry_get,
_mupdf.fz_archive_has_entry_set)
1282: (4)                read_entry = property(_mupdf.fz_archive_read_entry_get,
_mupdf.fz_archive_read_entry_set)
1283: (4)                open_entry = property(_mupdf.fz_archive_open_entry_get,
_mupdf.fz_archive_open_entry_set)
1284: (4)                def __init__(self):
1285: (8)                    _mupdf.fz_archive_swiginit(self, _mupdf.new_fz_archive())
1286: (4)                __swig_destroy__ = _mupdf.delete_fz_archive
1287: (0)            _mupdf.fz_archive_swigregister(fz_archive)
1288: (0)            fz_libarchive_archive_handler = cvar.fz_libarchive_archive_handler
1289: (0)            class fz_storable(object):
1290: (4)                r"""
1291: (4)                Any storable object should include an fz_storable structure
1292: (4)                at the start (by convention at least) of their structure.
1293: (4)                (Unless it starts with an fz_key_storable, see below).
1294: (4)                """
1295: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1296: (4)                __repr__ = _swig_repr
1297: (4)                refs = property(_mupdf.fz_storable_refs_get, _mupdf.fz_storable_refs_set)
1298: (4)                drop = property(_mupdf.fz_storable_drop_get, _mupdf.fz_storable_drop_set)
1299: (4)                droppable = property(_mupdf.fz_storable_droppable_get,
_mupdf.fz_storable_droppable_set)
1300: (4)                def __init__(self):
1301: (8)                    _mupdf.fz_storable_swiginit(self, _mupdf.new_fz_storable())
1302: (4)                __swig_destroy__ = _mupdf.delete_fz_storable
1303: (0)            _mupdf.fz_storable_swigregister(fz_storable)
1304: (0)            class fz_key_storable(object):
1305: (4)                r"""
1306: (4)                Any storable object that can appear in the key of another
1307: (4)                storable object should include an fz_key_storable structure
1308: (4)                at the start (by convention at least) of their structure.
1309: (4)                """
1310: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1311: (4)                __repr__ = _swig_repr
1312: (4)                storable = property(_mupdf.fz_key_storable_storable_get,
_mupdf.fz_key_storable_storable_set)
1313: (4)                store_key_refs = property(_mupdf.fz_key_storable_store_key_refs_get,
_mupdf.fz_key_storable_store_key_refs_set)
1314: (4)                def __init__(self):
1315: (8)                    _mupdf.fz_key_storable_swiginit(self, _mupdf.new_fz_key_storable())
1316: (4)                __swig_destroy__ = _mupdf.delete_fz_key_storable
1317: (0)            _mupdf.fz_key_storable_swigregister(fz_key_storable)
1318: (0)            class fz_store_hash(object):
1319: (4)                r"""
1320: (4)                The store can be seen as a dictionary that maps keys to
1321: (4)                fz_storable values. In order to allow keys of different types to
1322: (4)                be stored, we have a structure full of functions for each key
```

```
1323: (4)               'type'; this fz_store_type pointer is stored with each key, and
1324: (4)               tells the store how to perform certain operations (like taking/
1325: (4)               dropping a reference, comparing two keys, outputting details for
1326: (4)               debugging etc).
1327: (4)               The store uses a hash table internally for speed where possible.
1328: (4)               In order for this to work, we need a mechanism for turning a
1329: (4)               generic 'key' into 'a hashable string'. For this purpose the
1330: (4)               type structure contains a make_hash_key function pointer that
1331: (4)               maps from a void * to a fz_store_hash structure. If
1332: (4)               make_hash_key function returns 0, then the key is determined not
1333: (4)               to be hashable, and the value is not stored in the hash table.
1334: (4)               Some objects can be used both as values within the store, and as
1335: (4)               a component of keys within the store. We refer to these objects
1336: (4)               as "key storable" objects. In this case, we need to take
1337: (4)               additional care to ensure that we do not end up keeping an item
1338: (4)               within the store, purely because its value is referred to by
1339: (4)               another key in the store.
1340: (4)               An example of this are fz_images in PDF files. Each fz_image is
1341: (4)               placed into the    store to enable it to be easily reused. When the
1342: (4)               image is rendered, a pixmap is generated from the image, and the
1343: (4)               pixmap is placed into the store so it can be reused on
1344: (4)               subsequent renders. The image forms part of the key for the
1345: (4)               pixmap.
1346: (4)               When we close the pdf document (and any associated pages/display
1347: (4)               lists etc), we drop the images from the store. This may leave us
1348: (4)               in the position of the images having non-zero reference counts
1349: (4)               purely because they are used as part of the keys for the
1350: (4)               pixmaps.
1351: (4)               We therefore use special reference counting functions to keep
1352: (4)               track of these "key storable" items, and hence store the number
1353: (4)               of references to these items that are used in keys.
1354: (4)               When the number of references to an object == the number of
1355: (4)               references to an object from keys in the store, we know that we
1356: (4)               can remove all the items which have that object as part of the
1357: (4)               key. This is done by running a pass over the store, 'reaping'
1358: (4)               those items.
1359: (4)               Reap passes are slower than we would like as they touch every
1360: (4)               item in the store. We therefore provide a way to 'batch' such
1361: (4)               reap passes together, using fz_defer_reap_start/
1362: (4)               fz_defer_reap_end to bracket a region in which many may be
1363: (4)               triggered.
1364: (4)               """
1365: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1366: (4)               __repr__ = _swig_repr
1367: (4)               drop = property(_mupdf.fz_store_hash_drop_get,
_mupdf.fz_store_hash_drop_set)
1368: (4)               def __init__(self):
1369: (8)                   _mupdf.fz_store_hash_swiginit(self, _mupdf.new_fz_store_hash())
1370: (4)               __swig_destroy__ = _mupdf.delete_fz_store_hash
1371: (0)           _mupdf.fz_store_hash_swigregister(fz_store_hash)
1372: (0)           class fz_store_type(object):
1373: (4)               r"""
1374: (4)               Every type of object to be placed into the store defines an
1375: (4)               fz_store_type. This contains the pointers to functions to
1376: (4)               make hashes, manipulate keys, and check for needing reaping.
1377: (4)               """
1378: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1379: (4)               __repr__ = _swig_repr
1380: (4)               name = property(_mupdf.fz_store_type_name_get,
_mupdf.fz_store_type_name_set)
1381: (4)               make_hash_key = property(_mupdf.fz_store_type_make_hash_key_get,
_mupdf.fz_store_type_make_hash_key_set)
1382: (4)               keep_key = property(_mupdf.fz_store_type_keep_key_get,
_mupdf.fz_store_type_keep_key_set)
1383: (4)               drop_key = property(_mupdf.fz_store_type_drop_key_get,
_mupdf.fz_store_type_drop_key_set)
1384: (4)               cmp_key = property(_mupdf.fz_store_type_cmp_key_get,
```

```
_mupdf.fz_store_type_cmp_key_set)
1385: (4)               format_key = property(_mupdf.fz_store_type_format_key_get,
_mupdf.fz_store_type_format_key_set)
1386: (4)               needs_reap = property(_mupdf.fz_store_type_needs_reap_get,
_mupdf.fz_store_type_needs_reap_set)
1387: (4)               def __init__(self):
1388: (8)                   _mupdf.fz_store_type_swiginit(self, _mupdf.new_fz_store_type())
1389: (4)               __swig_destroy__ = _mupdf.delete_fz_store_type
1390: (0)           _mupdf.fz_store_type_swigregister(fz_store_type)
1391: (0)           FZ_RI_PERCEPTUAL = _mupdf.FZ_RI_PERCEPTUAL
1392: (0)           FZ_RI_RELATIVE_COLORIMETRIC = _mupdf.FZ_RI_RELATIVE_COLORIMETRIC
1393: (0)           FZ_RI_SATURATION = _mupdf.FZ_RI_SATURATION
1394: (0)           FZ_RI_ABSOLUTE_COLORIMETRIC = _mupdf.FZ_RI_ABSOLUTE_COLORIMETRIC
1395: (0)           class fz_color_params(object):
1396: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1397: (4)               __repr__ = _swig_repr
1398: (4)               ri = property(_mupdf.fz_color_params_ri_get,
_mupdf.fz_color_params_ri_set)
1399: (4)               bp = property(_mupdf.fz_color_params_bp_get,
_mupdf.fz_color_params_bp_set)
1400: (4)               op = property(_mupdf.fz_color_params_op_get,
_mupdf.fz_color_params_op_set)
1401: (4)               opm = property(_mupdf.fz_color_params_opm_get,
_mupdf.fz_color_params_opm_set)
1402: (4)               def __init__(self):
1403: (8)                   _mupdf.fz_color_params_swiginit(self, _mupdf.new_fz_color_params())
1404: (4)               __swig_destroy__ = _mupdf.delete_fz_color_params
1405: (0)           _mupdf.fz_color_params_swigregister(fz_color_params)
1406: (0)           FZ_MAX_COLORS = _mupdf.FZ_MAX_COLORS
1407: (0)           FZ_COLORSPACE_IS_DEVICE = _mupdf.FZ_COLORSPACE_IS_DEVICE
1408: (0)           FZ_COLORSPACE_IS_ICC = _mupdf.FZ_COLORSPACE_IS_ICC
1409: (0)           FZ_COLORSPACE_HAS_CMYK = _mupdf.FZ_COLORSPACE_HAS_CMYK
1410: (0)           FZ_COLORSPACE_HAS_SPOTS = _mupdf.FZ_COLORSPACE_HAS_SPOTS
1411: (0)           FZ_COLORSPACE_HAS_CMYK_AND_SPOTS = _mupdf.FZ_COLORSPACE_HAS_CMYK_AND_SPOTS
1412: (0)           class fz_default_colorspaces(object):
1413: (4)               r"""    Structure to hold default colorspaces."""
1414: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1415: (4)               __repr__ = _swig_repr
1416: (4)               refs = property(_mupdf.fz_default_colorspaces_refs_get,
_mupdf.fz_default_colorspaces_refs_set)
1417: (4)               gray = property(_mupdf.fz_default_colorspaces_gray_get,
_mupdf.fz_default_colorspaces_gray_set)
1418: (4)               rgb = property(_mupdf.fz_default_colorspaces_rgb_get,
_mupdf.fz_default_colorspaces_rgb_set)
1419: (4)               cmyk = property(_mupdf.fz_default_colorspaces_cmyk_get,
_mupdf.fz_default_colorspaces_cmyk_set)
1420: (4)               oi = property(_mupdf.fz_default_colorspaces_oi_get,
_mupdf.fz_default_colorspaces_oi_set)
1421: (4)               def __init__(self):
1422: (8)                   _mupdf.fz_default_colorspaces_swiginit(self,
_mupdf.new_fz_default_colorspaces())
1423: (4)               __swig_destroy__ = _mupdf.delete_fz_default_colorspaces
1424: (0)           _mupdf.fz_default_colorspaces_swigregister(fz_default_colorspaces)
1425: (0)           fz_default_color_params = cvar.fz_default_color_params
1426: (0)           class fz_colorspace(object):
1427: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1428: (4)               __repr__ = _swig_repr
1429: (4)               key_storable = property(_mupdf.fz_colorspace_key_storable_get,
_mupdf.fz_colorspace_key_storable_set)
1430: (4)               type = property(_mupdf.fz_colorspace_type_get,
_mupdf.fz_colorspace_type_set)
1431: (4)               flags = property(_mupdf.fz_colorspace_flags_get,
_mupdf.fz_colorspace_flags_set)
1432: (4)               n = property(_mupdf.fz_colorspace_n_get, _mupdf.fz_colorspace_n_set)
1433: (4)               name = property(_mupdf.fz_colorspace_name_get,
_mupdf.fz_colorspace_name_set)
```

```
1434: (4)               def __init__(self):
1435: (8)                   _mupdf.fz_colorspace_swiginit(self, _mupdf.new_fz_colorspace())
1436: (4)               __swig_destroy__ = _mupdf.delete_fz_colorspace
1437: (0)           _mupdf.fz_colorspace_swigregister(fz_colorspace)
1438: (0)           FZ_MAX_SEPARATIONS = _mupdf.FZ_MAX_SEPARATIONS
1439: (0)           FZ_SEPARATION_COMPOSITE = _mupdf.FZ_SEPARATION_COMPOSITE
1440: (0)           FZ_SEPARATION_SPOT = _mupdf.FZ_SEPARATION_SPOT
1441: (0)           FZ_SEPARATION_DISABLED = _mupdf.FZ_SEPARATION_DISABLED
1442: (0)           class fz_pixmap(object):
1443: (4)               r"""
1444: (4)               Pixmaps represent a set of pixels for a 2 dimensional region of
1445: (4)               a plane. Each pixel has n components per pixel. The components
1446: (4)               are in the order process-components, spot-colors, alpha, where
1447: (4)               there can be 0 of any of those types. The data is in
1448: (4)               premultiplied alpha when rendering, but non-premultiplied for
1449: (4)               colorspace conversions and rescaling.
1450: (4)               x, y: The minimum x and y coord of the region in pixels.
1451: (4)               w, h: The width and height of the region in pixels.
1452: (4)               n: The number of color components in the image.
1453: (8)                   n = num composite colors + num spots + num alphas
1454: (4)               s: The number of spot channels in the image.
1455: (4)               alpha: 0 for no alpha, 1 for alpha present.
1456: (4)               flags: flag bits.
1457: (8)                   Bit 0: If set, draw the image with linear interpolation.
1458: (8)                   Bit 1: If set, free the samples buffer when the pixmap
1459: (8)                   is destroyed.
1460: (4)               stride: The byte offset from the data for any given pixel
1461: (4)               to the data for the same pixel on the row below.
1462: (4)               seps: NULL, or a pointer to a separations structure. If NULL,
1463: (4)               s should be 0.
1464: (4)               xres, yres: Image resolution in dpi. Default is 96 dpi.
1465: (4)               colorspace: Pointer to a colorspace object describing the
1466: (4)               colorspace the pixmap is in. If NULL, the image is a mask.
1467: (4)               samples: Pointer to the first byte of the pixmap sample data.
1468: (4)               This is typically a simple block of memory w * h * n bytes of
1469: (4)               memory in which the components are stored linearly, but with the
1470: (4)               use of appropriate stride values, scanlines can be stored in
1471: (4)               different orders, and have different amounts of padding. The
1472: (4)               first n bytes are components 0 to n-1 for the pixel at (x,y).
1473: (4)               Each successive n bytes gives another pixel in scanline order
1474: (4)               as we move across the line. The start of each scanline is offset
1475: (4)               the start of the previous one by stride bytes.
1476: (4)               """
1477: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1478: (4)               __repr__ = _swig_repr
1479: (4)               storable = property(_mupdf.fz_pixmap_storable_get,
_mupdf.fz_pixmap_storable_set)
1480: (4)               x = property(_mupdf.fz_pixmap_x_get, _mupdf.fz_pixmap_x_set)
1481: (4)               y = property(_mupdf.fz_pixmap_y_get, _mupdf.fz_pixmap_y_set)
1482: (4)               w = property(_mupdf.fz_pixmap_w_get, _mupdf.fz_pixmap_w_set)
1483: (4)               h = property(_mupdf.fz_pixmap_h_get, _mupdf.fz_pixmap_h_set)
1484: (4)               n = property(_mupdf.fz_pixmap_n_get, _mupdf.fz_pixmap_n_set)
1485: (4)               s = property(_mupdf.fz_pixmap_s_get, _mupdf.fz_pixmap_s_set)
1486: (4)               alpha = property(_mupdf.fz_pixmap_alpha_get, _mupdf.fz_pixmap_alpha_set)
1487: (4)               flags = property(_mupdf.fz_pixmap_flags_get, _mupdf.fz_pixmap_flags_set)
1488: (4)               stride = property(_mupdf.fz_pixmap_stride_get,
_mupdf.fz_pixmap_stride_set)
1489: (4)               seps = property(_mupdf.fz_pixmap_seps_get, _mupdf.fz_pixmap_seps_set)
1490: (4)               xres = property(_mupdf.fz_pixmap_xres_get, _mupdf.fz_pixmap_xres_set)
1491: (4)               yres = property(_mupdf.fz_pixmap_yres_get, _mupdf.fz_pixmap_yres_set)
1492: (4)               colorspace = property(_mupdf.fz_pixmap_colorspace_get,
_mupdf.fz_pixmap_colorspace_set)
1493: (4)               samples = property(_mupdf.fz_pixmap_samples_get,
_mupdf.fz_pixmap_samples_set)
1494: (4)               underlying = property(_mupdf.fz_pixmap_underlying_get,
_mupdf.fz_pixmap_underlying_set)
1495: (4)               def __init__(self):
1496: (8)                   _mupdf.fz_pixmap_swiginit(self, _mupdf.new_fz_pixmap())
```

```
1497: (4)                __swig_destroy__ = _mupdf.delete_fz_pixmap
1498: (0)            _mupdf.fz_pixmap_swigregister(fz_pixmap)
1499: (0)            FZ_PIXMAP_FLAG_INTERPOLATE = _mupdf.FZ_PIXMAP_FLAG_INTERPOLATE
1500: (0)            FZ_PIXMAP_FLAG_FREE_SAMPLES = _mupdf.FZ_PIXMAP_FLAG_FREE_SAMPLES
1501: (0)            FZ_DEFLATE_NONE = _mupdf.FZ_DEFLATE_NONE
1502: (0)            FZ_DEFLATE_BEST_SPEED = _mupdf.FZ_DEFLATE_BEST_SPEED
1503: (0)            FZ_DEFLATE_BEST = _mupdf.FZ_DEFLATE_BEST
1504: (0)            FZ_DEFLATE_DEFAULT = _mupdf.FZ_DEFLATE_DEFAULT
1505: (0)            class fz_range(object):
1506: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1507: (4)                __repr__ = _swig_repr
1508: (4)                offset = property(_mupdf.fz_range_offset_get, _mupdf.fz_range_offset_set)
1509: (4)                length = property(_mupdf.fz_range_length_get, _mupdf.fz_range_length_set)
1510: (4)                def __init__(self):
1511: (8)                    _mupdf.fz_range_swiginit(self, _mupdf.new_fz_range())
1512: (4)                __swig_destroy__ = _mupdf.delete_fz_range
1513: (0)            _mupdf.fz_range_swigregister(fz_range)
1514: (0)            class fz_compression_params(object):
1515: (4)                r"""
1516: (4)                Compression parameters used for buffers of compressed data;
1517: (4)                typically for the source data for images.
1518: (4)                """
1519: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1520: (4)                __repr__ = _swig_repr
1521: (4)                type = property(_mupdf.fz_compression_params_type_get,
_mupdf.fz_compression_params_type_set)
1522: (4)                def __init__(self):
1523: (8)                    _mupdf.fz_compression_params_swiginit(self,
_mupdf.new_fz_compression_params())
1524: (4)                __swig_destroy__ = _mupdf.delete_fz_compression_params
1525: (0)            _mupdf.fz_compression_params_swigregister(fz_compression_params)
1526: (0)            class fz_compressed_buffer(object):
1527: (4)                r"""
1528: (4)                Buffers of compressed data; typically for the source data
1529: (4)                for images.
1530: (4)                """
1531: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1532: (4)                __repr__ = _swig_repr
1533: (4)                refs = property(_mupdf.fz_compressed_buffer_refs_get,
_mupdf.fz_compressed_buffer_refs_set)
1534: (4)                params = property(_mupdf.fz_compressed_buffer_params_get,
_mupdf.fz_compressed_buffer_params_set)
1535: (4)                buffer = property(_mupdf.fz_compressed_buffer_buffer_get,
_mupdf.fz_compressed_buffer_buffer_set)
1536: (4)                def __init__(self):
1537: (8)                    _mupdf.fz_compressed_buffer_swiginit(self,
_mupdf.new_fz_compressed_buffer())
1538: (4)                __swig_destroy__ = _mupdf.delete_fz_compressed_buffer
1539: (0)            _mupdf.fz_compressed_buffer_swigregister(fz_compressed_buffer)
1540: (0)            FZ_IMAGE_UNKNOWN = _mupdf.FZ_IMAGE_UNKNOWN
1541: (0)            FZ_IMAGE_RAW = _mupdf.FZ_IMAGE_RAW
1542: (0)            FZ_IMAGE_FAX = _mupdf.FZ_IMAGE_FAX
1543: (0)            FZ_IMAGE_FLATE = _mupdf.FZ_IMAGE_FLATE
1544: (0)            FZ_IMAGE_LZW = _mupdf.FZ_IMAGE_LZW
1545: (0)            FZ_IMAGE_RLD = _mupdf.FZ_IMAGE_RLD
1546: (0)            FZ_IMAGE_BMP = _mupdf.FZ_IMAGE_BMP
1547: (0)            FZ_IMAGE_GIF = _mupdf.FZ_IMAGE_GIF
1548: (0)            FZ_IMAGE_JBIG2 = _mupdf.FZ_IMAGE_JBIG2
1549: (0)            FZ_IMAGE_JPEG = _mupdf.FZ_IMAGE_JPEG
1550: (0)            FZ_IMAGE_JPX = _mupdf.FZ_IMAGE_JPX
1551: (0)            FZ_IMAGE_JXR = _mupdf.FZ_IMAGE_JXR
1552: (0)            FZ_IMAGE_PNG = _mupdf.FZ_IMAGE_PNG
1553: (0)            FZ_IMAGE_PNM = _mupdf.FZ_IMAGE_PNM
1554: (0)            FZ_IMAGE_TIFF = _mupdf.FZ_IMAGE_TIFF
1555: (0)            FZ_IMAGE_PSD = _mupdf.FZ_IMAGE_PSD
1556: (0)            class fz_bitmap(object):
```

```
1557: (4)               r"""
1558: (4)               Bitmaps have 1 bit per component. Only used for creating
1559: (4)               halftoned versions of contone buffers, and saving out. Samples
1560: (4)               are stored msb first, akin to pbms.
1561: (4)               The internals of this struct are considered implementation
1562: (4)               details and subject to change. Where possible, accessor
1563: (4)               functions should be used in preference.
1564: (4)               """
1565: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1566: (4)               __repr__ = _swig_repr
1567: (4)               refs = property(_mupdf.fz_bitmap_refs_get, _mupdf.fz_bitmap_refs_set)
1568: (4)               w = property(_mupdf.fz_bitmap_w_get, _mupdf.fz_bitmap_w_set)
1569: (4)               h = property(_mupdf.fz_bitmap_h_get, _mupdf.fz_bitmap_h_set)
1570: (4)               stride = property(_mupdf.fz_bitmap_stride_get,
_mupdf.fz_bitmap_stride_set)
1571: (4)               n = property(_mupdf.fz_bitmap_n_get, _mupdf.fz_bitmap_n_set)
1572: (4)               xres = property(_mupdf.fz_bitmap_xres_get, _mupdf.fz_bitmap_xres_set)
1573: (4)               yres = property(_mupdf.fz_bitmap_yres_get, _mupdf.fz_bitmap_yres_set)
1574: (4)               samples = property(_mupdf.fz_bitmap_samples_get,
_mupdf.fz_bitmap_samples_set)
1575: (4)               def __init__(self):
1576: (8)                   _mupdf.fz_bitmap_swiginit(self, _mupdf.new_fz_bitmap())
1577: (4)               __swig_destroy__ = _mupdf.delete_fz_bitmap
1578: (0)           _mupdf.fz_bitmap_swigregister(fz_bitmap)
1579: (0)           class fz_image(object):
1580: (4)               r"""
1581: (4)               Structure is public to allow other structures to
1582: (4)               be derived from it. Do not access members directly.
1583: (4)               """
1584: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1585: (4)               __repr__ = _swig_repr
1586: (4)               key_storable = property(_mupdf.fz_image_key_storable_get,
_mupdf.fz_image_key_storable_set)
1587: (4)               w = property(_mupdf.fz_image_w_get, _mupdf.fz_image_w_set)
1588: (4)               h = property(_mupdf.fz_image_h_get, _mupdf.fz_image_h_set)
1589: (4)               n = property(_mupdf.fz_image_n_get, _mupdf.fz_image_n_set)
1590: (4)               bpc = property(_mupdf.fz_image_bpc_get, _mupdf.fz_image_bpc_set)
1591: (4)               imagemask = property(_mupdf.fz_image_imagemask_get,
_mupdf.fz_image_imagemask_set)
1592: (4)               interpolate = property(_mupdf.fz_image_interpolate_get,
_mupdf.fz_image_interpolate_set)
1593: (4)               use_colorkey = property(_mupdf.fz_image_use_colorkey_get,
_mupdf.fz_image_use_colorkey_set)
1594: (4)               use_decode = property(_mupdf.fz_image_use_decode_get,
_mupdf.fz_image_use_decode_set)
1595: (4)               decoded = property(_mupdf.fz_image_decoded_get,
_mupdf.fz_image_decoded_set)
1596: (4)               scalable = property(_mupdf.fz_image_scalable_get,
_mupdf.fz_image_scalable_set)
1597: (4)               orientation = property(_mupdf.fz_image_orientation_get,
_mupdf.fz_image_orientation_set)
1598: (4)               mask = property(_mupdf.fz_image_mask_get, _mupdf.fz_image_mask_set)
1599: (4)               xres = property(_mupdf.fz_image_xres_get, _mupdf.fz_image_xres_set)
1600: (4)               yres = property(_mupdf.fz_image_yres_get, _mupdf.fz_image_yres_set)
1601: (4)               colorspace = property(_mupdf.fz_image_colorspace_get,
_mupdf.fz_image_colorspace_set)
1602: (4)               drop_image = property(_mupdf.fz_image_drop_image_get,
_mupdf.fz_image_drop_image_set)
1603: (4)               get_pixmap = property(_mupdf.fz_image_get_pixmap_get,
_mupdf.fz_image_get_pixmap_set)
1604: (4)               get_size = property(_mupdf.fz_image_get_size_get,
_mupdf.fz_image_get_size_set)
1605: (4)               colorkey = property(_mupdf.fz_image_colorkey_get,
_mupdf.fz_image_colorkey_set)
1606: (4)               decode = property(_mupdf.fz_image_decode_get, _mupdf.fz_image_decode_set)
1607: (4)               def __init__(self):
1608: (8)                   _mupdf.fz_image_swiginit(self, _mupdf.new_fz_image())
```

```
1609: (4)                __swig_destroy__ = _mupdf.delete_fz_image
1610: (0)            _mupdf.fz_image_swigregister(fz_image)
1611: (0)            FZ_FUNCTION_BASED = _mupdf.FZ_FUNCTION_BASED
1612: (0)            FZ_LINEAR = _mupdf.FZ_LINEAR
1613: (0)            FZ_RADIAL = _mupdf.FZ_RADIAL
1614: (0)            FZ_MESH_TYPE4 = _mupdf.FZ_MESH_TYPE4
1615: (0)            FZ_MESH_TYPE5 = _mupdf.FZ_MESH_TYPE5
1616: (0)            FZ_MESH_TYPE6 = _mupdf.FZ_MESH_TYPE6
1617: (0)            FZ_MESH_TYPE7 = _mupdf.FZ_MESH_TYPE7
1618: (0)            class fz_shade(object):
1619: (4)                r"""
1620: (4)                Structure is public to allow derived classes. Do not
1621: (4)                access the members directly.
1622: (4)                """
1623: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1624: (4)                __repr__ = _swig_repr
1625: (4)                storable = property(_mupdf.fz_shade_storable_get,
_mupdf.fz_shade_storable_set)
1626: (4)                bbox = property(_mupdf.fz_shade_bbox_get, _mupdf.fz_shade_bbox_set)
1627: (4)                colorspace = property(_mupdf.fz_shade_colorspace_get,
_mupdf.fz_shade_colorspace_set)
1628: (4)                matrix = property(_mupdf.fz_shade_matrix_get, _mupdf.fz_shade_matrix_set)
1629: (4)                use_background = property(_mupdf.fz_shade_use_background_get,
_mupdf.fz_shade_use_background_set)
1630: (4)                background = property(_mupdf.fz_shade_background_get,
_mupdf.fz_shade_background_set)
1631: (4)                function_stride = property(_mupdf.fz_shade_function_stride_get,
_mupdf.fz_shade_function_stride_set)
1632: (4)                function = property(_mupdf.fz_shade_function_get,
_mupdf.fz_shade_function_set)
1633: (4)                type = property(_mupdf.fz_shade_type_get, _mupdf.fz_shade_type_set)
1634: (4)                buffer = property(_mupdf.fz_shade_buffer_get, _mupdf.fz_shade_buffer_set)
1635: (4)                def __init__(self):
1636: (8)                    _mupdf.fz_shade_swiginit(self, _mupdf.new_fz_shade())
1637: (4)                __swig_destroy__ = _mupdf.delete_fz_shade
1638: (0)            _mupdf.fz_shade_swigregister(fz_shade)
1639: (0)            class fz_vertex(object):
1640: (4)                r"""    Handy routine for processing mesh based shades"""
1641: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1642: (4)                __repr__ = _swig_repr
1643: (4)                p = property(_mupdf.fz_vertex_p_get, _mupdf.fz_vertex_p_set)
1644: (4)                c = property(_mupdf.fz_vertex_c_get, _mupdf.fz_vertex_c_set)
1645: (4)                def __init__(self):
1646: (8)                    _mupdf.fz_vertex_swiginit(self, _mupdf.new_fz_vertex())
1647: (4)                __swig_destroy__ = _mupdf.delete_fz_vertex
1648: (0)            _mupdf.fz_vertex_swigregister(fz_vertex)
1649: (0)            class fz_text_decoder(object):
1650: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1651: (4)                __repr__ = _swig_repr
1652: (4)                decode_bound = property(_mupdf.fz_text_decoder_decode_bound_get,
_mupdf.fz_text_decoder_decode_bound_set)
1653: (4)                decode_size = property(_mupdf.fz_text_decoder_decode_size_get,
_mupdf.fz_text_decoder_decode_size_set)
1654: (4)                decode = property(_mupdf.fz_text_decoder_decode_get,
_mupdf.fz_text_decoder_decode_set)
1655: (4)                table1 = property(_mupdf.fz_text_decoder_table1_get,
_mupdf.fz_text_decoder_table1_set)
1656: (4)                table2 = property(_mupdf.fz_text_decoder_table2_get,
_mupdf.fz_text_decoder_table2_set)
1657: (4)                def __init__(self):
1658: (8)                    _mupdf.fz_text_decoder_swiginit(self, _mupdf.new_fz_text_decoder())
1659: (4)                __swig_destroy__ = _mupdf.delete_fz_text_decoder
1660: (0)            _mupdf.fz_text_decoder_swigregister(fz_text_decoder)
1661: (0)            fz_unicode_from_iso8859_1 = cvar.fz_unicode_from_iso8859_1
1662: (0)            fz_unicode_from_iso8859_7 = cvar.fz_unicode_from_iso8859_7
1663: (0)            fz_unicode_from_koi8u = cvar.fz_unicode_from_koi8u
```

```
1664: (0)           fz_unicode_from_pdf_doc_encoding = cvar.fz_unicode_from_pdf_doc_encoding
1665: (0)           fz_unicode_from_windows_1250 = cvar.fz_unicode_from_windows_1250
1666: (0)           fz_unicode_from_windows_1251 = cvar.fz_unicode_from_windows_1251
1667: (0)           fz_unicode_from_windows_1252 = cvar.fz_unicode_from_windows_1252
1668: (0)           FZ_ADOBE_CNS = _mupdf.FZ_ADOBE_CNS
1669: (0)           FZ_ADOBE_GB = _mupdf.FZ_ADOBE_GB
1670: (0)           FZ_ADOBE_JAPAN = _mupdf.FZ_ADOBE_JAPAN
1671: (0)           FZ_ADOBE_KOREA = _mupdf.FZ_ADOBE_KOREA
1672: (0)           class fz_font_flags_t(object):
1673: (4)               r"""
1674: (4)               Every fz_font carries a set of flags
1675: (4)               within it, in a fz_font_flags_t structure.
1676: (4)               """
1677: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1678: (4)               __repr__ = _swig_repr
1679: (4)               is_mono = property(_mupdf.fz_font_flags_t_is_mono_get,
_mupdf.fz_font_flags_t_is_mono_set)
1680: (4)               is_serif = property(_mupdf.fz_font_flags_t_is_serif_get,
_mupdf.fz_font_flags_t_is_serif_set)
1681: (4)               is_bold = property(_mupdf.fz_font_flags_t_is_bold_get,
_mupdf.fz_font_flags_t_is_bold_set)
1682: (4)               is_italic = property(_mupdf.fz_font_flags_t_is_italic_get,
_mupdf.fz_font_flags_t_is_italic_set)
1683: (4)               ft_substitute = property(_mupdf.fz_font_flags_t_ft_substitute_get,
_mupdf.fz_font_flags_t_ft_substitute_set)
1684: (4)               ft_stretch = property(_mupdf.fz_font_flags_t_ft_stretch_get,
_mupdf.fz_font_flags_t_ft_stretch_set)
1685: (4)               fake_bold = property(_mupdf.fz_font_flags_t_fake_bold_get,
_mupdf.fz_font_flags_t_fake_bold_set)
1686: (4)               fake_italic = property(_mupdf.fz_font_flags_t_fake_italic_get,
_mupdf.fz_font_flags_t_fake_italic_set)
1687: (4)               has_opentype = property(_mupdf.fz_font_flags_t_has_opentype_get,
_mupdf.fz_font_flags_t_has_opentype_set)
1688: (4)               invalid_bbox = property(_mupdf.fz_font_flags_t_invalid_bbox_get,
_mupdf.fz_font_flags_t_invalid_bbox_set)
1689: (4)               cjk = property(_mupdf.fz_font_flags_t_cjk_get,
_mupdf.fz_font_flags_t_cjk_set)
1690: (4)               cjk_lang = property(_mupdf.fz_font_flags_t_cjk_lang_get,
_mupdf.fz_font_flags_t_cjk_lang_set)
1691: (4)               embed = property(_mupdf.fz_font_flags_t_embed_get,
_mupdf.fz_font_flags_t_embed_set)
1692: (4)               never_embed = property(_mupdf.fz_font_flags_t_never_embed_get,
_mupdf.fz_font_flags_t_never_embed_set)
1693: (4)               def __init__(self):
1694: (8)                   _mupdf.fz_font_flags_t_swiginit(self, _mupdf.new_fz_font_flags_t())
1695: (4)               __swig_destroy__ = _mupdf.delete_fz_font_flags_t
1696: (0)           _mupdf.fz_font_flags_t_swigregister(fz_font_flags_t)
1697: (0)           class fz_shaper_data_t(object):
1698: (4)               r"""
1699: (4)               In order to shape a given font, we need to
1700: (4)               declare it to a shaper library (harfbuzz, by default, but others
1701: (4)               are possible). To avoid redeclaring it every time we need to
1702: (4)               shape, we hold a shaper handle and the destructor for it within
1703: (4)               the font itself. The handle is initialised by the caller when
1704: (4)               first required and the destructor is called when the fz_font is
1705: (4)               destroyed.
1706: (4)               """
1707: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1708: (4)               __repr__ = _swig_repr
1709: (4)               shaper_handle = property(_mupdf.fz_shaper_data_t_shaper_handle_get,
_mupdf.fz_shaper_data_t_shaper_handle_set)
1710: (4)               destroy = property(_mupdf.fz_shaper_data_t_destroy_get,
_mupdf.fz_shaper_data_t_destroy_set)
1711: (4)               def __init__(self):
1712: (8)                   _mupdf.fz_shaper_data_t_swiginit(self, _mupdf.new_fz_shaper_data_t())
1713: (4)               __swig_destroy__ = _mupdf.delete_fz_shaper_data_t
1714: (0)           _mupdf.fz_shaper_data_t_swigregister(fz_shaper_data_t)
```

```
1715: (0)           class fz_font(object):
1716: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1717: (4)               __repr__ = _swig_repr
1718: (4)               refs = property(_mupdf.fz_font_refs_get, _mupdf.fz_font_refs_set)
1719: (4)               name = property(_mupdf.fz_font_name_get, _mupdf.fz_font_name_set)
1720: (4)               buffer = property(_mupdf.fz_font_buffer_get, _mupdf.fz_font_buffer_set)
1721: (4)               flags = property(_mupdf.fz_font_flags_get, _mupdf.fz_font_flags_set)
1722: (4)               ft_face = property(_mupdf.fz_font_ft_face_get, _mupdf.fz_font_ft_face_set)
1723: (4)               shaper_data = property(_mupdf.fz_font_shaper_data_get,
_mupdf.fz_font_shaper_data_set)
1724: (4)               t3matrix = property(_mupdf.fz_font_t3matrix_get,
_mupdf.fz_font_t3matrix_set)
1725: (4)               t3resources = property(_mupdf.fz_font_t3resources_get,
_mupdf.fz_font_t3resources_set)
1726: (4)               t3procs = property(_mupdf.fz_font_t3procs_get, _mupdf.fz_font_t3procs_set)
1727: (4)               t3lists = property(_mupdf.fz_font_t3lists_get, _mupdf.fz_font_t3lists_set)
1728: (4)               t3widths = property(_mupdf.fz_font_t3widths_get,
_mupdf.fz_font_t3widths_set)
1729: (4)               t3flags = property(_mupdf.fz_font_t3flags_get, _mupdf.fz_font_t3flags_set)
1730: (4)               t3doc = property(_mupdf.fz_font_t3doc_get, _mupdf.fz_font_t3doc_set)
1731: (4)               t3run = property(_mupdf.fz_font_t3run_get, _mupdf.fz_font_t3run_set)
1732: (4)               t3freeres = property(_mupdf.fz_font_t3freeres_get,
_mupdf.fz_font_t3freeres_set)
1733: (4)               bbox = property(_mupdf.fz_font_bbox_get, _mupdf.fz_font_bbox_set)
1734: (4)               glyph_count = property(_mupdf.fz_font_glyph_count_get,
_mupdf.fz_font_glyph_count_set)
1735: (4)               bbox_table = property(_mupdf.fz_font_bbox_table_get,
_mupdf.fz_font_bbox_table_set)
1736: (4)               use_glyph_bbox = property(_mupdf.fz_font_use_glyph_bbox_get,
_mupdf.fz_font_use_glyph_bbox_set)
1737: (4)               width_count = property(_mupdf.fz_font_width_count_get,
_mupdf.fz_font_width_count_set)
1738: (4)               width_default = property(_mupdf.fz_font_width_default_get,
_mupdf.fz_font_width_default_set)
1739: (4)               width_table = property(_mupdf.fz_font_width_table_get,
_mupdf.fz_font_width_table_set)
1740: (4)               advance_cache = property(_mupdf.fz_font_advance_cache_get,
_mupdf.fz_font_advance_cache_set)
1741: (4)               encoding_cache = property(_mupdf.fz_font_encoding_cache_get,
_mupdf.fz_font_encoding_cache_set)
1742: (4)               has_digest = property(_mupdf.fz_font_has_digest_get,
_mupdf.fz_font_has_digest_set)
1743: (4)               digest = property(_mupdf.fz_font_digest_get, _mupdf.fz_font_digest_set)
1744: (4)               subfont = property(_mupdf.fz_font_subfont_get, _mupdf.fz_font_subfont_set)
1745: (4)               def __init__(self):
1746: (8)                   _mupdf.fz_font_swiginit(self, _mupdf.new_fz_font())
1747: (4)               __swig_destroy__ = _mupdf.delete_fz_font
1748: (0)           _mupdf.fz_font_swigregister(fz_font)
1749: (0)           FZ_LINECAP_BUTT = _mupdf.FZ_LINECAP_BUTT
1750: (0)           FZ_LINECAP_ROUND = _mupdf.FZ_LINECAP_ROUND
1751: (0)           FZ_LINECAP_SQUARE = _mupdf.FZ_LINECAP_SQUARE
1752: (0)           FZ_LINECAP_TRIANGLE = _mupdf.FZ_LINECAP_TRIANGLE
1753: (0)           FZ_LINEJOIN_MITER = _mupdf.FZ_LINEJOIN_MITER
1754: (0)           FZ_LINEJOIN_ROUND = _mupdf.FZ_LINEJOIN_ROUND
1755: (0)           FZ_LINEJOIN_BEVEL = _mupdf.FZ_LINEJOIN_BEVEL
1756: (0)           FZ_LINEJOIN_MITER_XPS = _mupdf.FZ_LINEJOIN_MITER_XPS
1757: (0)           class fz_stroke_state(object):
1758: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1759: (4)               __repr__ = _swig_repr
1760: (4)               refs = property(_mupdf.fz_stroke_state_refs_get,
_mupdf.fz_stroke_state_refs_set)
1761: (4)               start_cap = property(_mupdf.fz_stroke_state_start_cap_get,
_mupdf.fz_stroke_state_start_cap_set)
1762: (4)               dash_cap = property(_mupdf.fz_stroke_state_dash_cap_get,
_mupdf.fz_stroke_state_dash_cap_set)
1763: (4)               end_cap = property(_mupdf.fz_stroke_state_end_cap_get,
_mupdf.fz_stroke_state_end_cap_set)
```

```
1764: (4)               linejoin = property(_mupdf.fz_stroke_state_linejoin_get,
_mupdf.fz_stroke_state_linejoin_set)
1765: (4)               linewidth = property(_mupdf.fz_stroke_state_linewidth_get,
_mupdf.fz_stroke_state_linewidth_set)
1766: (4)               miterlimit = property(_mupdf.fz_stroke_state_miterlimit_get,
_mupdf.fz_stroke_state_miterlimit_set)
1767: (4)               dash_phase = property(_mupdf.fz_stroke_state_dash_phase_get,
_mupdf.fz_stroke_state_dash_phase_set)
1768: (4)               dash_len = property(_mupdf.fz_stroke_state_dash_len_get,
_mupdf.fz_stroke_state_dash_len_set)
1769: (4)               dash_list = property(_mupdf.fz_stroke_state_dash_list_get,
_mupdf.fz_stroke_state_dash_list_set)
1770: (4)               def __init__(self):
1771: (8)                   _mupdf.fz_stroke_state_swiginit(self, _mupdf.new_fz_stroke_state())
1772: (4)               __swig_destroy__ = _mupdf.delete_fz_stroke_state
1773: (0)           _mupdf.fz_stroke_state_swigregister(fz_stroke_state)
1774: (0)           class fz_path_walker(object):
1775: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1776: (4)               __repr__ = _swig_repr
1777: (4)               moveto = property(_mupdf.fz_path_walker_moveto_get,
_mupdf.fz_path_walker_moveto_set)
1778: (4)               lineto = property(_mupdf.fz_path_walker_lineto_get,
_mupdf.fz_path_walker_lineto_set)
1779: (4)               curveto = property(_mupdf.fz_path_walker_curveto_get,
_mupdf.fz_path_walker_curveto_set)
1780: (4)               closepath = property(_mupdf.fz_path_walker_closepath_get,
_mupdf.fz_path_walker_closepath_set)
1781: (4)               quadto = property(_mupdf.fz_path_walker_quadto_get,
_mupdf.fz_path_walker_quadto_set)
1782: (4)               curvetov = property(_mupdf.fz_path_walker_curvetov_get,
_mupdf.fz_path_walker_curvetov_set)
1783: (4)               curvetoy = property(_mupdf.fz_path_walker_curvetoy_get,
_mupdf.fz_path_walker_curvetoy_set)
1784: (4)               rectto = property(_mupdf.fz_path_walker_rectto_get,
_mupdf.fz_path_walker_rectto_set)
1785: (4)               def __init__(self):
1786: (8)                   _mupdf.fz_path_walker_swiginit(self, _mupdf.new_fz_path_walker())
1787: (4)               __swig_destroy__ = _mupdf.delete_fz_path_walker
1788: (0)           _mupdf.fz_path_walker_swigregister(fz_path_walker)
1789: (0)           class fz_text_item(object):
1790: (4)               r"""
1791: (4)               Text buffer.
1792: (4)               The trm field contains the a, b, c and d coefficients.
1793: (4)               The e and f coefficients come from the individual elements,
1794: (4)               together they form the transform matrix for the glyph.
1795: (4)               Glyphs are referenced by glyph ID.
1796: (4)               The Unicode text equivalent is kept in a separate array
1797: (4)               with indexes into the glyph array.
1798: (4)               """
1799: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1800: (4)               __repr__ = _swig_repr
1801: (4)               x = property(_mupdf.fz_text_item_x_get, _mupdf.fz_text_item_x_set)
1802: (4)               y = property(_mupdf.fz_text_item_y_get, _mupdf.fz_text_item_y_set)
1803: (4)               gid = property(_mupdf.fz_text_item_gid_get, _mupdf.fz_text_item_gid_set)
1804: (4)               ucs = property(_mupdf.fz_text_item_ucs_get, _mupdf.fz_text_item_ucs_set)
1805: (4)               cid = property(_mupdf.fz_text_item_cid_get, _mupdf.fz_text_item_cid_set)
1806: (4)               def __init__(self):
1807: (8)                   _mupdf.fz_text_item_swiginit(self, _mupdf.new_fz_text_item())
1808: (4)               __swig_destroy__ = _mupdf.delete_fz_text_item
1809: (0)           _mupdf.fz_text_item_swigregister(fz_text_item)
1810: (0)           fz_default_stroke_state = cvar.fz_default_stroke_state
1811: (0)           FZ_LANG_UNSET = _mupdf.FZ_LANG_UNSET
1812: (0)           FZ_LANG_ur = _mupdf.FZ_LANG_ur
1813: (0)           FZ_LANG_urd = _mupdf.FZ_LANG_urd
1814: (0)           FZ_LANG_ko = _mupdf.FZ_LANG_ko
1815: (0)           FZ_LANG_ja = _mupdf.FZ_LANG_ja
1816: (0)           FZ_LANG_zh = _mupdf.FZ_LANG_zh
```

```
1817: (0)           FZ_LANG_zh_Hans = _mupdf.FZ_LANG_zh_Hans
1818: (0)           FZ_LANG_zh_Hant = _mupdf.FZ_LANG_zh_Hant
1819: (0)           class fz_text_span(object):
1820: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1821: (4)               __repr__ = _swig_repr
1822: (4)               font = property(_mupdf.fz_text_span_font_get,
_mupdf.fz_text_span_font_set)
1823: (4)               trm = property(_mupdf.fz_text_span_trm_get, _mupdf.fz_text_span_trm_set)
1824: (4)               wmode = property(_mupdf.fz_text_span_wmode_get,
_mupdf.fz_text_span_wmode_set)
1825: (4)               bidi_level = property(_mupdf.fz_text_span_bidi_level_get,
_mupdf.fz_text_span_bidi_level_set)
1826: (4)               markup_dir = property(_mupdf.fz_text_span_markup_dir_get,
_mupdf.fz_text_span_markup_dir_set)
1827: (4)               language = property(_mupdf.fz_text_span_language_get,
_mupdf.fz_text_span_language_set)
1828: (4)               len = property(_mupdf.fz_text_span_len_get, _mupdf.fz_text_span_len_set)
1829: (4)               cap = property(_mupdf.fz_text_span_cap_get, _mupdf.fz_text_span_cap_set)
1830: (4)               items = property(_mupdf.fz_text_span_items_get,
_mupdf.fz_text_span_items_set)
1831: (4)               next = property(_mupdf.fz_text_span_next_get,
_mupdf.fz_text_span_next_set)
1832: (4)               def __init__(self):
1833: (8)                   _mupdf.fz_text_span_swiginit(self, _mupdf.new_fz_text_span())
1834: (4)               __swig_destroy__ = _mupdf.delete_fz_text_span
1835: (0)           _mupdf.fz_text_span_swigregister(fz_text_span)
1836: (0)           class fz_text(object):
1837: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1838: (4)               __repr__ = _swig_repr
1839: (4)               refs = property(_mupdf.fz_text_refs_get, _mupdf.fz_text_refs_set)
1840: (4)               head = property(_mupdf.fz_text_head_get, _mupdf.fz_text_head_set)
1841: (4)               tail = property(_mupdf.fz_text_tail_get, _mupdf.fz_text_tail_set)
1842: (4)               def __init__(self):
1843: (8)                   _mupdf.fz_text_swiginit(self, _mupdf.new_fz_text())
1844: (4)               __swig_destroy__ = _mupdf.delete_fz_text
1845: (0)           _mupdf.fz_text_swigregister(fz_text)
1846: (0)           FZ_DEVFLAG_MASK = _mupdf.FZ_DEVFLAG_MASK
1847: (0)           FZ_DEVFLAG_COLOR = _mupdf.FZ_DEVFLAG_COLOR
1848: (0)           FZ_DEVFLAG_UNCACHEABLE = _mupdf.FZ_DEVFLAG_UNCACHEABLE
1849: (0)           FZ_DEVFLAG_FILLCOLOR_UNDEFINED = _mupdf.FZ_DEVFLAG_FILLCOLOR_UNDEFINED
1850: (0)           FZ_DEVFLAG_STROKECOLOR_UNDEFINED = _mupdf.FZ_DEVFLAG_STROKECOLOR_UNDEFINED
1851: (0)           FZ_DEVFLAG_STARTCAP_UNDEFINED = _mupdf.FZ_DEVFLAG_STARTCAP_UNDEFINED
1852: (0)           FZ_DEVFLAG_DASHCAP_UNDEFINED = _mupdf.FZ_DEVFLAG_DASHCAP_UNDEFINED
1853: (0)           FZ_DEVFLAG_ENDCAP_UNDEFINED = _mupdf.FZ_DEVFLAG_ENDCAP_UNDEFINED
1854: (0)           FZ_DEVFLAG_LINEJOIN_UNDEFINED = _mupdf.FZ_DEVFLAG_LINEJOIN_UNDEFINED
1855: (0)           FZ_DEVFLAG_MITERLIMIT_UNDEFINED = _mupdf.FZ_DEVFLAG_MITERLIMIT_UNDEFINED
1856: (0)           FZ_DEVFLAG_LINEWIDTH_UNDEFINED = _mupdf.FZ_DEVFLAG_LINEWIDTH_UNDEFINED
1857: (0)           FZ_DEVFLAG_BBOX_DEFINED = _mupdf.FZ_DEVFLAG_BBOX_DEFINED
1858: (0)           FZ_DEVFLAG_GRIDFIT_AS_TILED = _mupdf.FZ_DEVFLAG_GRIDFIT_AS_TILED
1859: (0)           FZ_BLEND_NORMAL = _mupdf.FZ_BLEND_NORMAL
1860: (0)           FZ_BLEND_MULTIPLY = _mupdf.FZ_BLEND_MULTIPLY
1861: (0)           FZ_BLEND_SCREEN = _mupdf.FZ_BLEND_SCREEN
1862: (0)           FZ_BLEND_OVERLAY = _mupdf.FZ_BLEND_OVERLAY
1863: (0)           FZ_BLEND_DARKEN = _mupdf.FZ_BLEND_DARKEN
1864: (0)           FZ_BLEND_LIGHTEN = _mupdf.FZ_BLEND_LIGHTEN
1865: (0)           FZ_BLEND_COLOR_DODGE = _mupdf.FZ_BLEND_COLOR_DODGE
1866: (0)           FZ_BLEND_COLOR_BURN = _mupdf.FZ_BLEND_COLOR_BURN
1867: (0)           FZ_BLEND_HARD_LIGHT = _mupdf.FZ_BLEND_HARD_LIGHT
1868: (0)           FZ_BLEND_SOFT_LIGHT = _mupdf.FZ_BLEND_SOFT_LIGHT
1869: (0)           FZ_BLEND_DIFFERENCE = _mupdf.FZ_BLEND_DIFFERENCE
1870: (0)           FZ_BLEND_EXCLUSION = _mupdf.FZ_BLEND_EXCLUSION
1871: (0)           FZ_BLEND_HUE = _mupdf.FZ_BLEND_HUE
1872: (0)           FZ_BLEND_SATURATION = _mupdf.FZ_BLEND_SATURATION
1873: (0)           FZ_BLEND_COLOR = _mupdf.FZ_BLEND_COLOR
1874: (0)           FZ_BLEND_LUMINOSITY = _mupdf.FZ_BLEND_LUMINOSITY
1875: (0)           FZ_BLEND_MODEMASK = _mupdf.FZ_BLEND_MODEMASK
1876: (0)           FZ_BLEND_ISOLATED = _mupdf.FZ_BLEND_ISOLATED
```

```
1877: (0)           FZ_BLEND_KNOCKOUT = _mupdf.FZ_BLEND_KNOCKOUT
1878: (0)           FZ_FUNCTION_MAX_N = _mupdf.FZ_FUNCTION_MAX_N
1879: (0)           FZ_FUNCTION_MAX_M = _mupdf.FZ_FUNCTION_MAX_M
1880: (0)           class fz_function(object):
1881: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1882: (4)               __repr__ = _swig_repr
1883: (4)               storable = property(_mupdf.fz_function_storable_get,
_mupdf.fz_function_storable_set)
1884: (4)               size = property(_mupdf.fz_function_size_get, _mupdf.fz_function_size_set)
1885: (4)               m = property(_mupdf.fz_function_m_get, _mupdf.fz_function_m_set)
1886: (4)               n = property(_mupdf.fz_function_n_get, _mupdf.fz_function_n_set)
1887: (4)               eval = property(_mupdf.fz_function_eval_get, _mupdf.fz_function_eval_set)
1888: (4)               def __init__(self):
1889: (8)                   _mupdf.fz_function_swiginit(self, _mupdf.new_fz_function())
1890: (4)               __swig_destroy__ = _mupdf.delete_fz_function
1891: (0)           _mupdf.fz_function_swigregister(fz_function)
1892: (0)           class fz_device_container_stack(object):
1893: (4)               r"""
1894: (4)               The device structure is public to allow devices to be
1895: (4)               implemented outside of fitz.
1896: (4)               Device methods should always be called using e.g.
1897: (4)               fz_fill_path(ctx, dev, ...) rather than
1898: (4)               dev->fill_path(ctx, dev, ...)
1899: (4)               Devices can keep track of containers (clips/masks/groups/tiles)
1900: (4)               as they go to save callers having to do it.
1901: (4)               """
1902: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1903: (4)               __repr__ = _swig_repr
1904: (4)               scissor = property(_mupdf.fz_device_container_stack_scissor_get,
_mupdf.fz_device_container_stack_scissor_set)
1905: (4)               type = property(_mupdf.fz_device_container_stack_type_get,
_mupdf.fz_device_container_stack_type_set)
1906: (4)               user = property(_mupdf.fz_device_container_stack_user_get,
_mupdf.fz_device_container_stack_user_set)
1907: (4)               def __init__(self):
1908: (8)                   _mupdf.fz_device_container_stack_swiginit(self,
_mupdf.new_fz_device_container_stack())
1909: (4)               __swig_destroy__ = _mupdf.delete_fz_device_container_stack
1910: (0)           _mupdf.fz_device_container_stack_swigregister(fz_device_container_stack)
1911: (0)           fz_device_container_stack_is_clip = _mupdf.fz_device_container_stack_is_clip
1912: (0)           fz_device_container_stack_is_mask = _mupdf.fz_device_container_stack_is_mask
1913: (0)           fz_device_container_stack_is_group = _mupdf.fz_device_container_stack_is_group
1914: (0)           fz_device_container_stack_is_tile = _mupdf.fz_device_container_stack_is_tile
1915: (0)           FZ_STRUCTURE_INVALID = _mupdf.FZ_STRUCTURE_INVALID
1916: (0)           FZ_STRUCTURE_DOCUMENT = _mupdf.FZ_STRUCTURE_DOCUMENT
1917: (0)           FZ_STRUCTURE_PART = _mupdf.FZ_STRUCTURE_PART
1918: (0)           FZ_STRUCTURE_ART = _mupdf.FZ_STRUCTURE_ART
1919: (0)           FZ_STRUCTURE_SECT = _mupdf.FZ_STRUCTURE_SECT
1920: (0)           FZ_STRUCTURE_DIV = _mupdf.FZ_STRUCTURE_DIV
1921: (0)           FZ_STRUCTURE_BLOCKQUOTE = _mupdf.FZ_STRUCTURE_BLOCKQUOTE
1922: (0)           FZ_STRUCTURE_CAPTION = _mupdf.FZ_STRUCTURE_CAPTION
1923: (0)           FZ_STRUCTURE_TOC = _mupdf.FZ_STRUCTURE_TOC
1924: (0)           FZ_STRUCTURE_TOCI = _mupdf.FZ_STRUCTURE_TOCI
1925: (0)           FZ_STRUCTURE_INDEX = _mupdf.FZ_STRUCTURE_INDEX
1926: (0)           FZ_STRUCTURE_NONSTRUCT = _mupdf.FZ_STRUCTURE_NONSTRUCT
1927: (0)           FZ_STRUCTURE_PRIVATE = _mupdf.FZ_STRUCTURE_PRIVATE
1928: (0)           FZ_STRUCTURE_DOCUMENTFRAGMENT = _mupdf.FZ_STRUCTURE_DOCUMENTFRAGMENT
1929: (0)           FZ_STRUCTURE_ASIDE = _mupdf.FZ_STRUCTURE_ASIDE
1930: (0)           FZ_STRUCTURE_TITLE = _mupdf.FZ_STRUCTURE_TITLE
1931: (0)           FZ_STRUCTURE_FENOTE = _mupdf.FZ_STRUCTURE_FENOTE
1932: (0)           FZ_STRUCTURE_SUB = _mupdf.FZ_STRUCTURE_SUB
1933: (0)           FZ_STRUCTURE_P = _mupdf.FZ_STRUCTURE_P
1934: (0)           FZ_STRUCTURE_H = _mupdf.FZ_STRUCTURE_H
1935: (0)           FZ_STRUCTURE_H1 = _mupdf.FZ_STRUCTURE_H1
1936: (0)           FZ_STRUCTURE_H2 = _mupdf.FZ_STRUCTURE_H2
1937: (0)           FZ_STRUCTURE_H3 = _mupdf.FZ_STRUCTURE_H3
1938: (0)           FZ_STRUCTURE_H4 = _mupdf.FZ_STRUCTURE_H4
```

```
1939: (0)           FZ_STRUCTURE_H5 = _mupdf.FZ_STRUCTURE_H5
1940: (0)           FZ_STRUCTURE_H6 = _mupdf.FZ_STRUCTURE_H6
1941: (0)           FZ_STRUCTURE_LIST = _mupdf.FZ_STRUCTURE_LIST
1942: (0)           FZ_STRUCTURE_LISTITEM = _mupdf.FZ_STRUCTURE_LISTITEM
1943: (0)           FZ_STRUCTURE_LABEL = _mupdf.FZ_STRUCTURE_LABEL
1944: (0)           FZ_STRUCTURE_LISTBODY = _mupdf.FZ_STRUCTURE_LISTBODY
1945: (0)           FZ_STRUCTURE_TABLE = _mupdf.FZ_STRUCTURE_TABLE
1946: (0)           FZ_STRUCTURE_TR = _mupdf.FZ_STRUCTURE_TR
1947: (0)           FZ_STRUCTURE_TH = _mupdf.FZ_STRUCTURE_TH
1948: (0)           FZ_STRUCTURE_TD = _mupdf.FZ_STRUCTURE_TD
1949: (0)           FZ_STRUCTURE_THEAD = _mupdf.FZ_STRUCTURE_THEAD
1950: (0)           FZ_STRUCTURE_TBODY = _mupdf.FZ_STRUCTURE_TBODY
1951: (0)           FZ_STRUCTURE_TFOOT = _mupdf.FZ_STRUCTURE_TFOOT
1952: (0)           FZ_STRUCTURE_SPAN = _mupdf.FZ_STRUCTURE_SPAN
1953: (0)           FZ_STRUCTURE_QUOTE = _mupdf.FZ_STRUCTURE_QUOTE
1954: (0)           FZ_STRUCTURE_NOTE = _mupdf.FZ_STRUCTURE_NOTE
1955: (0)           FZ_STRUCTURE_REFERENCE = _mupdf.FZ_STRUCTURE_REFERENCE
1956: (0)           FZ_STRUCTURE_BIBENTRY = _mupdf.FZ_STRUCTURE_BIBENTRY
1957: (0)           FZ_STRUCTURE_CODE = _mupdf.FZ_STRUCTURE_CODE
1958: (0)           FZ_STRUCTURE_LINK = _mupdf.FZ_STRUCTURE_LINK
1959: (0)           FZ_STRUCTURE_ANNOT = _mupdf.FZ_STRUCTURE_ANNOT
1960: (0)           FZ_STRUCTURE_EM = _mupdf.FZ_STRUCTURE_EM
1961: (0)           FZ_STRUCTURE_STRONG = _mupdf.FZ_STRUCTURE_STRONG
1962: (0)           FZ_STRUCTURE_RUBY = _mupdf.FZ_STRUCTURE_RUBY
1963: (0)           FZ_STRUCTURE_RB = _mupdf.FZ_STRUCTURE_RB
1964: (0)           FZ_STRUCTURE_RT = _mupdf.FZ_STRUCTURE_RT
1965: (0)           FZ_STRUCTURE_RP = _mupdf.FZ_STRUCTURE_RP
1966: (0)           FZ_STRUCTURE_WARICHU = _mupdf.FZ_STRUCTURE_WARICHU
1967: (0)           FZ_STRUCTURE_WT = _mupdf.FZ_STRUCTURE_WT
1968: (0)           FZ_STRUCTURE_WP = _mupdf.FZ_STRUCTURE_WP
1969: (0)           FZ_STRUCTURE_FIGURE = _mupdf.FZ_STRUCTURE_FIGURE
1970: (0)           FZ_STRUCTURE_FORMULA = _mupdf.FZ_STRUCTURE_FORMULA
1971: (0)           FZ_STRUCTURE_FORM = _mupdf.FZ_STRUCTURE_FORM
1972: (0)           FZ_STRUCTURE_ARTIFACT = _mupdf.FZ_STRUCTURE_ARTIFACT
1973: (0)           FZ_METATEXT_ACTUALTEXT = _mupdf.FZ_METATEXT_ACTUALTEXT
1974: (0)           FZ_METATEXT_ALT = _mupdf.FZ_METATEXT_ALT
1975: (0)           FZ_METATEXT_ABBREVIATION = _mupdf.FZ_METATEXT_ABBREVIATION
1976: (0)           FZ_METATEXT_TITLE = _mupdf.FZ_METATEXT_TITLE
1977: (0)           class fz_device(object):
1978: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
1979: (4)               __repr__ = _swig_repr
1980: (4)               refs = property(_mupdf.fz_device_refs_get, _mupdf.fz_device_refs_set)
1981: (4)               hints = property(_mupdf.fz_device_hints_get, _mupdf.fz_device_hints_set)
1982: (4)               flags = property(_mupdf.fz_device_flags_get, _mupdf.fz_device_flags_set)
1983: (4)               close_device = property(_mupdf.fz_device_close_device_get,
_mupdf.fz_device_close_device_set)
1984: (4)               drop_device = property(_mupdf.fz_device_drop_device_get,
_mupdf.fz_device_drop_device_set)
1985: (4)               fill_path = property(_mupdf.fz_device_fill_path_get,
_mupdf.fz_device_fill_path_set)
1986: (4)               stroke_path = property(_mupdf.fz_device_stroke_path_get,
_mupdf.fz_device_stroke_path_set)
1987: (4)               clip_path = property(_mupdf.fz_device_clip_path_get,
_mupdf.fz_device_clip_path_set)
1988: (4)               clip_stroke_path = property(_mupdf.fz_device_clip_stroke_path_get,
_mupdf.fz_device_clip_stroke_path_set)
1989: (4)               fill_text = property(_mupdf.fz_device_fill_text_get,
_mupdf.fz_device_fill_text_set)
1990: (4)               stroke_text = property(_mupdf.fz_device_stroke_text_get,
_mupdf.fz_device_stroke_text_set)
1991: (4)               clip_text = property(_mupdf.fz_device_clip_text_get,
_mupdf.fz_device_clip_text_set)
1992: (4)               clip_stroke_text = property(_mupdf.fz_device_clip_stroke_text_get,
_mupdf.fz_device_clip_stroke_text_set)
1993: (4)               ignore_text = property(_mupdf.fz_device_ignore_text_get,
_mupdf.fz_device_ignore_text_set)
1994: (4)               fill_shade = property(_mupdf.fz_device_fill_shade_get,
_mupdf.fz_device_fill_shade_set)
```

```
1995: (4)                fill_image = property(_mupdf.fz_device_fill_image_get,
_mupdf.fz_device_fill_image_set)
1996: (4)                fill_image_mask = property(_mupdf.fz_device_fill_image_mask_get,
_mupdf.fz_device_fill_image_mask_set)
1997: (4)                clip_image_mask = property(_mupdf.fz_device_clip_image_mask_get,
_mupdf.fz_device_clip_image_mask_set)
1998: (4)                pop_clip = property(_mupdf.fz_device_pop_clip_get,
_mupdf.fz_device_pop_clip_set)
1999: (4)                begin_mask = property(_mupdf.fz_device_begin_mask_get,
_mupdf.fz_device_begin_mask_set)
2000: (4)                end_mask = property(_mupdf.fz_device_end_mask_get,
_mupdf.fz_device_end_mask_set)
2001: (4)                begin_group = property(_mupdf.fz_device_begin_group_get,
_mupdf.fz_device_begin_group_set)
2002: (4)                end_group = property(_mupdf.fz_device_end_group_get,
_mupdf.fz_device_end_group_set)
2003: (4)                begin_tile = property(_mupdf.fz_device_begin_tile_get,
_mupdf.fz_device_begin_tile_set)
2004: (4)                end_tile = property(_mupdf.fz_device_end_tile_get,
_mupdf.fz_device_end_tile_set)
2005: (4)                render_flags = property(_mupdf.fz_device_render_flags_get,
_mupdf.fz_device_render_flags_set)
2006: (4)                set_default_colorspaces =
property(_mupdf.fz_device_set_default_colorspaces_get,
_mupdf.fz_device_set_default_colorspaces_set)
2007: (4)                begin_layer = property(_mupdf.fz_device_begin_layer_get,
_mupdf.fz_device_begin_layer_set)
2008: (4)                end_layer = property(_mupdf.fz_device_end_layer_get,
_mupdf.fz_device_end_layer_set)
2009: (4)                begin_structure = property(_mupdf.fz_device_begin_structure_get,
_mupdf.fz_device_begin_structure_set)
2010: (4)                end_structure = property(_mupdf.fz_device_end_structure_get,
_mupdf.fz_device_end_structure_set)
2011: (4)                begin_metatext = property(_mupdf.fz_device_begin_metatext_get,
_mupdf.fz_device_begin_metatext_set)
2012: (4)                end_metatext = property(_mupdf.fz_device_end_metatext_get,
_mupdf.fz_device_end_metatext_set)
2013: (4)                d1_rect = property(_mupdf.fz_device_d1_rect_get,
_mupdf.fz_device_d1_rect_set)
2014: (4)                container_len = property(_mupdf.fz_device_container_len_get,
_mupdf.fz_device_container_len_set)
2015: (4)                container_cap = property(_mupdf.fz_device_container_cap_get,
_mupdf.fz_device_container_cap_set)
2016: (4)                container = property(_mupdf.fz_device_container_get,
_mupdf.fz_device_container_set)
2017: (4)                def __init__(self):
2018: (8)                    _mupdf.fz_device_swiginit(self, _mupdf.new_fz_device())
2019: (4)                __swig_destroy__ = _mupdf.delete_fz_device
2020: (0)            _mupdf.fz_device_swigregister(fz_device)
2021: (0)            FZ_DONT_INTERPOLATE_IMAGES = _mupdf.FZ_DONT_INTERPOLATE_IMAGES
2022: (0)            FZ_NO_CACHE = _mupdf.FZ_NO_CACHE
2023: (0)            FZ_DONT_DECODE_IMAGES = _mupdf.FZ_DONT_DECODE_IMAGES
2024: (0)            class fz_cookie(object):
2025: (4)                r"""
2026: (4)                Cookie support - simple communication channel between app/library.
2027: (4)                Provide two-way communication between application and library.
2028: (4)                Intended for multi-threaded applications where one thread is
2029: (4)                rendering pages and another thread wants to read progress
2030: (4)                feedback or abort a job that takes a long time to finish. The
2031: (4)                communication is unsynchronized without locking.
2032: (4)                abort: The application should set this field to 0 before
2033: (4)                calling fz_run_page to render a page. At any point when the
2034: (4)                page is being rendered the application my set this field to 1
2035: (4)                which will cause the rendering to finish soon. This field is
2036: (4)                checked periodically when the page is rendered, but exactly
2037: (4)                when is not known, therefore there is no upper bound on
2038: (4)                exactly when the rendering will abort. If the application
2039: (4)                did not provide a set of locks to fz_new_context, it must also
2040: (4)                await the completion of fz_run_page before issuing another
```

```
2041: (4)               call to fz_run_page. Note that once the application has set
2042: (4)               this field to 1 after it called fz_run_page it may not change
2043: (4)               the value again.
2044: (4)               progress: Communicates rendering progress back to the
2045: (4)               application and is read only. Increments as a page is being
2046: (4)               rendered. The value starts out at 0 and is limited to less
2047: (4)               than or equal to progress_max, unless progress_max is -1.
2048: (4)               progress_max: Communicates the known upper bound of rendering
2049: (4)               back to the application and is read only. The maximum value
2050: (4)               that the progress field may take. If there is no known upper
2051: (4)               bound on how long the rendering may take this value is -1 and
2052: (4)               progress is not limited. Note that the value of progress_max
2053: (4)               may change from -1 to a positive value once an upper bound is
2054: (4)               known, so take this into consideration when comparing the
2055: (4)               value of progress to that of progress_max.
2056: (4)               errors: count of errors during current rendering.
2057: (4)               incomplete: Initially should be set to 0. Will be set to
2058: (4)               non-zero if a TRYLATER error is thrown during rendering.
2059: (4)               """
2060: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2061: (4)               __repr__ = _swig_repr
2062: (4)               abort = property(_mupdf.fz_cookie_abort_get, _mupdf.fz_cookie_abort_set)
2063: (4)               progress = property(_mupdf.fz_cookie_progress_get,
_mupdf.fz_cookie_progress_set)
2064: (4)               progress_max = property(_mupdf.fz_cookie_progress_max_get,
_mupdf.fz_cookie_progress_max_set)
2065: (4)               errors = property(_mupdf.fz_cookie_errors_get,
_mupdf.fz_cookie_errors_set)
2066: (4)               incomplete = property(_mupdf.fz_cookie_incomplete_get,
_mupdf.fz_cookie_incomplete_set)
2067: (4)               def __init__(self):
2068: (8)                   _mupdf.fz_cookie_swiginit(self, _mupdf.new_fz_cookie())
2069: (4)               __swig_destroy__ = _mupdf.delete_fz_cookie
2070: (0)           _mupdf.fz_cookie_swigregister(fz_cookie)
2071: (0)           FZ_TEST_OPT_IMAGES = _mupdf.FZ_TEST_OPT_IMAGES
2072: (0)           FZ_TEST_OPT_SHADINGS = _mupdf.FZ_TEST_OPT_SHADINGS
2073: (0)           class fz_draw_options(object):
2074: (4)               r"""
2075: (4)               struct fz_draw_options: Options for creating a pixmap and draw
2076: (4)               device.
2077: (4)               """
2078: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2079: (4)               __repr__ = _swig_repr
2080: (4)               rotate = property(_mupdf.fz_draw_options_rotate_get,
_mupdf.fz_draw_options_rotate_set)
2081: (4)               x_resolution = property(_mupdf.fz_draw_options_x_resolution_get,
_mupdf.fz_draw_options_x_resolution_set)
2082: (4)               y_resolution = property(_mupdf.fz_draw_options_y_resolution_get,
_mupdf.fz_draw_options_y_resolution_set)
2083: (4)               width = property(_mupdf.fz_draw_options_width_get,
_mupdf.fz_draw_options_width_set)
2084: (4)               height = property(_mupdf.fz_draw_options_height_get,
_mupdf.fz_draw_options_height_set)
2085: (4)               colorspace = property(_mupdf.fz_draw_options_colorspace_get,
_mupdf.fz_draw_options_colorspace_set)
2086: (4)               alpha = property(_mupdf.fz_draw_options_alpha_get,
_mupdf.fz_draw_options_alpha_set)
2087: (4)               graphics = property(_mupdf.fz_draw_options_graphics_get,
_mupdf.fz_draw_options_graphics_set)
2088: (4)               text = property(_mupdf.fz_draw_options_text_get,
_mupdf.fz_draw_options_text_set)
2089: (4)               def __init__(self):
2090: (8)                   _mupdf.fz_draw_options_swiginit(self, _mupdf.new_fz_draw_options())
2091: (4)               __swig_destroy__ = _mupdf.delete_fz_draw_options
2092: (0)           _mupdf.fz_draw_options_swigregister(fz_draw_options)
2093: (0)           class fz_location(object):
2094: (4)               r"""
```

```
2095: (4)               Locations within the document are referred to in terms of
2096: (4)               chapter and page, rather than just a page number. For some
2097: (4)               documents (such as epub documents with large numbers of pages
2098: (4)               broken into many chapters) this can make navigation much faster
2099: (4)               as only the required chapter needs to be decoded at a time.
2100: (4)               """
2101: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2102: (4)               __repr__ = _swig_repr
2103: (4)               chapter = property(_mupdf.fz_location_chapter_get,
_mupdf.fz_location_chapter_set)
2104: (4)               page = property(_mupdf.fz_location_page_get, _mupdf.fz_location_page_set)
2105: (4)               def __init__(self):
2106: (8)                   _mupdf.fz_location_swiginit(self, _mupdf.new_fz_location())
2107: (4)               __swig_destroy__ = _mupdf.delete_fz_location
2108: (0)           _mupdf.fz_location_swigregister(fz_location)
2109: (0)           class fz_layout_char(object):
2110: (4)               r"""    Simple text layout (for use with annotation editing primarily)."""
2111: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2112: (4)               __repr__ = _swig_repr
2113: (4)               x = property(_mupdf.fz_layout_char_x_get, _mupdf.fz_layout_char_x_set)
2114: (4)               advance = property(_mupdf.fz_layout_char_advance_get,
_mupdf.fz_layout_char_advance_set)
2115: (4)               p = property(_mupdf.fz_layout_char_p_get, _mupdf.fz_layout_char_p_set)
2116: (4)               next = property(_mupdf.fz_layout_char_next_get,
_mupdf.fz_layout_char_next_set)
2117: (4)               def __init__(self):
2118: (8)                   _mupdf.fz_layout_char_swiginit(self, _mupdf.new_fz_layout_char())
2119: (4)               __swig_destroy__ = _mupdf.delete_fz_layout_char
2120: (0)           _mupdf.fz_layout_char_swigregister(fz_layout_char)
2121: (0)           class fz_layout_line(object):
2122: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2123: (4)               __repr__ = _swig_repr
2124: (4)               x = property(_mupdf.fz_layout_line_x_get, _mupdf.fz_layout_line_x_set)
2125: (4)               y = property(_mupdf.fz_layout_line_y_get, _mupdf.fz_layout_line_y_set)
2126: (4)               font_size = property(_mupdf.fz_layout_line_font_size_get,
_mupdf.fz_layout_line_font_size_set)
2127: (4)               p = property(_mupdf.fz_layout_line_p_get, _mupdf.fz_layout_line_p_set)
2128: (4)               text = property(_mupdf.fz_layout_line_text_get,
_mupdf.fz_layout_line_text_set)
2129: (4)               next = property(_mupdf.fz_layout_line_next_get,
_mupdf.fz_layout_line_next_set)
2130: (4)               def __init__(self):
2131: (8)                   _mupdf.fz_layout_line_swiginit(self, _mupdf.new_fz_layout_line())
2132: (4)               __swig_destroy__ = _mupdf.delete_fz_layout_line
2133: (0)           _mupdf.fz_layout_line_swigregister(fz_layout_line)
2134: (0)           class fz_layout_block(object):
2135: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2136: (4)               __repr__ = _swig_repr
2137: (4)               pool = property(_mupdf.fz_layout_block_pool_get,
_mupdf.fz_layout_block_pool_set)
2138: (4)               matrix = property(_mupdf.fz_layout_block_matrix_get,
_mupdf.fz_layout_block_matrix_set)
2139: (4)               inv_matrix = property(_mupdf.fz_layout_block_inv_matrix_get,
_mupdf.fz_layout_block_inv_matrix_set)
2140: (4)               head = property(_mupdf.fz_layout_block_head_get,
_mupdf.fz_layout_block_head_set)
2141: (4)               tailp = property(_mupdf.fz_layout_block_tailp_get,
_mupdf.fz_layout_block_tailp_set)
2142: (4)               text_tailp = property(_mupdf.fz_layout_block_text_tailp_get,
_mupdf.fz_layout_block_text_tailp_set)
2143: (4)               def __init__(self):
2144: (8)                   _mupdf.fz_layout_block_swiginit(self, _mupdf.new_fz_layout_block())
2145: (4)               __swig_destroy__ = _mupdf.delete_fz_layout_block
2146: (0)           _mupdf.fz_layout_block_swigregister(fz_layout_block)
2147: (0)           FZ_STEXT_PRESERVE_LIGATURES = _mupdf.FZ_STEXT_PRESERVE_LIGATURES
```

```
2148: (0)           FZ_STEXT_PRESERVE_WHITESPACE = _mupdf.FZ_STEXT_PRESERVE_WHITESPACE
2149: (0)           FZ_STEXT_PRESERVE_IMAGES = _mupdf.FZ_STEXT_PRESERVE_IMAGES
2150: (0)           FZ_STEXT_INHIBIT_SPACES = _mupdf.FZ_STEXT_INHIBIT_SPACES
2151: (0)           FZ_STEXT_DEHYPHENATE = _mupdf.FZ_STEXT_DEHYPHENATE
2152: (0)           FZ_STEXT_PRESERVE_SPANS = _mupdf.FZ_STEXT_PRESERVE_SPANS
2153: (0)           FZ_STEXT_MEDIABOX_CLIP = _mupdf.FZ_STEXT_MEDIABOX_CLIP
2154: (0)           FZ_STEXT_USE_CID_FOR_UNKNOWN_UNICODE =
_mupdf.FZ_STEXT_USE_CID_FOR_UNKNOWN_UNICODE
2155: (0)           class fz_stext_page(object):
2156: (4)               r"""
2157: (4)               A text page is a list of blocks, together with an overall
2158: (4)               bounding box.
2159: (4)               """
2160: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2161: (4)               __repr__ = _swig_repr
2162: (4)               pool = property(_mupdf.fz_stext_page_pool_get,
_mupdf.fz_stext_page_pool_set)
2163: (4)               mediabox = property(_mupdf.fz_stext_page_mediabox_get,
_mupdf.fz_stext_page_mediabox_set)
2164: (4)               first_block = property(_mupdf.fz_stext_page_first_block_get,
_mupdf.fz_stext_page_first_block_set)
2165: (4)               last_block = property(_mupdf.fz_stext_page_last_block_get,
_mupdf.fz_stext_page_last_block_set)
2166: (4)               def __init__(self):
2167: (8)                   _mupdf.fz_stext_page_swiginit(self, _mupdf.new_fz_stext_page())
2168: (4)               __swig_destroy__ = _mupdf.delete_fz_stext_page
2169: (0)           _mupdf.fz_stext_page_swigregister(fz_stext_page)
2170: (0)           FZ_STEXT_BLOCK_TEXT = _mupdf.FZ_STEXT_BLOCK_TEXT
2171: (0)           FZ_STEXT_BLOCK_IMAGE = _mupdf.FZ_STEXT_BLOCK_IMAGE
2172: (0)           class fz_stext_block(object):
2173: (4)               r"""
2174: (4)               A text block is a list of lines of text (typically a paragraph),
2175: (4)               or an image.
2176: (4)               """
2177: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2178: (4)               __repr__ = _swig_repr
2179: (4)               type = property(_mupdf.fz_stext_block_type_get,
_mupdf.fz_stext_block_type_set)
2180: (4)               bbox = property(_mupdf.fz_stext_block_bbox_get,
_mupdf.fz_stext_block_bbox_set)
2181: (4)               prev = property(_mupdf.fz_stext_block_prev_get,
_mupdf.fz_stext_block_prev_set)
2182: (4)               next = property(_mupdf.fz_stext_block_next_get,
_mupdf.fz_stext_block_next_set)
2183: (4)               def __init__(self):
2184: (8)                   _mupdf.fz_stext_block_swiginit(self, _mupdf.new_fz_stext_block())
2185: (4)               __swig_destroy__ = _mupdf.delete_fz_stext_block
2186: (0)           _mupdf.fz_stext_block_swigregister(fz_stext_block)
2187: (0)           class fz_stext_line(object):
2188: (4)               r"""    A text line is a list of characters that share a common
baseline."""
2189: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2190: (4)               __repr__ = _swig_repr
2191: (4)               wmode = property(_mupdf.fz_stext_line_wmode_get,
_mupdf.fz_stext_line_wmode_set)
2192: (4)               dir = property(_mupdf.fz_stext_line_dir_get, _mupdf.fz_stext_line_dir_set)
2193: (4)               bbox = property(_mupdf.fz_stext_line_bbox_get,
_mupdf.fz_stext_line_bbox_set)
2194: (4)               first_char = property(_mupdf.fz_stext_line_first_char_get,
_mupdf.fz_stext_line_first_char_set)
2195: (4)               last_char = property(_mupdf.fz_stext_line_last_char_get,
_mupdf.fz_stext_line_last_char_set)
2196: (4)               prev = property(_mupdf.fz_stext_line_prev_get,
_mupdf.fz_stext_line_prev_set)
2197: (4)               next = property(_mupdf.fz_stext_line_next_get,
_mupdf.fz_stext_line_next_set)
```

```
2198: (4)               def __init__(self):
2199: (8)                   _mupdf.fz_stext_line_swiginit(self, _mupdf.new_fz_stext_line())
2200: (4)               __swig_destroy__ = _mupdf.delete_fz_stext_line
2201: (0)           _mupdf.fz_stext_line_swigregister(fz_stext_line)
2202: (0)           class fz_stext_char(object):
2203: (4)               r"""
2204: (4)               A text char is a unicode character, the style in which is
2205: (4)               appears, and the point at which it is positioned.
2206: (4)               """
2207: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2208: (4)               __repr__ = _swig_repr
2209: (4)               c = property(_mupdf.fz_stext_char_c_get, _mupdf.fz_stext_char_c_set)
2210: (4)               bidi = property(_mupdf.fz_stext_char_bidi_get,
_mupdf.fz_stext_char_bidi_set)
2211: (4)               color = property(_mupdf.fz_stext_char_color_get,
_mupdf.fz_stext_char_color_set)
2212: (4)               origin = property(_mupdf.fz_stext_char_origin_get,
_mupdf.fz_stext_char_origin_set)
2213: (4)               quad = property(_mupdf.fz_stext_char_quad_get,
_mupdf.fz_stext_char_quad_set)
2214: (4)               size = property(_mupdf.fz_stext_char_size_get,
_mupdf.fz_stext_char_size_set)
2215: (4)               font = property(_mupdf.fz_stext_char_font_get,
_mupdf.fz_stext_char_font_set)
2216: (4)               next = property(_mupdf.fz_stext_char_next_get,
_mupdf.fz_stext_char_next_set)
2217: (4)               def __init__(self):
2218: (8)                   _mupdf.fz_stext_char_swiginit(self, _mupdf.new_fz_stext_char())
2219: (4)               __swig_destroy__ = _mupdf.delete_fz_stext_char
2220: (0)           _mupdf.fz_stext_char_swigregister(fz_stext_char)
2221: (0)           FZ_SELECT_CHARS = _mupdf.FZ_SELECT_CHARS
2222: (0)           FZ_SELECT_WORDS = _mupdf.FZ_SELECT_WORDS
2223: (0)           FZ_SELECT_LINES = _mupdf.FZ_SELECT_LINES
2224: (0)           class fz_stext_options(object):
2225: (4)               r"""    Options for creating structured text."""
2226: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2227: (4)               __repr__ = _swig_repr
2228: (4)               flags = property(_mupdf.fz_stext_options_flags_get,
_mupdf.fz_stext_options_flags_set)
2229: (4)               scale = property(_mupdf.fz_stext_options_scale_get,
_mupdf.fz_stext_options_scale_set)
2230: (4)               def __init__(self):
2231: (8)                   _mupdf.fz_stext_options_swiginit(self, _mupdf.new_fz_stext_options())
2232: (4)               __swig_destroy__ = _mupdf.delete_fz_stext_options
2233: (0)           _mupdf.fz_stext_options_swigregister(fz_stext_options)
2234: (0)           FZ_TRANSITION_NONE = _mupdf.FZ_TRANSITION_NONE
2235: (0)           FZ_TRANSITION_SPLIT = _mupdf.FZ_TRANSITION_SPLIT
2236: (0)           FZ_TRANSITION_BLINDS = _mupdf.FZ_TRANSITION_BLINDS
2237: (0)           FZ_TRANSITION_BOX = _mupdf.FZ_TRANSITION_BOX
2238: (0)           FZ_TRANSITION_WIPE = _mupdf.FZ_TRANSITION_WIPE
2239: (0)           FZ_TRANSITION_DISSOLVE = _mupdf.FZ_TRANSITION_DISSOLVE
2240: (0)           FZ_TRANSITION_GLITTER = _mupdf.FZ_TRANSITION_GLITTER
2241: (0)           FZ_TRANSITION_FLY = _mupdf.FZ_TRANSITION_FLY
2242: (0)           FZ_TRANSITION_PUSH = _mupdf.FZ_TRANSITION_PUSH
2243: (0)           FZ_TRANSITION_COVER = _mupdf.FZ_TRANSITION_COVER
2244: (0)           FZ_TRANSITION_UNCOVER = _mupdf.FZ_TRANSITION_UNCOVER
2245: (0)           FZ_TRANSITION_FADE = _mupdf.FZ_TRANSITION_FADE
2246: (0)           class fz_transition(object):
2247: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2248: (4)               __repr__ = _swig_repr
2249: (4)               type = property(_mupdf.fz_transition_type_get,
_mupdf.fz_transition_type_set)
2250: (4)               duration = property(_mupdf.fz_transition_duration_get,
_mupdf.fz_transition_duration_set)
2251: (4)               vertical = property(_mupdf.fz_transition_vertical_get,
_mupdf.fz_transition_vertical_set)
```

```
2252: (4)               outwards = property(_mupdf.fz_transition_outwards_get,
_mupdf.fz_transition_outwards_set)
2253: (4)               direction = property(_mupdf.fz_transition_direction_get,
_mupdf.fz_transition_direction_set)
2254: (4)               state0 = property(_mupdf.fz_transition_state0_get,
_mupdf.fz_transition_state0_set)
2255: (4)               state1 = property(_mupdf.fz_transition_state1_get,
_mupdf.fz_transition_state1_set)
2256: (4)               def __init__(self):
2257: (8)                   _mupdf.fz_transition_swiginit(self, _mupdf.new_fz_transition())
2258: (4)               __swig_destroy__ = _mupdf.delete_fz_transition
2259: (0)           _mupdf.fz_transition_swigregister(fz_transition)
2260: (0)           class fz_link(object):
2261: (4)               r"""
2262: (4)               fz_link is a list of interactive links on a page.
2263: (4)               There is no relation between the order of the links in the
2264: (4)               list and the order they appear on the page. The list of links
2265: (4)               for a given page can be obtained from fz_load_links.
2266: (4)               A link is reference counted. Dropping a reference to a link is
2267: (4)               done by calling fz_drop_link.
2268: (4)               rect: The hot zone. The area that can be clicked in
2269: (4)               untransformed coordinates.
2270: (4)               uri: Link destinations come in two forms: internal and external.
2271: (4)               Internal links refer to other pages in the same document.
2272: (4)               External links are URLs to other documents.
2273: (4)               next: A pointer to the next link on the same page.
2274: (4)               """
2275: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2276: (4)               __repr__ = _swig_repr
2277: (4)               refs = property(_mupdf.fz_link_refs_get, _mupdf.fz_link_refs_set)
2278: (4)               next = property(_mupdf.fz_link_next_get, _mupdf.fz_link_next_set)
2279: (4)               rect = property(_mupdf.fz_link_rect_get, _mupdf.fz_link_rect_set)
2280: (4)               uri = property(_mupdf.fz_link_uri_get, _mupdf.fz_link_uri_set)
2281: (4)               set_rect_fn = property(_mupdf.fz_link_set_rect_fn_get,
_mupdf.fz_link_set_rect_fn_set)
2282: (4)               set_uri_fn = property(_mupdf.fz_link_set_uri_fn_get,
_mupdf.fz_link_set_uri_fn_set)
2283: (4)               drop = property(_mupdf.fz_link_drop_get, _mupdf.fz_link_drop_set)
2284: (4)               def __init__(self):
2285: (8)                   _mupdf.fz_link_swiginit(self, _mupdf.new_fz_link())
2286: (4)               __swig_destroy__ = _mupdf.delete_fz_link
2287: (0)           _mupdf.fz_link_swigregister(fz_link)
2288: (0)           FZ_LINK_DEST_FIT = _mupdf.FZ_LINK_DEST_FIT
2289: (0)           FZ_LINK_DEST_FIT_B = _mupdf.FZ_LINK_DEST_FIT_B
2290: (0)           FZ_LINK_DEST_FIT_H = _mupdf.FZ_LINK_DEST_FIT_H
2291: (0)           FZ_LINK_DEST_FIT_BH = _mupdf.FZ_LINK_DEST_FIT_BH
2292: (0)           FZ_LINK_DEST_FIT_V = _mupdf.FZ_LINK_DEST_FIT_V
2293: (0)           FZ_LINK_DEST_FIT_BV = _mupdf.FZ_LINK_DEST_FIT_BV
2294: (0)           FZ_LINK_DEST_FIT_R = _mupdf.FZ_LINK_DEST_FIT_R
2295: (0)           FZ_LINK_DEST_XYZ = _mupdf.FZ_LINK_DEST_XYZ
2296: (0)           class fz_link_dest(object):
2297: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2298: (4)               __repr__ = _swig_repr
2299: (4)               loc = property(_mupdf.fz_link_dest_loc_get, _mupdf.fz_link_dest_loc_set)
2300: (4)               type = property(_mupdf.fz_link_dest_type_get,
_mupdf.fz_link_dest_type_set)
2301: (4)               x = property(_mupdf.fz_link_dest_x_get, _mupdf.fz_link_dest_x_set)
2302: (4)               y = property(_mupdf.fz_link_dest_y_get, _mupdf.fz_link_dest_y_set)
2303: (4)               w = property(_mupdf.fz_link_dest_w_get, _mupdf.fz_link_dest_w_set)
2304: (4)               h = property(_mupdf.fz_link_dest_h_get, _mupdf.fz_link_dest_h_set)
2305: (4)               zoom = property(_mupdf.fz_link_dest_zoom_get,
_mupdf.fz_link_dest_zoom_set)
2306: (4)               def __init__(self):
2307: (8)                   _mupdf.fz_link_dest_swiginit(self, _mupdf.new_fz_link_dest())
2308: (4)               __swig_destroy__ = _mupdf.delete_fz_link_dest
2309: (0)           _mupdf.fz_link_dest_swigregister(fz_link_dest)
2310: (0)           class fz_outline_item(object):
```

```
2311: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2312: (4)               __repr__ = _swig_repr
2313: (4)               title = property(_mupdf.fz_outline_item_title_get,
_mupdf.fz_outline_item_title_set)
2314: (4)               uri = property(_mupdf.fz_outline_item_uri_get,
_mupdf.fz_outline_item_uri_set)
2315: (4)               is_open = property(_mupdf.fz_outline_item_is_open_get,
_mupdf.fz_outline_item_is_open_set)
2316: (4)               def __init__(self):
2317: (8)                   _mupdf.fz_outline_item_swiginit(self, _mupdf.new_fz_outline_item())
2318: (4)               __swig_destroy__ = _mupdf.delete_fz_outline_item
2319: (0)           _mupdf.fz_outline_item_swigregister(fz_outline_item)
2320: (0)           class fz_outline(object):
2321: (4)               r"""
2322: (4)               Structure based API
2323: (4)               fz_outline is a tree of the outline of a document (also known
2324: (4)               as table of contents).
2325: (4)               title: Title of outline item using UTF-8 encoding. May be NULL
2326: (4)               if the outline item has no text string.
2327: (4)               uri: Destination in the document to be displayed when this
2328: (4)               outline item is activated. May be an internal or external
2329: (4)               link, or NULL if the outline item does not have a destination.
2330: (4)               page: The page number of an internal link, or -1 for external
2331: (4)               links or links with no destination.
2332: (4)               next: The next outline item at the same level as this outline
2333: (4)               item. May be NULL if no more outline items exist at this level.
2334: (4)               down: The outline items immediate children in the hierarchy.
2335: (4)               May be NULL if no children exist.
2336: (4)               """
2337: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2338: (4)               __repr__ = _swig_repr
2339: (4)               refs = property(_mupdf.fz_outline_refs_get, _mupdf.fz_outline_refs_set)
2340: (4)               title = property(_mupdf.fz_outline_title_get, _mupdf.fz_outline_title_set)
2341: (4)               uri = property(_mupdf.fz_outline_uri_get, _mupdf.fz_outline_uri_set)
2342: (4)               page = property(_mupdf.fz_outline_page_get, _mupdf.fz_outline_page_set)
2343: (4)               x = property(_mupdf.fz_outline_x_get, _mupdf.fz_outline_x_set)
2344: (4)               y = property(_mupdf.fz_outline_y_get, _mupdf.fz_outline_y_set)
2345: (4)               next = property(_mupdf.fz_outline_next_get, _mupdf.fz_outline_next_set)
2346: (4)               down = property(_mupdf.fz_outline_down_get, _mupdf.fz_outline_down_set)
2347: (4)               is_open = property(_mupdf.fz_outline_is_open_get,
_mupdf.fz_outline_is_open_set)
2348: (4)               def __init__(self):
2349: (8)                   _mupdf.fz_outline_swiginit(self, _mupdf.new_fz_outline())
2350: (4)               __swig_destroy__ = _mupdf.delete_fz_outline
2351: (0)           _mupdf.fz_outline_swigregister(fz_outline)
2352: (0)           class fz_outline_iterator(object):
2353: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2354: (4)               __repr__ = _swig_repr
2355: (4)               drop = property(_mupdf.fz_outline_iterator_drop_get,
_mupdf.fz_outline_iterator_drop_set)
2356: (4)               item = property(_mupdf.fz_outline_iterator_item_get,
_mupdf.fz_outline_iterator_item_set)
2357: (4)               next = property(_mupdf.fz_outline_iterator_next_get,
_mupdf.fz_outline_iterator_next_set)
2358: (4)               prev = property(_mupdf.fz_outline_iterator_prev_get,
_mupdf.fz_outline_iterator_prev_set)
2359: (4)               up = property(_mupdf.fz_outline_iterator_up_get,
_mupdf.fz_outline_iterator_up_set)
2360: (4)               down = property(_mupdf.fz_outline_iterator_down_get,
_mupdf.fz_outline_iterator_down_set)
2361: (4)               insert = property(_mupdf.fz_outline_iterator_insert_get,
_mupdf.fz_outline_iterator_insert_set)
2362: (4)               update = property(_mupdf.fz_outline_iterator_update_get,
_mupdf.fz_outline_iterator_update_set)
2363: (4)               _del = property(_mupdf.fz_outline_iterator__del_get,
_mupdf.fz_outline_iterator__del_set)
```

```
2364: (4)               doc = property(_mupdf.fz_outline_iterator_doc_get,
_mupdf.fz_outline_iterator_doc_set)
2365: (4)               def __init__(self):
2366: (8)                   _mupdf.fz_outline_iterator_swiginit(self,
_mupdf.new_fz_outline_iterator())
2367: (4)               __swig_destroy__ = _mupdf.delete_fz_outline_iterator
2368: (0)           _mupdf.fz_outline_iterator_swigregister(fz_outline_iterator)
2369: (0)           FZ_MEDIA_BOX = _mupdf.FZ_MEDIA_BOX
2370: (0)           FZ_CROP_BOX = _mupdf.FZ_CROP_BOX
2371: (0)           FZ_BLEED_BOX = _mupdf.FZ_BLEED_BOX
2372: (0)           FZ_TRIM_BOX = _mupdf.FZ_TRIM_BOX
2373: (0)           FZ_ART_BOX = _mupdf.FZ_ART_BOX
2374: (0)           FZ_UNKNOWN_BOX = _mupdf.FZ_UNKNOWN_BOX
2375: (0)           FZ_LAYOUT_KINDLE_W = _mupdf.FZ_LAYOUT_KINDLE_W
2376: (0)           FZ_LAYOUT_KINDLE_H = _mupdf.FZ_LAYOUT_KINDLE_H
2377: (0)           FZ_LAYOUT_KINDLE_EM = _mupdf.FZ_LAYOUT_KINDLE_EM
2378: (0)           FZ_LAYOUT_US_POCKET_W = _mupdf.FZ_LAYOUT_US_POCKET_W
2379: (0)           FZ_LAYOUT_US_POCKET_H = _mupdf.FZ_LAYOUT_US_POCKET_H
2380: (0)           FZ_LAYOUT_US_POCKET_EM = _mupdf.FZ_LAYOUT_US_POCKET_EM
2381: (0)           FZ_LAYOUT_US_TRADE_W = _mupdf.FZ_LAYOUT_US_TRADE_W
2382: (0)           FZ_LAYOUT_US_TRADE_H = _mupdf.FZ_LAYOUT_US_TRADE_H
2383: (0)           FZ_LAYOUT_US_TRADE_EM = _mupdf.FZ_LAYOUT_US_TRADE_EM
2384: (0)           FZ_LAYOUT_UK_A_FORMAT_W = _mupdf.FZ_LAYOUT_UK_A_FORMAT_W
2385: (0)           FZ_LAYOUT_UK_A_FORMAT_H = _mupdf.FZ_LAYOUT_UK_A_FORMAT_H
2386: (0)           FZ_LAYOUT_UK_A_FORMAT_EM = _mupdf.FZ_LAYOUT_UK_A_FORMAT_EM
2387: (0)           FZ_LAYOUT_UK_B_FORMAT_W = _mupdf.FZ_LAYOUT_UK_B_FORMAT_W
2388: (0)           FZ_LAYOUT_UK_B_FORMAT_H = _mupdf.FZ_LAYOUT_UK_B_FORMAT_H
2389: (0)           FZ_LAYOUT_UK_B_FORMAT_EM = _mupdf.FZ_LAYOUT_UK_B_FORMAT_EM
2390: (0)           FZ_LAYOUT_UK_C_FORMAT_W = _mupdf.FZ_LAYOUT_UK_C_FORMAT_W
2391: (0)           FZ_LAYOUT_UK_C_FORMAT_H = _mupdf.FZ_LAYOUT_UK_C_FORMAT_H
2392: (0)           FZ_LAYOUT_UK_C_FORMAT_EM = _mupdf.FZ_LAYOUT_UK_C_FORMAT_EM
2393: (0)           FZ_LAYOUT_A5_W = _mupdf.FZ_LAYOUT_A5_W
2394: (0)           FZ_LAYOUT_A5_H = _mupdf.FZ_LAYOUT_A5_H
2395: (0)           FZ_LAYOUT_A5_EM = _mupdf.FZ_LAYOUT_A5_EM
2396: (0)           FZ_DEFAULT_LAYOUT_W = _mupdf.FZ_DEFAULT_LAYOUT_W
2397: (0)           FZ_DEFAULT_LAYOUT_H = _mupdf.FZ_DEFAULT_LAYOUT_H
2398: (0)           FZ_DEFAULT_LAYOUT_EM = _mupdf.FZ_DEFAULT_LAYOUT_EM
2399: (0)           FZ_PERMISSION_PRINT = _mupdf.FZ_PERMISSION_PRINT
2400: (0)           FZ_PERMISSION_COPY = _mupdf.FZ_PERMISSION_COPY
2401: (0)           FZ_PERMISSION_EDIT = _mupdf.FZ_PERMISSION_EDIT
2402: (0)           FZ_PERMISSION_ANNOTATE = _mupdf.FZ_PERMISSION_ANNOTATE
2403: (0)           FZ_PERMISSION_FORM = _mupdf.FZ_PERMISSION_FORM
2404: (0)           FZ_PERMISSION_ACCESSIBILITY = _mupdf.FZ_PERMISSION_ACCESSIBILITY
2405: (0)           FZ_PERMISSION_ASSEMBLE = _mupdf.FZ_PERMISSION_ASSEMBLE
2406: (0)           FZ_PERMISSION_PRINT_HQ = _mupdf.FZ_PERMISSION_PRINT_HQ
2407: (0)           FZ_META_FORMAT = _mupdf.FZ_META_FORMAT
2408: (0)           FZ_META_ENCRYPTION = _mupdf.FZ_META_ENCRYPTION
2409: (0)           FZ_META_INFO = _mupdf.FZ_META_INFO
2410: (0)           FZ_META_INFO_TITLE = _mupdf.FZ_META_INFO_TITLE
2411: (0)           FZ_META_INFO_AUTHOR = _mupdf.FZ_META_INFO_AUTHOR
2412: (0)           FZ_META_INFO_SUBJECT = _mupdf.FZ_META_INFO_SUBJECT
2413: (0)           FZ_META_INFO_KEYWORDS = _mupdf.FZ_META_INFO_KEYWORDS
2414: (0)           FZ_META_INFO_CREATOR = _mupdf.FZ_META_INFO_CREATOR
2415: (0)           FZ_META_INFO_PRODUCER = _mupdf.FZ_META_INFO_PRODUCER
2416: (0)           FZ_META_INFO_CREATIONDATE = _mupdf.FZ_META_INFO_CREATIONDATE
2417: (0)           FZ_META_INFO_MODIFICATIONDATE = _mupdf.FZ_META_INFO_MODIFICATIONDATE
2418: (0)           class fz_page(object):
2419: (4)               r"""
2420: (4)               Structure definition is public so other classes can
2421: (4)               derive from it. Do not access the members directly.
2422: (4)               """
2423: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2424: (4)               __repr__ = _swig_repr
2425: (4)               refs = property(_mupdf.fz_page_refs_get, _mupdf.fz_page_refs_set)
2426: (4)               doc = property(_mupdf.fz_page_doc_get, _mupdf.fz_page_doc_set)
2427: (4)               chapter = property(_mupdf.fz_page_chapter_get, _mupdf.fz_page_chapter_set)
2428: (4)               number = property(_mupdf.fz_page_number_get, _mupdf.fz_page_number_set)
2429: (4)               incomplete = property(_mupdf.fz_page_incomplete_get,
```

```
_mupdf.fz_page_incomplete_set)
2430: (4)               drop_page = property(_mupdf.fz_page_drop_page_get,
_mupdf.fz_page_drop_page_set)
2431: (4)               bound_page = property(_mupdf.fz_page_bound_page_get,
_mupdf.fz_page_bound_page_set)
2432: (4)               run_page_contents = property(_mupdf.fz_page_run_page_contents_get,
_mupdf.fz_page_run_page_contents_set)
2433: (4)               run_page_annots = property(_mupdf.fz_page_run_page_annots_get,
_mupdf.fz_page_run_page_annots_set)
2434: (4)               run_page_widgets = property(_mupdf.fz_page_run_page_widgets_get,
_mupdf.fz_page_run_page_widgets_set)
2435: (4)               load_links = property(_mupdf.fz_page_load_links_get,
_mupdf.fz_page_load_links_set)
2436: (4)               page_presentation = property(_mupdf.fz_page_page_presentation_get,
_mupdf.fz_page_page_presentation_set)
2437: (4)               control_separation = property(_mupdf.fz_page_control_separation_get,
_mupdf.fz_page_control_separation_set)
2438: (4)               separation_disabled = property(_mupdf.fz_page_separation_disabled_get,
_mupdf.fz_page_separation_disabled_set)
2439: (4)               separations = property(_mupdf.fz_page_separations_get,
_mupdf.fz_page_separations_set)
2440: (4)               overprint = property(_mupdf.fz_page_overprint_get,
_mupdf.fz_page_overprint_set)
2441: (4)               create_link = property(_mupdf.fz_page_create_link_get,
_mupdf.fz_page_create_link_set)
2442: (4)               delete_link = property(_mupdf.fz_page_delete_link_get,
_mupdf.fz_page_delete_link_set)
2443: (4)               prev = property(_mupdf.fz_page_prev_get, _mupdf.fz_page_prev_set)
2444: (4)               next = property(_mupdf.fz_page_next_get, _mupdf.fz_page_next_set)
2445: (4)               def __init__(self):
2446: (8)                   _mupdf.fz_page_swiginit(self, _mupdf.new_fz_page())
2447: (4)               __swig_destroy__ = _mupdf.delete_fz_page
2448: (0)           _mupdf.fz_page_swigregister(fz_page)
2449: (0)           class fz_document(object):
2450: (4)               r"""
2451: (4)               Structure definition is public so other classes can
2452: (4)               derive from it. Callers should not access the members
2453: (4)               directly, though implementations will need initialize
2454: (4)               functions directly.
2455: (4)               """
2456: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2457: (4)               __repr__ = _swig_repr
2458: (4)               refs = property(_mupdf.fz_document_refs_get, _mupdf.fz_document_refs_set)
2459: (4)               drop_document = property(_mupdf.fz_document_drop_document_get,
_mupdf.fz_document_drop_document_set)
2460: (4)               needs_password = property(_mupdf.fz_document_needs_password_get,
_mupdf.fz_document_needs_password_set)
2461: (4)               authenticate_password =
property(_mupdf.fz_document_authenticate_password_get,
_mupdf.fz_document_authenticate_password_set)
2462: (4)               has_permission = property(_mupdf.fz_document_has_permission_get,
_mupdf.fz_document_has_permission_set)
2463: (4)               load_outline = property(_mupdf.fz_document_load_outline_get,
_mupdf.fz_document_load_outline_set)
2464: (4)               outline_iterator = property(_mupdf.fz_document_outline_iterator_get,
_mupdf.fz_document_outline_iterator_set)
2465: (4)               layout = property(_mupdf.fz_document_layout_get,
_mupdf.fz_document_layout_set)
2466: (4)               make_bookmark = property(_mupdf.fz_document_make_bookmark_get,
_mupdf.fz_document_make_bookmark_set)
2467: (4)               lookup_bookmark = property(_mupdf.fz_document_lookup_bookmark_get,
_mupdf.fz_document_lookup_bookmark_set)
2468: (4)               resolve_link_dest = property(_mupdf.fz_document_resolve_link_dest_get,
_mupdf.fz_document_resolve_link_dest_set)
2469: (4)               format_link_uri = property(_mupdf.fz_document_format_link_uri_get,
_mupdf.fz_document_format_link_uri_set)
2470: (4)               count_chapters = property(_mupdf.fz_document_count_chapters_get,
_mupdf.fz_document_count_chapters_set)
```

```
2471: (4)               count_pages = property(_mupdf.fz_document_count_pages_get,
_mupdf.fz_document_count_pages_set)
2472: (4)               load_page = property(_mupdf.fz_document_load_page_get,
_mupdf.fz_document_load_page_set)
2473: (4)               page_label = property(_mupdf.fz_document_page_label_get,
_mupdf.fz_document_page_label_set)
2474: (4)               lookup_metadata = property(_mupdf.fz_document_lookup_metadata_get,
_mupdf.fz_document_lookup_metadata_set)
2475: (4)               set_metadata = property(_mupdf.fz_document_set_metadata_get,
_mupdf.fz_document_set_metadata_set)
2476: (4)               get_output_intent = property(_mupdf.fz_document_get_output_intent_get,
_mupdf.fz_document_get_output_intent_set)
2477: (4)               output_accelerator = property(_mupdf.fz_document_output_accelerator_get,
_mupdf.fz_document_output_accelerator_set)
2478: (4)               run_structure = property(_mupdf.fz_document_run_structure_get,
_mupdf.fz_document_run_structure_set)
2479: (4)               as_pdf = property(_mupdf.fz_document_as_pdf_get,
_mupdf.fz_document_as_pdf_set)
2480: (4)               did_layout = property(_mupdf.fz_document_did_layout_get,
_mupdf.fz_document_did_layout_set)
2481: (4)               is_reflowable = property(_mupdf.fz_document_is_reflowable_get,
_mupdf.fz_document_is_reflowable_set)
2482: (4)               open = property(_mupdf.fz_document_open_get, _mupdf.fz_document_open_set)
2483: (4)               def __init__(self):
2484: (8)                   _mupdf.fz_document_swiginit(self, _mupdf.new_fz_document())
2485: (4)               __swig_destroy__ = _mupdf.delete_fz_document
2486: (0)           _mupdf.fz_document_swigregister(fz_document)
2487: (0)           class fz_document_handler(object):
2488: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2489: (4)               __repr__ = _swig_repr
2490: (4)               recognize = property(_mupdf.fz_document_handler_recognize_get,
_mupdf.fz_document_handler_recognize_set)
2491: (4)               open = property(_mupdf.fz_document_handler_open_get,
_mupdf.fz_document_handler_open_set)
2492: (4)               extensions = property(_mupdf.fz_document_handler_extensions_get,
_mupdf.fz_document_handler_extensions_set)
2493: (4)               mimetypes = property(_mupdf.fz_document_handler_mimetypes_get,
_mupdf.fz_document_handler_mimetypes_set)
2494: (4)               recognize_content =
property(_mupdf.fz_document_handler_recognize_content_get,
_mupdf.fz_document_handler_recognize_content_set)
2495: (4)               wants_dir = property(_mupdf.fz_document_handler_wants_dir_get,
_mupdf.fz_document_handler_wants_dir_set)
2496: (4)               wants_file = property(_mupdf.fz_document_handler_wants_file_get,
_mupdf.fz_document_handler_wants_file_set)
2497: (4)               fin = property(_mupdf.fz_document_handler_fin_get,
_mupdf.fz_document_handler_fin_set)
2498: (4)               def __init__(self):
2499: (8)                   _mupdf.fz_document_handler_swiginit(self,
_mupdf.new_fz_document_handler())
2500: (4)               __swig_destroy__ = _mupdf.delete_fz_document_handler
2501: (0)           _mupdf.fz_document_handler_swigregister(fz_document_handler)
2502: (0)           class fz_document_writer(object):
2503: (4)               r"""
2504: (4)               Structure is public to allow other structures to
2505: (4)               be derived from it. Do not access members directly.
2506: (4)               """
2507: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2508: (4)               __repr__ = _swig_repr
2509: (4)               begin_page = property(_mupdf.fz_document_writer_begin_page_get,
_mupdf.fz_document_writer_begin_page_set)
2510: (4)               end_page = property(_mupdf.fz_document_writer_end_page_get,
_mupdf.fz_document_writer_end_page_set)
2511: (4)               close_writer = property(_mupdf.fz_document_writer_close_writer_get,
_mupdf.fz_document_writer_close_writer_set)
2512: (4)               drop_writer = property(_mupdf.fz_document_writer_drop_writer_get,
_mupdf.fz_document_writer_drop_writer_set)
```

```
2513: (4)               dev = property(_mupdf.fz_document_writer_dev_get,
_mupdf.fz_document_writer_dev_set)
2514: (4)               def __init__(self):
2515: (8)                   _mupdf.fz_document_writer_swiginit(self,
_mupdf.new_fz_document_writer())
2516: (4)               __swig_destroy__ = _mupdf.delete_fz_document_writer
2517: (0)           _mupdf.fz_document_writer_swigregister(fz_document_writer)
2518: (0)           class fz_band_writer(object):
2519: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2520: (4)               __repr__ = _swig_repr
2521: (4)               drop = property(_mupdf.fz_band_writer_drop_get,
_mupdf.fz_band_writer_drop_set)
2522: (4)               close = property(_mupdf.fz_band_writer_close_get,
_mupdf.fz_band_writer_close_set)
2523: (4)               header = property(_mupdf.fz_band_writer_header_get,
_mupdf.fz_band_writer_header_set)
2524: (4)               band = property(_mupdf.fz_band_writer_band_get,
_mupdf.fz_band_writer_band_set)
2525: (4)               trailer = property(_mupdf.fz_band_writer_trailer_get,
_mupdf.fz_band_writer_trailer_set)
2526: (4)               out = property(_mupdf.fz_band_writer_out_get,
_mupdf.fz_band_writer_out_set)
2527: (4)               w = property(_mupdf.fz_band_writer_w_get, _mupdf.fz_band_writer_w_set)
2528: (4)               h = property(_mupdf.fz_band_writer_h_get, _mupdf.fz_band_writer_h_set)
2529: (4)               n = property(_mupdf.fz_band_writer_n_get, _mupdf.fz_band_writer_n_set)
2530: (4)               s = property(_mupdf.fz_band_writer_s_get, _mupdf.fz_band_writer_s_set)
2531: (4)               alpha = property(_mupdf.fz_band_writer_alpha_get,
_mupdf.fz_band_writer_alpha_set)
2532: (4)               xres = property(_mupdf.fz_band_writer_xres_get,
_mupdf.fz_band_writer_xres_set)
2533: (4)               yres = property(_mupdf.fz_band_writer_yres_get,
_mupdf.fz_band_writer_yres_set)
2534: (4)               pagenum = property(_mupdf.fz_band_writer_pagenum_get,
_mupdf.fz_band_writer_pagenum_set)
2535: (4)               line = property(_mupdf.fz_band_writer_line_get,
_mupdf.fz_band_writer_line_set)
2536: (4)               seps = property(_mupdf.fz_band_writer_seps_get,
_mupdf.fz_band_writer_seps_set)
2537: (4)               def __init__(self):
2538: (8)                   _mupdf.fz_band_writer_swiginit(self, _mupdf.new_fz_band_writer())
2539: (4)               __swig_destroy__ = _mupdf.delete_fz_band_writer
2540: (0)           _mupdf.fz_band_writer_swigregister(fz_band_writer)
2541: (0)           class fz_pcl_options(object):
2542: (4)               r"""    PCL output"""
2543: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2544: (4)               __repr__ = _swig_repr
2545: (4)               features = property(_mupdf.fz_pcl_options_features_get,
_mupdf.fz_pcl_options_features_set)
2546: (4)               odd_page_init = property(_mupdf.fz_pcl_options_odd_page_init_get,
_mupdf.fz_pcl_options_odd_page_init_set)
2547: (4)               even_page_init = property(_mupdf.fz_pcl_options_even_page_init_get,
_mupdf.fz_pcl_options_even_page_init_set)
2548: (4)               tumble = property(_mupdf.fz_pcl_options_tumble_get,
_mupdf.fz_pcl_options_tumble_set)
2549: (4)               duplex_set = property(_mupdf.fz_pcl_options_duplex_set_get,
_mupdf.fz_pcl_options_duplex_set_set)
2550: (4)               duplex = property(_mupdf.fz_pcl_options_duplex_get,
_mupdf.fz_pcl_options_duplex_set)
2551: (4)               paper_size = property(_mupdf.fz_pcl_options_paper_size_get,
_mupdf.fz_pcl_options_paper_size_set)
2552: (4)               manual_feed_set = property(_mupdf.fz_pcl_options_manual_feed_set_get,
_mupdf.fz_pcl_options_manual_feed_set_set)
2553: (4)               manual_feed = property(_mupdf.fz_pcl_options_manual_feed_get,
_mupdf.fz_pcl_options_manual_feed_set)
2554: (4)               media_position_set =
property(_mupdf.fz_pcl_options_media_position_set_get,
_mupdf.fz_pcl_options_media_position_set_set)
```

```
2555: (4)                media_position = property(_mupdf.fz_pcl_options_media_position_get,
_mupdf.fz_pcl_options_media_position_set)
2556: (4)                orientation = property(_mupdf.fz_pcl_options_orientation_get,
_mupdf.fz_pcl_options_orientation_set)
2557: (4)                page_count = property(_mupdf.fz_pcl_options_page_count_get,
_mupdf.fz_pcl_options_page_count_set)
2558: (4)                def __init__(self):
2559: (8)                    _mupdf.fz_pcl_options_swiginit(self, _mupdf.new_fz_pcl_options())
2560: (4)                __swig_destroy__ = _mupdf.delete_fz_pcl_options
2561: (0)            _mupdf.fz_pcl_options_swigregister(fz_pcl_options)
2562: (0)            class fz_pclm_options(object):
2563: (4)                r"""    PCLm output"""
2564: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2565: (4)                __repr__ = _swig_repr
2566: (4)                compress = property(_mupdf.fz_pclm_options_compress_get,
_mupdf.fz_pclm_options_compress_set)
2567: (4)                strip_height = property(_mupdf.fz_pclm_options_strip_height_get,
_mupdf.fz_pclm_options_strip_height_set)
2568: (4)                page_count = property(_mupdf.fz_pclm_options_page_count_get,
_mupdf.fz_pclm_options_page_count_set)
2569: (4)                def __init__(self):
2570: (8)                    _mupdf.fz_pclm_options_swiginit(self, _mupdf.new_fz_pclm_options())
2571: (4)                __swig_destroy__ = _mupdf.delete_fz_pclm_options
2572: (0)            _mupdf.fz_pclm_options_swigregister(fz_pclm_options)
2573: (0)            class fz_pdfocr_options(object):
2574: (4)                r"""    PDFOCR output"""
2575: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2576: (4)                __repr__ = _swig_repr
2577: (4)                compress = property(_mupdf.fz_pdfocr_options_compress_get,
_mupdf.fz_pdfocr_options_compress_set)
2578: (4)                strip_height = property(_mupdf.fz_pdfocr_options_strip_height_get,
_mupdf.fz_pdfocr_options_strip_height_set)
2579: (4)                language = property(_mupdf.fz_pdfocr_options_language_get,
_mupdf.fz_pdfocr_options_language_set)
2580: (4)                datadir = property(_mupdf.fz_pdfocr_options_datadir_get,
_mupdf.fz_pdfocr_options_datadir_set)
2581: (4)                page_count = property(_mupdf.fz_pdfocr_options_page_count_get,
_mupdf.fz_pdfocr_options_page_count_set)
2582: (4)                def __init__(self):
2583: (8)                    _mupdf.fz_pdfocr_options_swiginit(self,
_mupdf.new_fz_pdfocr_options())
2584: (4)                __swig_destroy__ = _mupdf.delete_fz_pdfocr_options
2585: (0)            _mupdf.fz_pdfocr_options_swigregister(fz_pdfocr_options)
2586: (0)            class fz_pwg_options(object):
2587: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2588: (4)                __repr__ = _swig_repr
2589: (4)                media_class = property(_mupdf.fz_pwg_options_media_class_get,
_mupdf.fz_pwg_options_media_class_set)
2590: (4)                media_color = property(_mupdf.fz_pwg_options_media_color_get,
_mupdf.fz_pwg_options_media_color_set)
2591: (4)                media_type = property(_mupdf.fz_pwg_options_media_type_get,
_mupdf.fz_pwg_options_media_type_set)
2592: (4)                output_type = property(_mupdf.fz_pwg_options_output_type_get,
_mupdf.fz_pwg_options_output_type_set)
2593: (4)                advance_distance = property(_mupdf.fz_pwg_options_advance_distance_get,
_mupdf.fz_pwg_options_advance_distance_set)
2594: (4)                advance_media = property(_mupdf.fz_pwg_options_advance_media_get,
_mupdf.fz_pwg_options_advance_media_set)
2595: (4)                collate = property(_mupdf.fz_pwg_options_collate_get,
_mupdf.fz_pwg_options_collate_set)
2596: (4)                cut_media = property(_mupdf.fz_pwg_options_cut_media_get,
_mupdf.fz_pwg_options_cut_media_set)
2597: (4)                duplex = property(_mupdf.fz_pwg_options_duplex_get,
_mupdf.fz_pwg_options_duplex_set)
2598: (4)                insert_sheet = property(_mupdf.fz_pwg_options_insert_sheet_get,
_mupdf.fz_pwg_options_insert_sheet_set)
```

```
2599: (4)               jog = property(_mupdf.fz_pwg_options_jog_get,
_mupdf.fz_pwg_options_jog_set)
2600: (4)               leading_edge = property(_mupdf.fz_pwg_options_leading_edge_get,
_mupdf.fz_pwg_options_leading_edge_set)
2601: (4)               manual_feed = property(_mupdf.fz_pwg_options_manual_feed_get,
_mupdf.fz_pwg_options_manual_feed_set)
2602: (4)               media_position = property(_mupdf.fz_pwg_options_media_position_get,
_mupdf.fz_pwg_options_media_position_set)
2603: (4)               media_weight = property(_mupdf.fz_pwg_options_media_weight_get,
_mupdf.fz_pwg_options_media_weight_set)
2604: (4)               mirror_print = property(_mupdf.fz_pwg_options_mirror_print_get,
_mupdf.fz_pwg_options_mirror_print_set)
2605: (4)               negative_print = property(_mupdf.fz_pwg_options_negative_print_get,
_mupdf.fz_pwg_options_negative_print_set)
2606: (4)               num_copies = property(_mupdf.fz_pwg_options_num_copies_get,
_mupdf.fz_pwg_options_num_copies_set)
2607: (4)               orientation = property(_mupdf.fz_pwg_options_orientation_get,
_mupdf.fz_pwg_options_orientation_set)
2608: (4)               output_face_up = property(_mupdf.fz_pwg_options_output_face_up_get,
_mupdf.fz_pwg_options_output_face_up_set)
2609: (4)               PageSize = property(_mupdf.fz_pwg_options_PageSize_get,
_mupdf.fz_pwg_options_PageSize_set)
2610: (4)               separations = property(_mupdf.fz_pwg_options_separations_get,
_mupdf.fz_pwg_options_separations_set)
2611: (4)               tray_switch = property(_mupdf.fz_pwg_options_tray_switch_get,
_mupdf.fz_pwg_options_tray_switch_set)
2612: (4)               tumble = property(_mupdf.fz_pwg_options_tumble_get,
_mupdf.fz_pwg_options_tumble_set)
2613: (4)               media_type_num = property(_mupdf.fz_pwg_options_media_type_num_get,
_mupdf.fz_pwg_options_media_type_num_set)
2614: (4)               compression = property(_mupdf.fz_pwg_options_compression_get,
_mupdf.fz_pwg_options_compression_set)
2615: (4)               row_count = property(_mupdf.fz_pwg_options_row_count_get,
_mupdf.fz_pwg_options_row_count_set)
2616: (4)               row_feed = property(_mupdf.fz_pwg_options_row_feed_get,
_mupdf.fz_pwg_options_row_feed_set)
2617: (4)               row_step = property(_mupdf.fz_pwg_options_row_step_get,
_mupdf.fz_pwg_options_row_step_set)
2618: (4)               rendering_intent = property(_mupdf.fz_pwg_options_rendering_intent_get,
_mupdf.fz_pwg_options_rendering_intent_set)
2619: (4)               page_size_name = property(_mupdf.fz_pwg_options_page_size_name_get,
_mupdf.fz_pwg_options_page_size_name_set)
2620: (4)               def __init__(self):
2621: (8)                   _mupdf.fz_pwg_options_swiginit(self, _mupdf.new_fz_pwg_options())
2622: (4)               __swig_destroy__ = _mupdf.delete_fz_pwg_options
2623: (0)           _mupdf.fz_pwg_options_swigregister(fz_pwg_options)
2624: (0)           FZ_SVG_TEXT_AS_PATH = _mupdf.FZ_SVG_TEXT_AS_PATH
2625: (0)           FZ_SVG_TEXT_AS_TEXT = _mupdf.FZ_SVG_TEXT_AS_TEXT
2626: (0)           FZ_PLACE_STORY_FLAG_NO_OVERFLOW = _mupdf.FZ_PLACE_STORY_FLAG_NO_OVERFLOW
2627: (0)           FZ_PLACE_STORY_RETURN_ALL_FITTED = _mupdf.FZ_PLACE_STORY_RETURN_ALL_FITTED
2628: (0)           FZ_PLACE_STORY_RETURN_OVERFLOW_WIDTH =
_mupdf.FZ_PLACE_STORY_RETURN_OVERFLOW_WIDTH
2629: (0)           class fz_story_element_position(object):
2630: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2631: (4)               __repr__ = _swig_repr
2632: (4)               depth = property(_mupdf.fz_story_element_position_depth_get,
_mupdf.fz_story_element_position_depth_set)
2633: (4)               heading = property(_mupdf.fz_story_element_position_heading_get,
_mupdf.fz_story_element_position_heading_set)
2634: (4)               id = property(_mupdf.fz_story_element_position_id_get,
_mupdf.fz_story_element_position_id_set)
2635: (4)               href = property(_mupdf.fz_story_element_position_href_get,
_mupdf.fz_story_element_position_href_set)
2636: (4)               rect = property(_mupdf.fz_story_element_position_rect_get,
_mupdf.fz_story_element_position_rect_set)
2637: (4)               text = property(_mupdf.fz_story_element_position_text_get,
_mupdf.fz_story_element_position_text_set)
2638: (4)               open_close = property(_mupdf.fz_story_element_position_open_close_get,
```

```
_mupdf.fz_story_element_position_open_close_set)
2639: (4)               rectangle_num =
property(_mupdf.fz_story_element_position_rectangle_num_get,
_mupdf.fz_story_element_position_rectangle_num_set)
2640: (4)               def __init__(self):
2641: (8)                   _mupdf.fz_story_element_position_swiginit(self,
_mupdf.new_fz_story_element_position())
2642: (4)               __swig_destroy__ = _mupdf.delete_fz_story_element_position
2643: (0)           _mupdf.fz_story_element_position_swigregister(fz_story_element_position)
2644: (0)           class fz_write_story_position(object):
2645: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2646: (4)               __repr__ = _swig_repr
2647: (4)               element = property(_mupdf.fz_write_story_position_element_get,
_mupdf.fz_write_story_position_element_set)
2648: (4)               page_num = property(_mupdf.fz_write_story_position_page_num_get,
_mupdf.fz_write_story_position_page_num_set)
2649: (4)               def __init__(self):
2650: (8)                   _mupdf.fz_write_story_position_swiginit(self,
_mupdf.new_fz_write_story_position())
2651: (4)               __swig_destroy__ = _mupdf.delete_fz_write_story_position
2652: (0)           _mupdf.fz_write_story_position_swigregister(fz_write_story_position)
2653: (0)           class fz_write_story_positions(object):
2654: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2655: (4)               __repr__ = _swig_repr
2656: (4)               positions = property(_mupdf.fz_write_story_positions_positions_get,
_mupdf.fz_write_story_positions_positions_set)
2657: (4)               num = property(_mupdf.fz_write_story_positions_num_get,
_mupdf.fz_write_story_positions_num_set)
2658: (4)               def __init__(self):
2659: (8)                   _mupdf.fz_write_story_positions_swiginit(self,
_mupdf.new_fz_write_story_positions())
2660: (4)               __swig_destroy__ = _mupdf.delete_fz_write_story_positions
2661: (0)           _mupdf.fz_write_story_positions_swigregister(fz_write_story_positions)
2662: (0)           PDF_MAX_OBJECT_NUMBER = _mupdf.PDF_MAX_OBJECT_NUMBER
2663: (0)           PDF_MAX_GEN_NUMBER = _mupdf.PDF_MAX_GEN_NUMBER
2664: (0)           class pdf_cycle_list(object):
2665: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2666: (4)               __repr__ = _swig_repr
2667: (4)               up = property(_mupdf.pdf_cycle_list_up_get, _mupdf.pdf_cycle_list_up_set)
2668: (4)               num = property(_mupdf.pdf_cycle_list_num_get,
_mupdf.pdf_cycle_list_num_set)
2669: (4)               def __init__(self):
2670: (8)                   _mupdf.pdf_cycle_list_swiginit(self, _mupdf.new_pdf_cycle_list())
2671: (4)               __swig_destroy__ = _mupdf.delete_pdf_cycle_list
2672: (0)           _mupdf.pdf_cycle_list_swigregister(pdf_cycle_list)
2673: (0)           class pdf_mark_bits(object):
2674: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2675: (4)               __repr__ = _swig_repr
2676: (4)               len = property(_mupdf.pdf_mark_bits_len_get, _mupdf.pdf_mark_bits_len_set)
2677: (4)               bits = property(_mupdf.pdf_mark_bits_bits_get,
_mupdf.pdf_mark_bits_bits_set)
2678: (4)               def __init__(self):
2679: (8)                   _mupdf.pdf_mark_bits_swiginit(self, _mupdf.new_pdf_mark_bits())
2680: (4)               __swig_destroy__ = _mupdf.delete_pdf_mark_bits
2681: (0)           _mupdf.pdf_mark_bits_swigregister(pdf_mark_bits)
2682: (0)           class pdf_mark_list(object):
2683: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
2684: (4)               __repr__ = _swig_repr
2685: (4)               len = property(_mupdf.pdf_mark_list_len_get, _mupdf.pdf_mark_list_len_set)
2686: (4)               max = property(_mupdf.pdf_mark_list_max_get, _mupdf.pdf_mark_list_max_set)
2687: (4)               list = property(_mupdf.pdf_mark_list_list_get,
_mupdf.pdf_mark_list_list_set)
2688: (4)               local_list = property(_mupdf.pdf_mark_list_local_list_get,
_mupdf.pdf_mark_list_local_list_set)
```

```
(4)                 def __init__(self):
(8)                     _mupdf.pdf_mark_list_swiginit(self, _mupdf.new_pdf_mark_list())
(4)                 __swig_destroy__ = _mupdf.delete_pdf_mark_list
(0)             _mupdf.pdf_mark_list_swigregister(pdf_mark_list)
(0)             PDF_ENUM_NULL = _mupdf.PDF_ENUM_NULL
(0)             PDF_ENUM_TRUE = _mupdf.PDF_ENUM_TRUE
(0)             PDF_ENUM_FALSE = _mupdf.PDF_ENUM_FALSE
(0)             PDF_ENUM_NAME_1_2 = _mupdf.PDF_ENUM_NAME_1_2
(0)             PDF_ENUM_NAME_1_5 = _mupdf.PDF_ENUM_NAME_1_5
(0)             PDF_ENUM_NAME_3D = _mupdf.PDF_ENUM_NAME_3D
(0)             PDF_ENUM_NAME_A = _mupdf.PDF_ENUM_NAME_A
(0)             PDF_ENUM_NAME_A85 = _mupdf.PDF_ENUM_NAME_A85
(0)             PDF_ENUM_NAME_AA = _mupdf.PDF_ENUM_NAME_AA
(0)             PDF_ENUM_NAME_AC = _mupdf.PDF_ENUM_NAME_AC
(0)             PDF_ENUM_NAME_AESV2 = _mupdf.PDF_ENUM_NAME_AESV2
(0)             PDF_ENUM_NAME_AESV3 = _mupdf.PDF_ENUM_NAME_AESV3
(0)             PDF_ENUM_NAME_AHx = _mupdf.PDF_ENUM_NAME_AHx
(0)             PDF_ENUM_NAME_AP = _mupdf.PDF_ENUM_NAME_AP
(0)             PDF_ENUM_NAME_AS = _mupdf.PDF_ENUM_NAME_AS
(0)             PDF_ENUM_NAME_ASCII85Decode = _mupdf.PDF_ENUM_NAME_ASCII85Decode
(0)             PDF_ENUM_NAME_ASCIIHexDecode = _mupdf.PDF_ENUM_NAME_ASCIIHexDecode
(0)             PDF_ENUM_NAME_AcroForm = _mupdf.PDF_ENUM_NAME_AcroForm
(0)             PDF_ENUM_NAME_Action = _mupdf.PDF_ENUM_NAME_Action
(0)             PDF_ENUM_NAME_ActualText = _mupdf.PDF_ENUM_NAME_ActualText
(0)             PDF_ENUM_NAME_Adobe_PPKLite = _mupdf.PDF_ENUM_NAME_Adobe_PPKLite
(0)             PDF_ENUM_NAME_All = _mupdf.PDF_ENUM_NAME_All
(0)             PDF_ENUM_NAME_AllOff = _mupdf.PDF_ENUM_NAME_AllOff
(0)             PDF_ENUM_NAME_AllOn = _mupdf.PDF_ENUM_NAME_AllOn
(0)             PDF_ENUM_NAME_Alpha = _mupdf.PDF_ENUM_NAME_Alpha
(0)             PDF_ENUM_NAME_Alt = _mupdf.PDF_ENUM_NAME_Alt
(0)             PDF_ENUM_NAME_Alternate = _mupdf.PDF_ENUM_NAME_Alternate
(0)             PDF_ENUM_NAME_Annot = _mupdf.PDF_ENUM_NAME_Annot
(0)             PDF_ENUM_NAME_Annots = _mupdf.PDF_ENUM_NAME_Annots
(0)             PDF_ENUM_NAME_AnyOff = _mupdf.PDF_ENUM_NAME_AnyOff
(0)             PDF_ENUM_NAME_App = _mupdf.PDF_ENUM_NAME_App
(0)             PDF_ENUM_NAME_Approved = _mupdf.PDF_ENUM_NAME_Approved
(0)             PDF_ENUM_NAME_Art = _mupdf.PDF_ENUM_NAME_Art
(0)             PDF_ENUM_NAME_ArtBox = _mupdf.PDF_ENUM_NAME_ArtBox
(0)             PDF_ENUM_NAME_Artifact = _mupdf.PDF_ENUM_NAME_Artifact
(0)             PDF_ENUM_NAME_AsIs = _mupdf.PDF_ENUM_NAME_AsIs
(0)             PDF_ENUM_NAME_Ascent = _mupdf.PDF_ENUM_NAME_Ascent
(0)             PDF_ENUM_NAME_Aside = _mupdf.PDF_ENUM_NAME_Aside
(0)             PDF_ENUM_NAME_AuthEvent = _mupdf.PDF_ENUM_NAME_AuthEvent
(0)             PDF_ENUM_NAME_Author = _mupdf.PDF_ENUM_NAME_Author
(0)             PDF_ENUM_NAME_B = _mupdf.PDF_ENUM_NAME_B
(0)             PDF_ENUM_NAME_BBox = _mupdf.PDF_ENUM_NAME_BBox
(0)             PDF_ENUM_NAME_BC = _mupdf.PDF_ENUM_NAME_BC
(0)             PDF_ENUM_NAME_BE = _mupdf.PDF_ENUM_NAME_BE
(0)             PDF_ENUM_NAME_BG = _mupdf.PDF_ENUM_NAME_BG
(0)             PDF_ENUM_NAME_BM = _mupdf.PDF_ENUM_NAME_BM
(0)             PDF_ENUM_NAME_BPC = _mupdf.PDF_ENUM_NAME_BPC
(0)             PDF_ENUM_NAME_BS = _mupdf.PDF_ENUM_NAME_BS
(0)             PDF_ENUM_NAME_Background = _mupdf.PDF_ENUM_NAME_Background
(0)             PDF_ENUM_NAME_BaseEncoding = _mupdf.PDF_ENUM_NAME_BaseEncoding
(0)             PDF_ENUM_NAME_BaseFont = _mupdf.PDF_ENUM_NAME_BaseFont
(0)             PDF_ENUM_NAME_BaseState = _mupdf.PDF_ENUM_NAME_BaseState
(0)             PDF_ENUM_NAME_BibEntry = _mupdf.PDF_ENUM_NAME_BibEntry
(0)             PDF_ENUM_NAME_BitsPerComponent = _mupdf.PDF_ENUM_NAME_BitsPerComponent
(0)             PDF_ENUM_NAME_BitsPerCoordinate = _mupdf.PDF_ENUM_NAME_BitsPerCoordinate
(0)             PDF_ENUM_NAME_BitsPerFlag = _mupdf.PDF_ENUM_NAME_BitsPerFlag
(0)             PDF_ENUM_NAME_BitsPerSample = _mupdf.PDF_ENUM_NAME_BitsPerSample
(0)             PDF_ENUM_NAME_BlackIs1 = _mupdf.PDF_ENUM_NAME_BlackIs1
(0)             PDF_ENUM_NAME_BlackPoint = _mupdf.PDF_ENUM_NAME_BlackPoint
(0)             PDF_ENUM_NAME_BleedBox = _mupdf.PDF_ENUM_NAME_BleedBox
(0)             PDF_ENUM_NAME_Blinds = _mupdf.PDF_ENUM_NAME_Blinds
(0)             PDF_ENUM_NAME_BlockQuote = _mupdf.PDF_ENUM_NAME_BlockQuote
(0)             PDF_ENUM_NAME_Border = _mupdf.PDF_ENUM_NAME_Border
(0)             PDF_ENUM_NAME_Bounds = _mupdf.PDF_ENUM_NAME_Bounds
(0)             PDF_ENUM_NAME_Box = _mupdf.PDF_ENUM_NAME_Box
```

```
2758: (0)            PDF_ENUM_NAME_Bt = _mupdf.PDF_ENUM_NAME_Bt
2759: (0)            PDF_ENUM_NAME_Btn = _mupdf.PDF_ENUM_NAME_Btn
2760: (0)            PDF_ENUM_NAME_Butt = _mupdf.PDF_ENUM_NAME_Butt
2761: (0)            PDF_ENUM_NAME_ByteRange = _mupdf.PDF_ENUM_NAME_ByteRange
2762: (0)            PDF_ENUM_NAME_C = _mupdf.PDF_ENUM_NAME_C
2763: (0)            PDF_ENUM_NAME_C0 = _mupdf.PDF_ENUM_NAME_C0
2764: (0)            PDF_ENUM_NAME_C1 = _mupdf.PDF_ENUM_NAME_C1
2765: (0)            PDF_ENUM_NAME_CA = _mupdf.PDF_ENUM_NAME_CA
2766: (0)            PDF_ENUM_NAME_CCF = _mupdf.PDF_ENUM_NAME_CCF
2767: (0)            PDF_ENUM_NAME_CCITTFaxDecode = _mupdf.PDF_ENUM_NAME_CCITTFaxDecode
2768: (0)            PDF_ENUM_NAME_CF = _mupdf.PDF_ENUM_NAME_CF
2769: (0)            PDF_ENUM_NAME_CFM = _mupdf.PDF_ENUM_NAME_CFM
2770: (0)            PDF_ENUM_NAME_CI = _mupdf.PDF_ENUM_NAME_CI
2771: (0)            PDF_ENUM_NAME_CIDFontType0 = _mupdf.PDF_ENUM_NAME_CIDFontType0
2772: (0)            PDF_ENUM_NAME_CIDFontType0C = _mupdf.PDF_ENUM_NAME_CIDFontType0C
2773: (0)            PDF_ENUM_NAME_CIDFontType2 = _mupdf.PDF_ENUM_NAME_CIDFontType2
2774: (0)            PDF_ENUM_NAME_CIDSystemInfo = _mupdf.PDF_ENUM_NAME_CIDSystemInfo
2775: (0)            PDF_ENUM_NAME_CIDToGIDMap = _mupdf.PDF_ENUM_NAME_CIDToGIDMap
2776: (0)            PDF_ENUM_NAME_CMYK = _mupdf.PDF_ENUM_NAME_CMYK
2777: (0)            PDF_ENUM_NAME_CS = _mupdf.PDF_ENUM_NAME_CS
2778: (0)            PDF_ENUM_NAME_CalCMYK = _mupdf.PDF_ENUM_NAME_CalCMYK
2779: (0)            PDF_ENUM_NAME_CalGray = _mupdf.PDF_ENUM_NAME_CalGray
2780: (0)            PDF_ENUM_NAME_CalRGB = _mupdf.PDF_ENUM_NAME_CalRGB
2781: (0)            PDF_ENUM_NAME_CapHeight = _mupdf.PDF_ENUM_NAME_CapHeight
2782: (0)            PDF_ENUM_NAME_Caption = _mupdf.PDF_ENUM_NAME_Caption
2783: (0)            PDF_ENUM_NAME_Caret = _mupdf.PDF_ENUM_NAME_Caret
2784: (0)            PDF_ENUM_NAME_Catalog = _mupdf.PDF_ENUM_NAME_Catalog
2785: (0)            PDF_ENUM_NAME_Cert = _mupdf.PDF_ENUM_NAME_Cert
2786: (0)            PDF_ENUM_NAME_Ch = _mupdf.PDF_ENUM_NAME_Ch
2787: (0)            PDF_ENUM_NAME_Changes = _mupdf.PDF_ENUM_NAME_Changes
2788: (0)            PDF_ENUM_NAME_CharProcs = _mupdf.PDF_ENUM_NAME_CharProcs
2789: (0)            PDF_ENUM_NAME_CheckSum = _mupdf.PDF_ENUM_NAME_CheckSum
2790: (0)            PDF_ENUM_NAME_Circle = _mupdf.PDF_ENUM_NAME_Circle
2791: (0)            PDF_ENUM_NAME_ClosedArrow = _mupdf.PDF_ENUM_NAME_ClosedArrow
2792: (0)            PDF_ENUM_NAME_Code = _mupdf.PDF_ENUM_NAME_Code
2793: (0)            PDF_ENUM_NAME_Collection = _mupdf.PDF_ENUM_NAME_Collection
2794: (0)            PDF_ENUM_NAME_ColorSpace = _mupdf.PDF_ENUM_NAME_ColorSpace
2795: (0)            PDF_ENUM_NAME_ColorTransform = _mupdf.PDF_ENUM_NAME_ColorTransform
2796: (0)            PDF_ENUM_NAME_Colorants = _mupdf.PDF_ENUM_NAME_Colorants
2797: (0)            PDF_ENUM_NAME_Colors = _mupdf.PDF_ENUM_NAME_Colors
2798: (0)            PDF_ENUM_NAME_Columns = _mupdf.PDF_ENUM_NAME_Columns
2799: (0)            PDF_ENUM_NAME_Confidential = _mupdf.PDF_ENUM_NAME_Confidential
2800: (0)            PDF_ENUM_NAME_Configs = _mupdf.PDF_ENUM_NAME_Configs
2801: (0)            PDF_ENUM_NAME_ContactInfo = _mupdf.PDF_ENUM_NAME_ContactInfo
2802: (0)            PDF_ENUM_NAME_Contents = _mupdf.PDF_ENUM_NAME_Contents
2803: (0)            PDF_ENUM_NAME_Coords = _mupdf.PDF_ENUM_NAME_Coords
2804: (0)            PDF_ENUM_NAME_Count = _mupdf.PDF_ENUM_NAME_Count
2805: (0)            PDF_ENUM_NAME_Cover = _mupdf.PDF_ENUM_NAME_Cover
2806: (0)            PDF_ENUM_NAME_CreationDate = _mupdf.PDF_ENUM_NAME_CreationDate
2807: (0)            PDF_ENUM_NAME_Creator = _mupdf.PDF_ENUM_NAME_Creator
2808: (0)            PDF_ENUM_NAME_CropBox = _mupdf.PDF_ENUM_NAME_CropBox
2809: (0)            PDF_ENUM_NAME_Crypt = _mupdf.PDF_ENUM_NAME_Crypt
2810: (0)            PDF_ENUM_NAME_D = _mupdf.PDF_ENUM_NAME_D
2811: (0)            PDF_ENUM_NAME_DA = _mupdf.PDF_ENUM_NAME_DA
2812: (0)            PDF_ENUM_NAME_DC = _mupdf.PDF_ENUM_NAME_DC
2813: (0)            PDF_ENUM_NAME_DCT = _mupdf.PDF_ENUM_NAME_DCT
2814: (0)            PDF_ENUM_NAME_DCTDecode = _mupdf.PDF_ENUM_NAME_DCTDecode
2815: (0)            PDF_ENUM_NAME_DL = _mupdf.PDF_ENUM_NAME_DL
2816: (0)            PDF_ENUM_NAME_DOS = _mupdf.PDF_ENUM_NAME_DOS
2817: (0)            PDF_ENUM_NAME_DP = _mupdf.PDF_ENUM_NAME_DP
2818: (0)            PDF_ENUM_NAME_DR = _mupdf.PDF_ENUM_NAME_DR
2819: (0)            PDF_ENUM_NAME_DS = _mupdf.PDF_ENUM_NAME_DS
2820: (0)            PDF_ENUM_NAME_DV = _mupdf.PDF_ENUM_NAME_DV
2821: (0)            PDF_ENUM_NAME_DW = _mupdf.PDF_ENUM_NAME_DW
2822: (0)            PDF_ENUM_NAME_DW2 = _mupdf.PDF_ENUM_NAME_DW2
2823: (0)            PDF_ENUM_NAME_DamagedRowsBeforeError =
_mupdf.PDF_ENUM_NAME_DamagedRowsBeforeError
2824: (0)            PDF_ENUM_NAME_Data = _mupdf.PDF_ENUM_NAME_Data
2825: (0)            PDF_ENUM_NAME_Date = _mupdf.PDF_ENUM_NAME_Date
```

```
(0)            PDF_ENUM_NAME_Decode = _mupdf.PDF_ENUM_NAME_Decode
(0)            PDF_ENUM_NAME_DecodeParms = _mupdf.PDF_ENUM_NAME_DecodeParms
(0)            PDF_ENUM_NAME_Default = _mupdf.PDF_ENUM_NAME_Default
(0)            PDF_ENUM_NAME_DefaultCMYK = _mupdf.PDF_ENUM_NAME_DefaultCMYK
(0)            PDF_ENUM_NAME_DefaultGray = _mupdf.PDF_ENUM_NAME_DefaultGray
(0)            PDF_ENUM_NAME_DefaultRGB = _mupdf.PDF_ENUM_NAME_DefaultRGB
(0)            PDF_ENUM_NAME_Departmental = _mupdf.PDF_ENUM_NAME_Departmental
(0)            PDF_ENUM_NAME_Desc = _mupdf.PDF_ENUM_NAME_Desc
(0)            PDF_ENUM_NAME_DescendantFonts = _mupdf.PDF_ENUM_NAME_DescendantFonts
(0)            PDF_ENUM_NAME_Descent = _mupdf.PDF_ENUM_NAME_Descent
(0)            PDF_ENUM_NAME_Design = _mupdf.PDF_ENUM_NAME_Design
(0)            PDF_ENUM_NAME_Dest = _mupdf.PDF_ENUM_NAME_Dest
(0)            PDF_ENUM_NAME_DestOutputProfile = _mupdf.PDF_ENUM_NAME_DestOutputProfile
(0)            PDF_ENUM_NAME_Dests = _mupdf.PDF_ENUM_NAME_Dests
(0)            PDF_ENUM_NAME_DeviceCMYK = _mupdf.PDF_ENUM_NAME_DeviceCMYK
(0)            PDF_ENUM_NAME_DeviceGray = _mupdf.PDF_ENUM_NAME_DeviceGray
(0)            PDF_ENUM_NAME_DeviceN = _mupdf.PDF_ENUM_NAME_DeviceN
(0)            PDF_ENUM_NAME_DeviceRGB = _mupdf.PDF_ENUM_NAME_DeviceRGB
(0)            PDF_ENUM_NAME_Di = _mupdf.PDF_ENUM_NAME_Di
(0)            PDF_ENUM_NAME_Diamond = _mupdf.PDF_ENUM_NAME_Diamond
(0)            PDF_ENUM_NAME_Differences = _mupdf.PDF_ENUM_NAME_Differences
(0)            PDF_ENUM_NAME_DigestLocation = _mupdf.PDF_ENUM_NAME_DigestLocation
(0)            PDF_ENUM_NAME_DigestMethod = _mupdf.PDF_ENUM_NAME_DigestMethod
(0)            PDF_ENUM_NAME_DigestValue = _mupdf.PDF_ENUM_NAME_DigestValue
(0)            PDF_ENUM_NAME_Dissolve = _mupdf.PDF_ENUM_NAME_Dissolve
(0)            PDF_ENUM_NAME_Div = _mupdf.PDF_ENUM_NAME_Div
(0)            PDF_ENUM_NAME_Dm = _mupdf.PDF_ENUM_NAME_Dm
(0)            PDF_ENUM_NAME_DocMDP = _mupdf.PDF_ENUM_NAME_DocMDP
(0)            PDF_ENUM_NAME_Document = _mupdf.PDF_ENUM_NAME_Document
(0)            PDF_ENUM_NAME_DocumentFragment = _mupdf.PDF_ENUM_NAME_DocumentFragment
(0)            PDF_ENUM_NAME_Domain = _mupdf.PDF_ENUM_NAME_Domain
(0)            PDF_ENUM_NAME_Draft = _mupdf.PDF_ENUM_NAME_Draft
(0)            PDF_ENUM_NAME_Dur = _mupdf.PDF_ENUM_NAME_Dur
(0)            PDF_ENUM_NAME_E = _mupdf.PDF_ENUM_NAME_E
(0)            PDF_ENUM_NAME_EF = _mupdf.PDF_ENUM_NAME_EF
(0)            PDF_ENUM_NAME_EarlyChange = _mupdf.PDF_ENUM_NAME_EarlyChange
(0)            PDF_ENUM_NAME_Em = _mupdf.PDF_ENUM_NAME_Em
(0)            PDF_ENUM_NAME_EmbeddedFile = _mupdf.PDF_ENUM_NAME_EmbeddedFile
(0)            PDF_ENUM_NAME_EmbeddedFiles = _mupdf.PDF_ENUM_NAME_EmbeddedFiles
(0)            PDF_ENUM_NAME_Encode = _mupdf.PDF_ENUM_NAME_Encode
(0)            PDF_ENUM_NAME_EncodedByteAlign = _mupdf.PDF_ENUM_NAME_EncodedByteAlign
(0)            PDF_ENUM_NAME_Encoding = _mupdf.PDF_ENUM_NAME_Encoding
(0)            PDF_ENUM_NAME_Encrypt = _mupdf.PDF_ENUM_NAME_Encrypt
(0)            PDF_ENUM_NAME_EncryptMetadata = _mupdf.PDF_ENUM_NAME_EncryptMetadata
(0)            PDF_ENUM_NAME_EndOfBlock = _mupdf.PDF_ENUM_NAME_EndOfBlock
(0)            PDF_ENUM_NAME_EndOfLine = _mupdf.PDF_ENUM_NAME_EndOfLine
(0)            PDF_ENUM_NAME_Exclude = _mupdf.PDF_ENUM_NAME_Exclude
(0)            PDF_ENUM_NAME_Experimental = _mupdf.PDF_ENUM_NAME_Experimental
(0)            PDF_ENUM_NAME_Expired = _mupdf.PDF_ENUM_NAME_Expired
(0)            PDF_ENUM_NAME_ExtGState = _mupdf.PDF_ENUM_NAME_ExtGState
(0)            PDF_ENUM_NAME_Extend = _mupdf.PDF_ENUM_NAME_Extend
(0)            PDF_ENUM_NAME_F = _mupdf.PDF_ENUM_NAME_F
(0)            PDF_ENUM_NAME_FENote = _mupdf.PDF_ENUM_NAME_FENote
(0)            PDF_ENUM_NAME_FL = _mupdf.PDF_ENUM_NAME_FL
(0)            PDF_ENUM_NAME_FRM = _mupdf.PDF_ENUM_NAME_FRM
(0)            PDF_ENUM_NAME_FS = _mupdf.PDF_ENUM_NAME_FS
(0)            PDF_ENUM_NAME_FT = _mupdf.PDF_ENUM_NAME_FT
(0)            PDF_ENUM_NAME_Fade = _mupdf.PDF_ENUM_NAME_Fade
(0)            PDF_ENUM_NAME_Ff = _mupdf.PDF_ENUM_NAME_Ff
(0)            PDF_ENUM_NAME_FieldMDP = _mupdf.PDF_ENUM_NAME_FieldMDP
(0)            PDF_ENUM_NAME_Fields = _mupdf.PDF_ENUM_NAME_Fields
(0)            PDF_ENUM_NAME_Figure = _mupdf.PDF_ENUM_NAME_Figure
(0)            PDF_ENUM_NAME_FileAttachment = _mupdf.PDF_ENUM_NAME_FileAttachment
(0)            PDF_ENUM_NAME_FileSize = _mupdf.PDF_ENUM_NAME_FileSize
(0)            PDF_ENUM_NAME_Filespec = _mupdf.PDF_ENUM_NAME_Filespec
(0)            PDF_ENUM_NAME_Filter = _mupdf.PDF_ENUM_NAME_Filter
(0)            PDF_ENUM_NAME_Final = _mupdf.PDF_ENUM_NAME_Final
(0)            PDF_ENUM_NAME_Fingerprint = _mupdf.PDF_ENUM_NAME_Fingerprint
(0)            PDF_ENUM_NAME_First = _mupdf.PDF_ENUM_NAME_First
```

```
(0)            PDF_ENUM_NAME_FirstChar = _mupdf.PDF_ENUM_NAME_FirstChar
(0)            PDF_ENUM_NAME_FirstPage = _mupdf.PDF_ENUM_NAME_FirstPage
(0)            PDF_ENUM_NAME_Fit = _mupdf.PDF_ENUM_NAME_Fit
(0)            PDF_ENUM_NAME_FitB = _mupdf.PDF_ENUM_NAME_FitB
(0)            PDF_ENUM_NAME_FitBH = _mupdf.PDF_ENUM_NAME_FitBH
(0)            PDF_ENUM_NAME_FitBV = _mupdf.PDF_ENUM_NAME_FitBV
(0)            PDF_ENUM_NAME_FitH = _mupdf.PDF_ENUM_NAME_FitH
(0)            PDF_ENUM_NAME_FitR = _mupdf.PDF_ENUM_NAME_FitR
(0)            PDF_ENUM_NAME_FitV = _mupdf.PDF_ENUM_NAME_FitV
(0)            PDF_ENUM_NAME_Fl = _mupdf.PDF_ENUM_NAME_Fl
(0)            PDF_ENUM_NAME_Flags = _mupdf.PDF_ENUM_NAME_Flags
(0)            PDF_ENUM_NAME_FlateDecode = _mupdf.PDF_ENUM_NAME_FlateDecode
(0)            PDF_ENUM_NAME_Fly = _mupdf.PDF_ENUM_NAME_Fly
(0)            PDF_ENUM_NAME_Font = _mupdf.PDF_ENUM_NAME_Font
(0)            PDF_ENUM_NAME_FontBBox = _mupdf.PDF_ENUM_NAME_FontBBox
(0)            PDF_ENUM_NAME_FontDescriptor = _mupdf.PDF_ENUM_NAME_FontDescriptor
(0)            PDF_ENUM_NAME_FontFile = _mupdf.PDF_ENUM_NAME_FontFile
(0)            PDF_ENUM_NAME_FontFile2 = _mupdf.PDF_ENUM_NAME_FontFile2
(0)            PDF_ENUM_NAME_FontFile3 = _mupdf.PDF_ENUM_NAME_FontFile3
(0)            PDF_ENUM_NAME_FontMatrix = _mupdf.PDF_ENUM_NAME_FontMatrix
(0)            PDF_ENUM_NAME_FontName = _mupdf.PDF_ENUM_NAME_FontName
(0)            PDF_ENUM_NAME_ForComment = _mupdf.PDF_ENUM_NAME_ForComment
(0)            PDF_ENUM_NAME_ForPublicRelease = _mupdf.PDF_ENUM_NAME_ForPublicRelease
(0)            PDF_ENUM_NAME_Form = _mupdf.PDF_ENUM_NAME_Form
(0)            PDF_ENUM_NAME_FormEx = _mupdf.PDF_ENUM_NAME_FormEx
(0)            PDF_ENUM_NAME_FormType = _mupdf.PDF_ENUM_NAME_FormType
(0)            PDF_ENUM_NAME_Formula = _mupdf.PDF_ENUM_NAME_Formula
(0)            PDF_ENUM_NAME_FreeText = _mupdf.PDF_ENUM_NAME_FreeText
(0)            PDF_ENUM_NAME_FreeTextCallout = _mupdf.PDF_ENUM_NAME_FreeTextCallout
(0)            PDF_ENUM_NAME_FreeTextTypeWriter = _mupdf.PDF_ENUM_NAME_FreeTextTypeWriter
(0)            PDF_ENUM_NAME_Function = _mupdf.PDF_ENUM_NAME_Function
(0)            PDF_ENUM_NAME_FunctionType = _mupdf.PDF_ENUM_NAME_FunctionType
(0)            PDF_ENUM_NAME_Functions = _mupdf.PDF_ENUM_NAME_Functions
(0)            PDF_ENUM_NAME_G = _mupdf.PDF_ENUM_NAME_G
(0)            PDF_ENUM_NAME_GTS_PDFX = _mupdf.PDF_ENUM_NAME_GTS_PDFX
(0)            PDF_ENUM_NAME_Gamma = _mupdf.PDF_ENUM_NAME_Gamma
(0)            PDF_ENUM_NAME_Glitter = _mupdf.PDF_ENUM_NAME_Glitter
(0)            PDF_ENUM_NAME_GoTo = _mupdf.PDF_ENUM_NAME_GoTo
(0)            PDF_ENUM_NAME_GoToR = _mupdf.PDF_ENUM_NAME_GoToR
(0)            PDF_ENUM_NAME_Group = _mupdf.PDF_ENUM_NAME_Group
(0)            PDF_ENUM_NAME_H = _mupdf.PDF_ENUM_NAME_H
(0)            PDF_ENUM_NAME_H1 = _mupdf.PDF_ENUM_NAME_H1
(0)            PDF_ENUM_NAME_H2 = _mupdf.PDF_ENUM_NAME_H2
(0)            PDF_ENUM_NAME_H3 = _mupdf.PDF_ENUM_NAME_H3
(0)            PDF_ENUM_NAME_H4 = _mupdf.PDF_ENUM_NAME_H4
(0)            PDF_ENUM_NAME_H5 = _mupdf.PDF_ENUM_NAME_H5
(0)            PDF_ENUM_NAME_H6 = _mupdf.PDF_ENUM_NAME_H6
(0)            PDF_ENUM_NAME_Height = _mupdf.PDF_ENUM_NAME_Height
(0)            PDF_ENUM_NAME_Helv = _mupdf.PDF_ENUM_NAME_Helv
(0)            PDF_ENUM_NAME_Highlight = _mupdf.PDF_ENUM_NAME_Highlight
(0)            PDF_ENUM_NAME_HistoryPos = _mupdf.PDF_ENUM_NAME_HistoryPos
(0)            PDF_ENUM_NAME_I = _mupdf.PDF_ENUM_NAME_I
(0)            PDF_ENUM_NAME_IC = _mupdf.PDF_ENUM_NAME_IC
(0)            PDF_ENUM_NAME_ICCBased = _mupdf.PDF_ENUM_NAME_ICCBased
(0)            PDF_ENUM_NAME_ID = _mupdf.PDF_ENUM_NAME_ID
(0)            PDF_ENUM_NAME_IM = _mupdf.PDF_ENUM_NAME_IM
(0)            PDF_ENUM_NAME_IRT = _mupdf.PDF_ENUM_NAME_IRT
(0)            PDF_ENUM_NAME_IT = _mupdf.PDF_ENUM_NAME_IT
(0)            PDF_ENUM_NAME_Identity = _mupdf.PDF_ENUM_NAME_Identity
(0)            PDF_ENUM_NAME_Identity_H = _mupdf.PDF_ENUM_NAME_Identity_H
(0)            PDF_ENUM_NAME_Identity_V = _mupdf.PDF_ENUM_NAME_Identity_V
(0)            PDF_ENUM_NAME_Image = _mupdf.PDF_ENUM_NAME_Image
(0)            PDF_ENUM_NAME_ImageB = _mupdf.PDF_ENUM_NAME_ImageB
(0)            PDF_ENUM_NAME_ImageC = _mupdf.PDF_ENUM_NAME_ImageC
(0)            PDF_ENUM_NAME_ImageI = _mupdf.PDF_ENUM_NAME_ImageI
(0)            PDF_ENUM_NAME_ImageMask = _mupdf.PDF_ENUM_NAME_ImageMask
(0)            PDF_ENUM_NAME_Include = _mupdf.PDF_ENUM_NAME_Include
(0)            PDF_ENUM_NAME_Index = _mupdf.PDF_ENUM_NAME_Index
(0)            PDF_ENUM_NAME_Indexed = _mupdf.PDF_ENUM_NAME_Indexed
```

```
(0)            PDF_ENUM_NAME_Info = _mupdf.PDF_ENUM_NAME_Info
(0)            PDF_ENUM_NAME_Ink = _mupdf.PDF_ENUM_NAME_Ink
(0)            PDF_ENUM_NAME_InkList = _mupdf.PDF_ENUM_NAME_InkList
(0)            PDF_ENUM_NAME_Intent = _mupdf.PDF_ENUM_NAME_Intent
(0)            PDF_ENUM_NAME_Interpolate = _mupdf.PDF_ENUM_NAME_Interpolate
(0)            PDF_ENUM_NAME_IsMap = _mupdf.PDF_ENUM_NAME_IsMap
(0)            PDF_ENUM_NAME_ItalicAngle = _mupdf.PDF_ENUM_NAME_ItalicAngle
(0)            PDF_ENUM_NAME_JBIG2Decode = _mupdf.PDF_ENUM_NAME_JBIG2Decode
(0)            PDF_ENUM_NAME_JBIG2Globals = _mupdf.PDF_ENUM_NAME_JBIG2Globals
(0)            PDF_ENUM_NAME_JPXDecode = _mupdf.PDF_ENUM_NAME_JPXDecode
(0)            PDF_ENUM_NAME_JS = _mupdf.PDF_ENUM_NAME_JS
(0)            PDF_ENUM_NAME_JavaScript = _mupdf.PDF_ENUM_NAME_JavaScript
(0)            PDF_ENUM_NAME_K = _mupdf.PDF_ENUM_NAME_K
(0)            PDF_ENUM_NAME_Keywords = _mupdf.PDF_ENUM_NAME_Keywords
(0)            PDF_ENUM_NAME_Kids = _mupdf.PDF_ENUM_NAME_Kids
(0)            PDF_ENUM_NAME_L = _mupdf.PDF_ENUM_NAME_L
(0)            PDF_ENUM_NAME_LBody = _mupdf.PDF_ENUM_NAME_LBody
(0)            PDF_ENUM_NAME_LC = _mupdf.PDF_ENUM_NAME_LC
(0)            PDF_ENUM_NAME_LE = _mupdf.PDF_ENUM_NAME_LE
(0)            PDF_ENUM_NAME_LI = _mupdf.PDF_ENUM_NAME_LI
(0)            PDF_ENUM_NAME_LJ = _mupdf.PDF_ENUM_NAME_LJ
(0)            PDF_ENUM_NAME_LW = _mupdf.PDF_ENUM_NAME_LW
(0)            PDF_ENUM_NAME_LZ = _mupdf.PDF_ENUM_NAME_LZ
(0)            PDF_ENUM_NAME_LZW = _mupdf.PDF_ENUM_NAME_LZW
(0)            PDF_ENUM_NAME_LZWDecode = _mupdf.PDF_ENUM_NAME_LZWDecode
(0)            PDF_ENUM_NAME_Lab = _mupdf.PDF_ENUM_NAME_Lab
(0)            PDF_ENUM_NAME_Label = _mupdf.PDF_ENUM_NAME_Label
(0)            PDF_ENUM_NAME_Lang = _mupdf.PDF_ENUM_NAME_Lang
(0)            PDF_ENUM_NAME_Last = _mupdf.PDF_ENUM_NAME_Last
(0)            PDF_ENUM_NAME_LastChar = _mupdf.PDF_ENUM_NAME_LastChar
(0)            PDF_ENUM_NAME_LastPage = _mupdf.PDF_ENUM_NAME_LastPage
(0)            PDF_ENUM_NAME_Launch = _mupdf.PDF_ENUM_NAME_Launch
(0)            PDF_ENUM_NAME_Layer = _mupdf.PDF_ENUM_NAME_Layer
(0)            PDF_ENUM_NAME_Lbl = _mupdf.PDF_ENUM_NAME_Lbl
(0)            PDF_ENUM_NAME_Length = _mupdf.PDF_ENUM_NAME_Length
(0)            PDF_ENUM_NAME_Length1 = _mupdf.PDF_ENUM_NAME_Length1
(0)            PDF_ENUM_NAME_Length2 = _mupdf.PDF_ENUM_NAME_Length2
(0)            PDF_ENUM_NAME_Length3 = _mupdf.PDF_ENUM_NAME_Length3
(0)            PDF_ENUM_NAME_Limits = _mupdf.PDF_ENUM_NAME_Limits
(0)            PDF_ENUM_NAME_Line = _mupdf.PDF_ENUM_NAME_Line
(0)            PDF_ENUM_NAME_LineArrow = _mupdf.PDF_ENUM_NAME_LineArrow
(0)            PDF_ENUM_NAME_LineDimension = _mupdf.PDF_ENUM_NAME_LineDimension
(0)            PDF_ENUM_NAME_Linearized = _mupdf.PDF_ENUM_NAME_Linearized
(0)            PDF_ENUM_NAME_Link = _mupdf.PDF_ENUM_NAME_Link
(0)            PDF_ENUM_NAME_List = _mupdf.PDF_ENUM_NAME_List
(0)            PDF_ENUM_NAME_Location = _mupdf.PDF_ENUM_NAME_Location
(0)            PDF_ENUM_NAME_Lock = _mupdf.PDF_ENUM_NAME_Lock
(0)            PDF_ENUM_NAME_Locked = _mupdf.PDF_ENUM_NAME_Locked
(0)            PDF_ENUM_NAME_Luminosity = _mupdf.PDF_ENUM_NAME_Luminosity
(0)            PDF_ENUM_NAME_M = _mupdf.PDF_ENUM_NAME_M
(0)            PDF_ENUM_NAME_MCID = _mupdf.PDF_ENUM_NAME_MCID
(0)            PDF_ENUM_NAME_MK = _mupdf.PDF_ENUM_NAME_MK
(0)            PDF_ENUM_NAME_ML = _mupdf.PDF_ENUM_NAME_ML
(0)            PDF_ENUM_NAME_MMType1 = _mupdf.PDF_ENUM_NAME_MMType1
(0)            PDF_ENUM_NAME_Mac = _mupdf.PDF_ENUM_NAME_Mac
(0)            PDF_ENUM_NAME_Mask = _mupdf.PDF_ENUM_NAME_Mask
(0)            PDF_ENUM_NAME_Matrix = _mupdf.PDF_ENUM_NAME_Matrix
(0)            PDF_ENUM_NAME_Matte = _mupdf.PDF_ENUM_NAME_Matte
(0)            PDF_ENUM_NAME_MaxLen = _mupdf.PDF_ENUM_NAME_MaxLen
(0)            PDF_ENUM_NAME_MediaBox = _mupdf.PDF_ENUM_NAME_MediaBox
(0)            PDF_ENUM_NAME_Metadata = _mupdf.PDF_ENUM_NAME_Metadata
(0)            PDF_ENUM_NAME_MissingWidth = _mupdf.PDF_ENUM_NAME_MissingWidth
(0)            PDF_ENUM_NAME_ModDate = _mupdf.PDF_ENUM_NAME_ModDate
(0)            PDF_ENUM_NAME_Movie = _mupdf.PDF_ENUM_NAME_Movie
(0)            PDF_ENUM_NAME_Msg = _mupdf.PDF_ENUM_NAME_Msg
(0)            PDF_ENUM_NAME_Multiply = _mupdf.PDF_ENUM_NAME_Multiply
(0)            PDF_ENUM_NAME_N = _mupdf.PDF_ENUM_NAME_N
(0)            PDF_ENUM_NAME_Name = _mupdf.PDF_ENUM_NAME_Name
(0)            PDF_ENUM_NAME_Named = _mupdf.PDF_ENUM_NAME_Named
```

```
(0)            PDF_ENUM_NAME_Names = _mupdf.PDF_ENUM_NAME_Names
(0)            PDF_ENUM_NAME_NewWindow = _mupdf.PDF_ENUM_NAME_NewWindow
(0)            PDF_ENUM_NAME_Next = _mupdf.PDF_ENUM_NAME_Next
(0)            PDF_ENUM_NAME_NextPage = _mupdf.PDF_ENUM_NAME_NextPage
(0)            PDF_ENUM_NAME_NonEFontNoWarn = _mupdf.PDF_ENUM_NAME_NonEFontNoWarn
(0)            PDF_ENUM_NAME_NonStruct = _mupdf.PDF_ENUM_NAME_NonStruct
(0)            PDF_ENUM_NAME_None = _mupdf.PDF_ENUM_NAME_None
(0)            PDF_ENUM_NAME_Normal = _mupdf.PDF_ENUM_NAME_Normal
(0)            PDF_ENUM_NAME_NotApproved = _mupdf.PDF_ENUM_NAME_NotApproved
(0)            PDF_ENUM_NAME_NotForPublicRelease = _mupdf.PDF_ENUM_NAME_NotForPublicRelease
(0)            PDF_ENUM_NAME_Note = _mupdf.PDF_ENUM_NAME_Note
(0)            PDF_ENUM_NAME_NumSections = _mupdf.PDF_ENUM_NAME_NumSections
(0)            PDF_ENUM_NAME_Nums = _mupdf.PDF_ENUM_NAME_Nums
(0)            PDF_ENUM_NAME_O = _mupdf.PDF_ENUM_NAME_O
(0)            PDF_ENUM_NAME_OC = _mupdf.PDF_ENUM_NAME_OC
(0)            PDF_ENUM_NAME_OCG = _mupdf.PDF_ENUM_NAME_OCG
(0)            PDF_ENUM_NAME_OCGs = _mupdf.PDF_ENUM_NAME_OCGs
(0)            PDF_ENUM_NAME_OCMD = _mupdf.PDF_ENUM_NAME_OCMD
(0)            PDF_ENUM_NAME_OCProperties = _mupdf.PDF_ENUM_NAME_OCProperties
(0)            PDF_ENUM_NAME_OE = _mupdf.PDF_ENUM_NAME_OE
(0)            PDF_ENUM_NAME_OFF = _mupdf.PDF_ENUM_NAME_OFF
(0)            PDF_ENUM_NAME_ON = _mupdf.PDF_ENUM_NAME_ON
(0)            PDF_ENUM_NAME_OP = _mupdf.PDF_ENUM_NAME_OP
(0)            PDF_ENUM_NAME_OPM = _mupdf.PDF_ENUM_NAME_OPM
(0)            PDF_ENUM_NAME_OS = _mupdf.PDF_ENUM_NAME_OS
(0)            PDF_ENUM_NAME_ObjStm = _mupdf.PDF_ENUM_NAME_ObjStm
(0)            PDF_ENUM_NAME_Of = _mupdf.PDF_ENUM_NAME_Of
(0)            PDF_ENUM_NAME_Off = _mupdf.PDF_ENUM_NAME_Off
(0)            PDF_ENUM_NAME_Open = _mupdf.PDF_ENUM_NAME_Open
(0)            PDF_ENUM_NAME_OpenArrow = _mupdf.PDF_ENUM_NAME_OpenArrow
(0)            PDF_ENUM_NAME_OpenType = _mupdf.PDF_ENUM_NAME_OpenType
(0)            PDF_ENUM_NAME_Opt = _mupdf.PDF_ENUM_NAME_Opt
(0)            PDF_ENUM_NAME_Order = _mupdf.PDF_ENUM_NAME_Order
(0)            PDF_ENUM_NAME_Ordering = _mupdf.PDF_ENUM_NAME_Ordering
(0)            PDF_ENUM_NAME_Outlines = _mupdf.PDF_ENUM_NAME_Outlines
(0)            PDF_ENUM_NAME_OutputCondition = _mupdf.PDF_ENUM_NAME_OutputCondition
(0)            PDF_ENUM_NAME_OutputConditionIdentifier =
_mupdf.PDF_ENUM_NAME_OutputConditionIdentifier
(0)            PDF_ENUM_NAME_OutputIntent = _mupdf.PDF_ENUM_NAME_OutputIntent
(0)            PDF_ENUM_NAME_OutputIntents = _mupdf.PDF_ENUM_NAME_OutputIntents
(0)            PDF_ENUM_NAME_P = _mupdf.PDF_ENUM_NAME_P
(0)            PDF_ENUM_NAME_PDF = _mupdf.PDF_ENUM_NAME_PDF
(0)            PDF_ENUM_NAME_PS = _mupdf.PDF_ENUM_NAME_PS
(0)            PDF_ENUM_NAME_Page = _mupdf.PDF_ENUM_NAME_Page
(0)            PDF_ENUM_NAME_PageLabels = _mupdf.PDF_ENUM_NAME_PageLabels
(0)            PDF_ENUM_NAME_PageMode = _mupdf.PDF_ENUM_NAME_PageMode
(0)            PDF_ENUM_NAME_Pages = _mupdf.PDF_ENUM_NAME_Pages
(0)            PDF_ENUM_NAME_PaintType = _mupdf.PDF_ENUM_NAME_PaintType
(0)            PDF_ENUM_NAME_Params = _mupdf.PDF_ENUM_NAME_Params
(0)            PDF_ENUM_NAME_Parent = _mupdf.PDF_ENUM_NAME_Parent
(0)            PDF_ENUM_NAME_ParentTree = _mupdf.PDF_ENUM_NAME_ParentTree
(0)            PDF_ENUM_NAME_Part = _mupdf.PDF_ENUM_NAME_Part
(0)            PDF_ENUM_NAME_Pattern = _mupdf.PDF_ENUM_NAME_Pattern
(0)            PDF_ENUM_NAME_PatternType = _mupdf.PDF_ENUM_NAME_PatternType
(0)            PDF_ENUM_NAME_Perms = _mupdf.PDF_ENUM_NAME_Perms
(0)            PDF_ENUM_NAME_PolyLine = _mupdf.PDF_ENUM_NAME_PolyLine
(0)            PDF_ENUM_NAME_PolyLineDimension = _mupdf.PDF_ENUM_NAME_PolyLineDimension
(0)            PDF_ENUM_NAME_Polygon = _mupdf.PDF_ENUM_NAME_Polygon
(0)            PDF_ENUM_NAME_PolygonCloud = _mupdf.PDF_ENUM_NAME_PolygonCloud
(0)            PDF_ENUM_NAME_PolygonDimension = _mupdf.PDF_ENUM_NAME_PolygonDimension
(0)            PDF_ENUM_NAME_Popup = _mupdf.PDF_ENUM_NAME_Popup
(0)            PDF_ENUM_NAME_PreRelease = _mupdf.PDF_ENUM_NAME_PreRelease
(0)            PDF_ENUM_NAME_Predictor = _mupdf.PDF_ENUM_NAME_Predictor
(0)            PDF_ENUM_NAME_Prev = _mupdf.PDF_ENUM_NAME_Prev
(0)            PDF_ENUM_NAME_PrevPage = _mupdf.PDF_ENUM_NAME_PrevPage
(0)            PDF_ENUM_NAME_Preview = _mupdf.PDF_ENUM_NAME_Preview
(0)            PDF_ENUM_NAME_Print = _mupdf.PDF_ENUM_NAME_Print
(0)            PDF_ENUM_NAME_PrinterMark = _mupdf.PDF_ENUM_NAME_PrinterMark
(0)            PDF_ENUM_NAME_Private = _mupdf.PDF_ENUM_NAME_Private
```

```
(0)            PDF_ENUM_NAME_ProcSet = _mupdf.PDF_ENUM_NAME_ProcSet
(0)            PDF_ENUM_NAME_Producer = _mupdf.PDF_ENUM_NAME_Producer
(0)            PDF_ENUM_NAME_Prop_AuthTime = _mupdf.PDF_ENUM_NAME_Prop_AuthTime
(0)            PDF_ENUM_NAME_Prop_AuthType = _mupdf.PDF_ENUM_NAME_Prop_AuthType
(0)            PDF_ENUM_NAME_Prop_Build = _mupdf.PDF_ENUM_NAME_Prop_Build
(0)            PDF_ENUM_NAME_Properties = _mupdf.PDF_ENUM_NAME_Properties
(0)            PDF_ENUM_NAME_PubSec = _mupdf.PDF_ENUM_NAME_PubSec
(0)            PDF_ENUM_NAME_Push = _mupdf.PDF_ENUM_NAME_Push
(0)            PDF_ENUM_NAME_Q = _mupdf.PDF_ENUM_NAME_Q
(0)            PDF_ENUM_NAME_QuadPoints = _mupdf.PDF_ENUM_NAME_QuadPoints
(0)            PDF_ENUM_NAME_Quote = _mupdf.PDF_ENUM_NAME_Quote
(0)            PDF_ENUM_NAME_R = _mupdf.PDF_ENUM_NAME_R
(0)            PDF_ENUM_NAME_RB = _mupdf.PDF_ENUM_NAME_RB
(0)            PDF_ENUM_NAME_RBGroups = _mupdf.PDF_ENUM_NAME_RBGroups
(0)            PDF_ENUM_NAME_RC = _mupdf.PDF_ENUM_NAME_RC
(0)            PDF_ENUM_NAME_RClosedArrow = _mupdf.PDF_ENUM_NAME_RClosedArrow
(0)            PDF_ENUM_NAME_RD = _mupdf.PDF_ENUM_NAME_RD
(0)            PDF_ENUM_NAME_REx = _mupdf.PDF_ENUM_NAME_REx
(0)            PDF_ENUM_NAME_RGB = _mupdf.PDF_ENUM_NAME_RGB
(0)            PDF_ENUM_NAME_RI = _mupdf.PDF_ENUM_NAME_RI
(0)            PDF_ENUM_NAME_RL = _mupdf.PDF_ENUM_NAME_RL
(0)            PDF_ENUM_NAME_ROpenArrow = _mupdf.PDF_ENUM_NAME_ROpenArrow
(0)            PDF_ENUM_NAME_RP = _mupdf.PDF_ENUM_NAME_RP
(0)            PDF_ENUM_NAME_RT = _mupdf.PDF_ENUM_NAME_RT
(0)            PDF_ENUM_NAME_Range = _mupdf.PDF_ENUM_NAME_Range
(0)            PDF_ENUM_NAME_Reason = _mupdf.PDF_ENUM_NAME_Reason
(0)            PDF_ENUM_NAME_Rect = _mupdf.PDF_ENUM_NAME_Rect
(0)            PDF_ENUM_NAME_Redact = _mupdf.PDF_ENUM_NAME_Redact
(0)            PDF_ENUM_NAME_Ref = _mupdf.PDF_ENUM_NAME_Ref
(0)            PDF_ENUM_NAME_Reference = _mupdf.PDF_ENUM_NAME_Reference
(0)            PDF_ENUM_NAME_Registry = _mupdf.PDF_ENUM_NAME_Registry
(0)            PDF_ENUM_NAME_ResetForm = _mupdf.PDF_ENUM_NAME_ResetForm
(0)            PDF_ENUM_NAME_Resources = _mupdf.PDF_ENUM_NAME_Resources
(0)            PDF_ENUM_NAME_RoleMap = _mupdf.PDF_ENUM_NAME_RoleMap
(0)            PDF_ENUM_NAME_Root = _mupdf.PDF_ENUM_NAME_Root
(0)            PDF_ENUM_NAME_Rotate = _mupdf.PDF_ENUM_NAME_Rotate
(0)            PDF_ENUM_NAME_Rows = _mupdf.PDF_ENUM_NAME_Rows
(0)            PDF_ENUM_NAME_Ruby = _mupdf.PDF_ENUM_NAME_Ruby
(0)            PDF_ENUM_NAME_RunLengthDecode = _mupdf.PDF_ENUM_NAME_RunLengthDecode
(0)            PDF_ENUM_NAME_S = _mupdf.PDF_ENUM_NAME_S
(0)            PDF_ENUM_NAME_SMask = _mupdf.PDF_ENUM_NAME_SMask
(0)            PDF_ENUM_NAME_SMaskInData = _mupdf.PDF_ENUM_NAME_SMaskInData
(0)            PDF_ENUM_NAME_Schema = _mupdf.PDF_ENUM_NAME_Schema
(0)            PDF_ENUM_NAME_Screen = _mupdf.PDF_ENUM_NAME_Screen
(0)            PDF_ENUM_NAME_Sect = _mupdf.PDF_ENUM_NAME_Sect
(0)            PDF_ENUM_NAME_Separation = _mupdf.PDF_ENUM_NAME_Separation
(0)            PDF_ENUM_NAME_Shading = _mupdf.PDF_ENUM_NAME_Shading
(0)            PDF_ENUM_NAME_ShadingType = _mupdf.PDF_ENUM_NAME_ShadingType
(0)            PDF_ENUM_NAME_Si = _mupdf.PDF_ENUM_NAME_Si
(0)            PDF_ENUM_NAME_Sig = _mupdf.PDF_ENUM_NAME_Sig
(0)            PDF_ENUM_NAME_SigFlags = _mupdf.PDF_ENUM_NAME_SigFlags
(0)            PDF_ENUM_NAME_SigQ = _mupdf.PDF_ENUM_NAME_SigQ
(0)            PDF_ENUM_NAME_SigRef = _mupdf.PDF_ENUM_NAME_SigRef
(0)            PDF_ENUM_NAME_Size = _mupdf.PDF_ENUM_NAME_Size
(0)            PDF_ENUM_NAME_Slash = _mupdf.PDF_ENUM_NAME_Slash
(0)            PDF_ENUM_NAME_Sold = _mupdf.PDF_ENUM_NAME_Sold
(0)            PDF_ENUM_NAME_Sound = _mupdf.PDF_ENUM_NAME_Sound
(0)            PDF_ENUM_NAME_Span = _mupdf.PDF_ENUM_NAME_Span
(0)            PDF_ENUM_NAME_Split = _mupdf.PDF_ENUM_NAME_Split
(0)            PDF_ENUM_NAME_Square = _mupdf.PDF_ENUM_NAME_Square
(0)            PDF_ENUM_NAME_Squiggly = _mupdf.PDF_ENUM_NAME_Squiggly
(0)            PDF_ENUM_NAME_St = _mupdf.PDF_ENUM_NAME_St
(0)            PDF_ENUM_NAME_Stamp = _mupdf.PDF_ENUM_NAME_Stamp
(0)            PDF_ENUM_NAME_StampImage = _mupdf.PDF_ENUM_NAME_StampImage
(0)            PDF_ENUM_NAME_StampSnapshot = _mupdf.PDF_ENUM_NAME_StampSnapshot
(0)            PDF_ENUM_NAME_Standard = _mupdf.PDF_ENUM_NAME_Standard
(0)            PDF_ENUM_NAME_StdCF = _mupdf.PDF_ENUM_NAME_StdCF
(0)            PDF_ENUM_NAME_StemV = _mupdf.PDF_ENUM_NAME_StemV
(0)            PDF_ENUM_NAME_StmF = _mupdf.PDF_ENUM_NAME_StmF
```

```
(0)            PDF_ENUM_NAME_StrF = _mupdf.PDF_ENUM_NAME_StrF
(0)            PDF_ENUM_NAME_StrikeOut = _mupdf.PDF_ENUM_NAME_StrikeOut
(0)            PDF_ENUM_NAME_Strong = _mupdf.PDF_ENUM_NAME_Strong
(0)            PDF_ENUM_NAME_StructParent = _mupdf.PDF_ENUM_NAME_StructParent
(0)            PDF_ENUM_NAME_StructParents = _mupdf.PDF_ENUM_NAME_StructParents
(0)            PDF_ENUM_NAME_StructTreeRoot = _mupdf.PDF_ENUM_NAME_StructTreeRoot
(0)            PDF_ENUM_NAME_Sub = _mupdf.PDF_ENUM_NAME_Sub
(0)            PDF_ENUM_NAME_SubFilter = _mupdf.PDF_ENUM_NAME_SubFilter
(0)            PDF_ENUM_NAME_Subject = _mupdf.PDF_ENUM_NAME_Subject
(0)            PDF_ENUM_NAME_Subtype = _mupdf.PDF_ENUM_NAME_Subtype
(0)            PDF_ENUM_NAME_Subtype2 = _mupdf.PDF_ENUM_NAME_Subtype2
(0)            PDF_ENUM_NAME_Supplement = _mupdf.PDF_ENUM_NAME_Supplement
(0)            PDF_ENUM_NAME_Symb = _mupdf.PDF_ENUM_NAME_Symb
(0)            PDF_ENUM_NAME_T = _mupdf.PDF_ENUM_NAME_T
(0)            PDF_ENUM_NAME_TBody = _mupdf.PDF_ENUM_NAME_TBody
(0)            PDF_ENUM_NAME_TD = _mupdf.PDF_ENUM_NAME_TD
(0)            PDF_ENUM_NAME_TFoot = _mupdf.PDF_ENUM_NAME_TFoot
(0)            PDF_ENUM_NAME_TH = _mupdf.PDF_ENUM_NAME_TH
(0)            PDF_ENUM_NAME_THead = _mupdf.PDF_ENUM_NAME_THead
(0)            PDF_ENUM_NAME_TI = _mupdf.PDF_ENUM_NAME_TI
(0)            PDF_ENUM_NAME_TOC = _mupdf.PDF_ENUM_NAME_TOC
(0)            PDF_ENUM_NAME_TOCI = _mupdf.PDF_ENUM_NAME_TOCI
(0)            PDF_ENUM_NAME_TR = _mupdf.PDF_ENUM_NAME_TR
(0)            PDF_ENUM_NAME_TR2 = _mupdf.PDF_ENUM_NAME_TR2
(0)            PDF_ENUM_NAME_TU = _mupdf.PDF_ENUM_NAME_TU
(0)            PDF_ENUM_NAME_Table = _mupdf.PDF_ENUM_NAME_Table
(0)            PDF_ENUM_NAME_Text = _mupdf.PDF_ENUM_NAME_Text
(0)            PDF_ENUM_NAME_TilingType = _mupdf.PDF_ENUM_NAME_TilingType
(0)            PDF_ENUM_NAME_Times = _mupdf.PDF_ENUM_NAME_Times
(0)            PDF_ENUM_NAME_Title = _mupdf.PDF_ENUM_NAME_Title
(0)            PDF_ENUM_NAME_ToUnicode = _mupdf.PDF_ENUM_NAME_ToUnicode
(0)            PDF_ENUM_NAME_TopSecret = _mupdf.PDF_ENUM_NAME_TopSecret
(0)            PDF_ENUM_NAME_Trans = _mupdf.PDF_ENUM_NAME_Trans
(0)            PDF_ENUM_NAME_TransformMethod = _mupdf.PDF_ENUM_NAME_TransformMethod
(0)            PDF_ENUM_NAME_TransformParams = _mupdf.PDF_ENUM_NAME_TransformParams
(0)            PDF_ENUM_NAME_Transparency = _mupdf.PDF_ENUM_NAME_Transparency
(0)            PDF_ENUM_NAME_TrapNet = _mupdf.PDF_ENUM_NAME_TrapNet
(0)            PDF_ENUM_NAME_TrimBox = _mupdf.PDF_ENUM_NAME_TrimBox
(0)            PDF_ENUM_NAME_TrueType = _mupdf.PDF_ENUM_NAME_TrueType
(0)            PDF_ENUM_NAME_TrustedMode = _mupdf.PDF_ENUM_NAME_TrustedMode
(0)            PDF_ENUM_NAME_Tx = _mupdf.PDF_ENUM_NAME_Tx
(0)            PDF_ENUM_NAME_Type = _mupdf.PDF_ENUM_NAME_Type
(0)            PDF_ENUM_NAME_Type0 = _mupdf.PDF_ENUM_NAME_Type0
(0)            PDF_ENUM_NAME_Type1 = _mupdf.PDF_ENUM_NAME_Type1
(0)            PDF_ENUM_NAME_Type1C = _mupdf.PDF_ENUM_NAME_Type1C
(0)            PDF_ENUM_NAME_Type3 = _mupdf.PDF_ENUM_NAME_Type3
(0)            PDF_ENUM_NAME_U = _mupdf.PDF_ENUM_NAME_U
(0)            PDF_ENUM_NAME_UE = _mupdf.PDF_ENUM_NAME_UE
(0)            PDF_ENUM_NAME_UF = _mupdf.PDF_ENUM_NAME_UF
(0)            PDF_ENUM_NAME_URI = _mupdf.PDF_ENUM_NAME_URI
(0)            PDF_ENUM_NAME_URL = _mupdf.PDF_ENUM_NAME_URL
(0)            PDF_ENUM_NAME_Unchanged = _mupdf.PDF_ENUM_NAME_Unchanged
(0)            PDF_ENUM_NAME_Uncover = _mupdf.PDF_ENUM_NAME_Uncover
(0)            PDF_ENUM_NAME_Underline = _mupdf.PDF_ENUM_NAME_Underline
(0)            PDF_ENUM_NAME_Unix = _mupdf.PDF_ENUM_NAME_Unix
(0)            PDF_ENUM_NAME_Usage = _mupdf.PDF_ENUM_NAME_Usage
(0)            PDF_ENUM_NAME_UseBlackPtComp = _mupdf.PDF_ENUM_NAME_UseBlackPtComp
(0)            PDF_ENUM_NAME_UseCMap = _mupdf.PDF_ENUM_NAME_UseCMap
(0)            PDF_ENUM_NAME_UseOutlines = _mupdf.PDF_ENUM_NAME_UseOutlines
(0)            PDF_ENUM_NAME_UserUnit = _mupdf.PDF_ENUM_NAME_UserUnit
(0)            PDF_ENUM_NAME_V = _mupdf.PDF_ENUM_NAME_V
(0)            PDF_ENUM_NAME_V2 = _mupdf.PDF_ENUM_NAME_V2
(0)            PDF_ENUM_NAME_VE = _mupdf.PDF_ENUM_NAME_VE
(0)            PDF_ENUM_NAME_Version = _mupdf.PDF_ENUM_NAME_Version
(0)            PDF_ENUM_NAME_Vertices = _mupdf.PDF_ENUM_NAME_Vertices
(0)            PDF_ENUM_NAME_VerticesPerRow = _mupdf.PDF_ENUM_NAME_VerticesPerRow
(0)            PDF_ENUM_NAME_View = _mupdf.PDF_ENUM_NAME_View
(0)            PDF_ENUM_NAME_W = _mupdf.PDF_ENUM_NAME_W
(0)            PDF_ENUM_NAME_W2 = _mupdf.PDF_ENUM_NAME_W2
```

```
3239: (0)           PDF_ENUM_NAME_WMode = _mupdf.PDF_ENUM_NAME_WMode
3240: (0)           PDF_ENUM_NAME_WP = _mupdf.PDF_ENUM_NAME_WP
3241: (0)           PDF_ENUM_NAME_WT = _mupdf.PDF_ENUM_NAME_WT
3242: (0)           PDF_ENUM_NAME_Warichu = _mupdf.PDF_ENUM_NAME_Warichu
3243: (0)           PDF_ENUM_NAME_Watermark = _mupdf.PDF_ENUM_NAME_Watermark
3244: (0)           PDF_ENUM_NAME_WhitePoint = _mupdf.PDF_ENUM_NAME_WhitePoint
3245: (0)           PDF_ENUM_NAME_Widget = _mupdf.PDF_ENUM_NAME_Widget
3246: (0)           PDF_ENUM_NAME_Width = _mupdf.PDF_ENUM_NAME_Width
3247: (0)           PDF_ENUM_NAME_Widths = _mupdf.PDF_ENUM_NAME_Widths
3248: (0)           PDF_ENUM_NAME_WinAnsiEncoding = _mupdf.PDF_ENUM_NAME_WinAnsiEncoding
3249: (0)           PDF_ENUM_NAME_Wipe = _mupdf.PDF_ENUM_NAME_Wipe
3250: (0)           PDF_ENUM_NAME_XFA = _mupdf.PDF_ENUM_NAME_XFA
3251: (0)           PDF_ENUM_NAME_XHeight = _mupdf.PDF_ENUM_NAME_XHeight
3252: (0)           PDF_ENUM_NAME_XML = _mupdf.PDF_ENUM_NAME_XML
3253: (0)           PDF_ENUM_NAME_XObject = _mupdf.PDF_ENUM_NAME_XObject
3254: (0)           PDF_ENUM_NAME_XRef = _mupdf.PDF_ENUM_NAME_XRef
3255: (0)           PDF_ENUM_NAME_XRefStm = _mupdf.PDF_ENUM_NAME_XRefStm
3256: (0)           PDF_ENUM_NAME_XStep = _mupdf.PDF_ENUM_NAME_XStep
3257: (0)           PDF_ENUM_NAME_XYZ = _mupdf.PDF_ENUM_NAME_XYZ
3258: (0)           PDF_ENUM_NAME_YStep = _mupdf.PDF_ENUM_NAME_YStep
3259: (0)           PDF_ENUM_NAME_Yes = _mupdf.PDF_ENUM_NAME_Yes
3260: (0)           PDF_ENUM_NAME_ZaDb = _mupdf.PDF_ENUM_NAME_ZaDb
3261: (0)           PDF_ENUM_NAME_a = _mupdf.PDF_ENUM_NAME_a
3262: (0)           PDF_ENUM_NAME_adbe_pkcs7_detached = _mupdf.PDF_ENUM_NAME_adbe_pkcs7_detached
3263: (0)           PDF_ENUM_NAME_ca = _mupdf.PDF_ENUM_NAME_ca
3264: (0)           PDF_ENUM_NAME_n0 = _mupdf.PDF_ENUM_NAME_n0
3265: (0)           PDF_ENUM_NAME_n1 = _mupdf.PDF_ENUM_NAME_n1
3266: (0)           PDF_ENUM_NAME_n2 = _mupdf.PDF_ENUM_NAME_n2
3267: (0)           PDF_ENUM_NAME_op = _mupdf.PDF_ENUM_NAME_op
3268: (0)           PDF_ENUM_NAME_r = _mupdf.PDF_ENUM_NAME_r
3269: (0)           PDF_ENUM_LIMIT = _mupdf.PDF_ENUM_LIMIT
3270: (0)           PDF_LEXBUF_SMALL = _mupdf.PDF_LEXBUF_SMALL
3271: (0)           PDF_LEXBUF_LARGE = _mupdf.PDF_LEXBUF_LARGE
3272: (0)           class pdf_lexbuf(object):
3273: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3274: (4)               __repr__ = _swig_repr
3275: (4)               size = property(_mupdf.pdf_lexbuf_size_get, _mupdf.pdf_lexbuf_size_set)
3276: (4)               base_size = property(_mupdf.pdf_lexbuf_base_size_get,
_mupdf.pdf_lexbuf_base_size_set)
3277: (4)               len = property(_mupdf.pdf_lexbuf_len_get, _mupdf.pdf_lexbuf_len_set)
3278: (4)               i = property(_mupdf.pdf_lexbuf_i_get, _mupdf.pdf_lexbuf_i_set)
3279: (4)               f = property(_mupdf.pdf_lexbuf_f_get, _mupdf.pdf_lexbuf_f_set)
3280: (4)               scratch = property(_mupdf.pdf_lexbuf_scratch_get,
_mupdf.pdf_lexbuf_scratch_set)
3281: (4)               buffer = property(_mupdf.pdf_lexbuf_buffer_get,
_mupdf.pdf_lexbuf_buffer_set)
3282: (4)               def __init__(self):
3283: (8)                   _mupdf.pdf_lexbuf_swiginit(self, _mupdf.new_pdf_lexbuf())
3284: (4)               __swig_destroy__ = _mupdf.delete_pdf_lexbuf
3285: (0)           _mupdf.pdf_lexbuf_swigregister(pdf_lexbuf)
3286: (0)           class pdf_lexbuf_large(object):
3287: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3288: (4)               __repr__ = _swig_repr
3289: (4)               base = property(_mupdf.pdf_lexbuf_large_base_get,
_mupdf.pdf_lexbuf_large_base_set)
3290: (4)               buffer = property(_mupdf.pdf_lexbuf_large_buffer_get,
_mupdf.pdf_lexbuf_large_buffer_set)
3291: (4)               def __init__(self):
3292: (8)                   _mupdf.pdf_lexbuf_large_swiginit(self, _mupdf.new_pdf_lexbuf_large())
3293: (4)               __swig_destroy__ = _mupdf.delete_pdf_lexbuf_large
3294: (0)           _mupdf.pdf_lexbuf_large_swigregister(pdf_lexbuf_large)
3295: (0)           class pdf_js_console(object):
3296: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3297: (4)               __repr__ = _swig_repr
3298: (4)               drop = property(_mupdf.pdf_js_console_drop_get,
_mupdf.pdf_js_console_drop_set)
```

```
3299: (4)               show = property(_mupdf.pdf_js_console_show_get,
_mupdf.pdf_js_console_show_set)
3300: (4)               hide = property(_mupdf.pdf_js_console_hide_get,
_mupdf.pdf_js_console_hide_set)
3301: (4)               clear = property(_mupdf.pdf_js_console_clear_get,
_mupdf.pdf_js_console_clear_set)
3302: (4)               write = property(_mupdf.pdf_js_console_write_get,
_mupdf.pdf_js_console_write_set)
3303: (4)               def __init__(self):
3304: (8)                   _mupdf.pdf_js_console_swiginit(self, _mupdf.new_pdf_js_console())
3305: (4)               __swig_destroy__ = _mupdf.delete_pdf_js_console
3306: (0)           _mupdf.pdf_js_console_swigregister(pdf_js_console)
3307: (0)           class pdf_layer_config(object):
3308: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3309: (4)               __repr__ = _swig_repr
3310: (4)               name = property(_mupdf.pdf_layer_config_name_get,
_mupdf.pdf_layer_config_name_set)
3311: (4)               creator = property(_mupdf.pdf_layer_config_creator_get,
_mupdf.pdf_layer_config_creator_set)
3312: (4)               def __init__(self):
3313: (8)                   _mupdf.pdf_layer_config_swiginit(self, _mupdf.new_pdf_layer_config())
3314: (4)               __swig_destroy__ = _mupdf.delete_pdf_layer_config
3315: (0)           _mupdf.pdf_layer_config_swigregister(pdf_layer_config)
3316: (0)           PDF_LAYER_UI_LABEL = _mupdf.PDF_LAYER_UI_LABEL
3317: (0)           PDF_LAYER_UI_CHECKBOX = _mupdf.PDF_LAYER_UI_CHECKBOX
3318: (0)           PDF_LAYER_UI_RADIOBOX = _mupdf.PDF_LAYER_UI_RADIOBOX
3319: (0)           class pdf_layer_config_ui(object):
3320: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3321: (4)               __repr__ = _swig_repr
3322: (4)               text = property(_mupdf.pdf_layer_config_ui_text_get,
_mupdf.pdf_layer_config_ui_text_set)
3323: (4)               depth = property(_mupdf.pdf_layer_config_ui_depth_get,
_mupdf.pdf_layer_config_ui_depth_set)
3324: (4)               type = property(_mupdf.pdf_layer_config_ui_type_get,
_mupdf.pdf_layer_config_ui_type_set)
3325: (4)               selected = property(_mupdf.pdf_layer_config_ui_selected_get,
_mupdf.pdf_layer_config_ui_selected_set)
3326: (4)               locked = property(_mupdf.pdf_layer_config_ui_locked_get,
_mupdf.pdf_layer_config_ui_locked_set)
3327: (4)               def __init__(self):
3328: (8)                   _mupdf.pdf_layer_config_ui_swiginit(self,
_mupdf.new_pdf_layer_config_ui())
3329: (4)               __swig_destroy__ = _mupdf.delete_pdf_layer_config_ui
3330: (0)           _mupdf.pdf_layer_config_ui_swigregister(pdf_layer_config_ui)
3331: (0)           class pdf_unsaved_sig(object):
3332: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3333: (4)               __repr__ = _swig_repr
3334: (4)               field = property(_mupdf.pdf_unsaved_sig_field_get,
_mupdf.pdf_unsaved_sig_field_set)
3335: (4)               byte_range_start = property(_mupdf.pdf_unsaved_sig_byte_range_start_get,
_mupdf.pdf_unsaved_sig_byte_range_start_set)
3336: (4)               byte_range_end = property(_mupdf.pdf_unsaved_sig_byte_range_end_get,
_mupdf.pdf_unsaved_sig_byte_range_end_set)
3337: (4)               contents_start = property(_mupdf.pdf_unsaved_sig_contents_start_get,
_mupdf.pdf_unsaved_sig_contents_start_set)
3338: (4)               contents_end = property(_mupdf.pdf_unsaved_sig_contents_end_get,
_mupdf.pdf_unsaved_sig_contents_end_set)
3339: (4)               signer = property(_mupdf.pdf_unsaved_sig_signer_get,
_mupdf.pdf_unsaved_sig_signer_set)
3340: (4)               next = property(_mupdf.pdf_unsaved_sig_next_get,
_mupdf.pdf_unsaved_sig_next_set)
3341: (4)               def __init__(self):
3342: (8)                   _mupdf.pdf_unsaved_sig_swiginit(self, _mupdf.new_pdf_unsaved_sig())
3343: (4)               __swig_destroy__ = _mupdf.delete_pdf_unsaved_sig
3344: (0)           _mupdf.pdf_unsaved_sig_swigregister(pdf_unsaved_sig)
3345: (0)           class pdf_rev_page_map(object):
```

```
3346: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3347: (4)               __repr__ = _swig_repr
3348: (4)               page = property(_mupdf.pdf_rev_page_map_page_get,
_mupdf.pdf_rev_page_map_page_set)
3349: (4)               object = property(_mupdf.pdf_rev_page_map_object_get,
_mupdf.pdf_rev_page_map_object_set)
3350: (4)               def __init__(self):
3351: (8)                   _mupdf.pdf_rev_page_map_swiginit(self, _mupdf.new_pdf_rev_page_map())
3352: (4)               __swig_destroy__ = _mupdf.delete_pdf_rev_page_map
3353: (0)           _mupdf.pdf_rev_page_map_swigregister(pdf_rev_page_map)
3354: (0)           class pdf_hint_page(object):
3355: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3356: (4)               __repr__ = _swig_repr
3357: (4)               number = property(_mupdf.pdf_hint_page_number_get,
_mupdf.pdf_hint_page_number_set)
3358: (4)               offset = property(_mupdf.pdf_hint_page_offset_get,
_mupdf.pdf_hint_page_offset_set)
3359: (4)               index = property(_mupdf.pdf_hint_page_index_get,
_mupdf.pdf_hint_page_index_set)
3360: (4)               def __init__(self):
3361: (8)                   _mupdf.pdf_hint_page_swiginit(self, _mupdf.new_pdf_hint_page())
3362: (4)               __swig_destroy__ = _mupdf.delete_pdf_hint_page
3363: (0)           _mupdf.pdf_hint_page_swigregister(pdf_hint_page)
3364: (0)           class pdf_hint_shared(object):
3365: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3366: (4)               __repr__ = _swig_repr
3367: (4)               number = property(_mupdf.pdf_hint_shared_number_get,
_mupdf.pdf_hint_shared_number_set)
3368: (4)               offset = property(_mupdf.pdf_hint_shared_offset_get,
_mupdf.pdf_hint_shared_offset_set)
3369: (4)               def __init__(self):
3370: (8)                   _mupdf.pdf_hint_shared_swiginit(self, _mupdf.new_pdf_hint_shared())
3371: (4)               __swig_destroy__ = _mupdf.delete_pdf_hint_shared
3372: (0)           _mupdf.pdf_hint_shared_swigregister(pdf_hint_shared)
3373: (0)           class pdf_document(object):
3374: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3375: (4)               __repr__ = _swig_repr
3376: (4)               super = property(_mupdf.pdf_document_super_get,
_mupdf.pdf_document_super_set)
3377: (4)               file = property(_mupdf.pdf_document_file_get,
_mupdf.pdf_document_file_set)
3378: (4)               version = property(_mupdf.pdf_document_version_get,
_mupdf.pdf_document_version_set)
3379: (4)               is_fdf = property(_mupdf.pdf_document_is_fdf_get,
_mupdf.pdf_document_is_fdf_set)
3380: (4)               startxref = property(_mupdf.pdf_document_startxref_get,
_mupdf.pdf_document_startxref_set)
3381: (4)               file_size = property(_mupdf.pdf_document_file_size_get,
_mupdf.pdf_document_file_size_set)
3382: (4)               crypt = property(_mupdf.pdf_document_crypt_get,
_mupdf.pdf_document_crypt_set)
3383: (4)               ocg = property(_mupdf.pdf_document_ocg_get, _mupdf.pdf_document_ocg_set)
3384: (4)               oi = property(_mupdf.pdf_document_oi_get, _mupdf.pdf_document_oi_set)
3385: (4)               max_xref_len = property(_mupdf.pdf_document_max_xref_len_get,
_mupdf.pdf_document_max_xref_len_set)
3386: (4)               num_xref_sections = property(_mupdf.pdf_document_num_xref_sections_get,
_mupdf.pdf_document_num_xref_sections_set)
3387: (4)               saved_num_xref_sections =
property(_mupdf.pdf_document_saved_num_xref_sections_get,
_mupdf.pdf_document_saved_num_xref_sections_set)
3388: (4)               num_incremental_sections =
property(_mupdf.pdf_document_num_incremental_sections_get,
_mupdf.pdf_document_num_incremental_sections_set)
3389: (4)               xref_base = property(_mupdf.pdf_document_xref_base_get,
_mupdf.pdf_document_xref_base_set)
```

```
3390: (4)               disallow_new_increments =
property(_mupdf.pdf_document_disallow_new_increments_get,
_mupdf.pdf_document_disallow_new_increments_set)
3391: (4)               local_xref = property(_mupdf.pdf_document_local_xref_get,
_mupdf.pdf_document_local_xref_set)
3392: (4)               local_xref_nesting = property(_mupdf.pdf_document_local_xref_nesting_get,
_mupdf.pdf_document_local_xref_nesting_set)
3393: (4)               xref_sections = property(_mupdf.pdf_document_xref_sections_get,
_mupdf.pdf_document_xref_sections_set)
3394: (4)               saved_xref_sections =
property(_mupdf.pdf_document_saved_xref_sections_get, _mupdf.pdf_document_saved_xref_sections_set)
3395: (4)               xref_index = property(_mupdf.pdf_document_xref_index_get,
_mupdf.pdf_document_xref_index_set)
3396: (4)               save_in_progress = property(_mupdf.pdf_document_save_in_progress_get,
_mupdf.pdf_document_save_in_progress_set)
3397: (4)               last_xref_was_old_style =
property(_mupdf.pdf_document_last_xref_was_old_style_get,
_mupdf.pdf_document_last_xref_was_old_style_set)
3398: (4)               has_linearization_object =
property(_mupdf.pdf_document_has_linearization_object_get,
_mupdf.pdf_document_has_linearization_object_set)
3399: (4)               map_page_count = property(_mupdf.pdf_document_map_page_count_get,
_mupdf.pdf_document_map_page_count_set)
3400: (4)               rev_page_map = property(_mupdf.pdf_document_rev_page_map_get,
_mupdf.pdf_document_rev_page_map_set)
3401: (4)               fwd_page_map = property(_mupdf.pdf_document_fwd_page_map_get,
_mupdf.pdf_document_fwd_page_map_set)
3402: (4)               page_tree_broken = property(_mupdf.pdf_document_page_tree_broken_get,
_mupdf.pdf_document_page_tree_broken_set)
3403: (4)               repair_attempted = property(_mupdf.pdf_document_repair_attempted_get,
_mupdf.pdf_document_repair_attempted_set)
3404: (4)               repair_in_progress = property(_mupdf.pdf_document_repair_in_progress_get,
_mupdf.pdf_document_repair_in_progress_set)
3405: (4)               non_structural_change =
property(_mupdf.pdf_document_non_structural_change_get,
_mupdf.pdf_document_non_structural_change_set)
3406: (4)               file_reading_linearly =
property(_mupdf.pdf_document_file_reading_linearly_get,
_mupdf.pdf_document_file_reading_linearly_set)
3407: (4)               file_length = property(_mupdf.pdf_document_file_length_get,
_mupdf.pdf_document_file_length_set)
3408: (4)               linear_page_count = property(_mupdf.pdf_document_linear_page_count_get,
_mupdf.pdf_document_linear_page_count_set)
3409: (4)               linear_obj = property(_mupdf.pdf_document_linear_obj_get,
_mupdf.pdf_document_linear_obj_set)
3410: (4)               linear_page_refs = property(_mupdf.pdf_document_linear_page_refs_get,
_mupdf.pdf_document_linear_page_refs_set)
3411: (4)               linear_page1_obj_num =
property(_mupdf.pdf_document_linear_page1_obj_num_get,
_mupdf.pdf_document_linear_page1_obj_num_set)
3412: (4)               linear_pos = property(_mupdf.pdf_document_linear_pos_get,
_mupdf.pdf_document_linear_pos_set)
3413: (4)               linear_page_num = property(_mupdf.pdf_document_linear_page_num_get,
_mupdf.pdf_document_linear_page_num_set)
3414: (4)               hint_object_offset = property(_mupdf.pdf_document_hint_object_offset_get,
_mupdf.pdf_document_hint_object_offset_set)
3415: (4)               hint_object_length = property(_mupdf.pdf_document_hint_object_length_get,
_mupdf.pdf_document_hint_object_length_set)
3416: (4)               hints_loaded = property(_mupdf.pdf_document_hints_loaded_get,
_mupdf.pdf_document_hints_loaded_set)
3417: (4)               hint_page = property(_mupdf.pdf_document_hint_page_get,
_mupdf.pdf_document_hint_page_set)
3418: (4)               hint_shared_ref = property(_mupdf.pdf_document_hint_shared_ref_get,
_mupdf.pdf_document_hint_shared_ref_set)
3419: (4)               hint_shared = property(_mupdf.pdf_document_hint_shared_get,
_mupdf.pdf_document_hint_shared_set)
3420: (4)               hint_obj_offsets_max =
property(_mupdf.pdf_document_hint_obj_offsets_max_get,
_mupdf.pdf_document_hint_obj_offsets_max_set)
```

```
3421: (4)               hint_obj_offsets = property(_mupdf.pdf_document_hint_obj_offsets_get,
_mupdf.pdf_document_hint_obj_offsets_set)
3422: (4)               resources_localised =
property(_mupdf.pdf_document_resources_localised_get, _mupdf.pdf_document_resources_localised_set)
3423: (4)               lexbuf = property(_mupdf.pdf_document_lexbuf_get,
_mupdf.pdf_document_lexbuf_set)
3424: (4)               js = property(_mupdf.pdf_document_js_get, _mupdf.pdf_document_js_set)
3425: (4)               recalculate = property(_mupdf.pdf_document_recalculate_get,
_mupdf.pdf_document_recalculate_set)
3426: (4)               redacted = property(_mupdf.pdf_document_redacted_get,
_mupdf.pdf_document_redacted_set)
3427: (4)               resynth_required = property(_mupdf.pdf_document_resynth_required_get,
_mupdf.pdf_document_resynth_required_set)
3428: (4)               event_cb = property(_mupdf.pdf_document_event_cb_get,
_mupdf.pdf_document_event_cb_set)
3429: (4)               free_event_data_cb = property(_mupdf.pdf_document_free_event_data_cb_get,
_mupdf.pdf_document_free_event_data_cb_set)
3430: (4)               event_cb_data = property(_mupdf.pdf_document_event_cb_data_get,
_mupdf.pdf_document_event_cb_data_set)
3431: (4)               num_type3_fonts = property(_mupdf.pdf_document_num_type3_fonts_get,
_mupdf.pdf_document_num_type3_fonts_set)
3432: (4)               max_type3_fonts = property(_mupdf.pdf_document_max_type3_fonts_get,
_mupdf.pdf_document_max_type3_fonts_set)
3433: (4)               type3_fonts = property(_mupdf.pdf_document_type3_fonts_get,
_mupdf.pdf_document_type3_fonts_set)
3434: (4)               orphans_max = property(_mupdf.pdf_document_orphans_max_get,
_mupdf.pdf_document_orphans_max_set)
3435: (4)               orphans_count = property(_mupdf.pdf_document_orphans_count_get,
_mupdf.pdf_document_orphans_count_set)
3436: (4)               orphans = property(_mupdf.pdf_document_orphans_get,
_mupdf.pdf_document_orphans_set)
3437: (4)               xfa = property(_mupdf.pdf_document_xfa_get, _mupdf.pdf_document_xfa_set)
3438: (4)               journal = property(_mupdf.pdf_document_journal_get,
_mupdf.pdf_document_journal_set)
3439: (4)               def __init__(self):
3440: (8)                   _mupdf.pdf_document_swiginit(self, _mupdf.new_pdf_document())
3441: (4)               __swig_destroy__ = _mupdf.delete_pdf_document
3442: (0)           _mupdf.pdf_document_swigregister(pdf_document)
3443: (0)           PDF_PAGE_LABEL_NONE = _mupdf.PDF_PAGE_LABEL_NONE
3444: (0)           PDF_PAGE_LABEL_DECIMAL = _mupdf.PDF_PAGE_LABEL_DECIMAL
3445: (0)           PDF_PAGE_LABEL_ROMAN_UC = _mupdf.PDF_PAGE_LABEL_ROMAN_UC
3446: (0)           PDF_PAGE_LABEL_ROMAN_LC = _mupdf.PDF_PAGE_LABEL_ROMAN_LC
3447: (0)           PDF_PAGE_LABEL_ALPHA_UC = _mupdf.PDF_PAGE_LABEL_ALPHA_UC
3448: (0)           PDF_PAGE_LABEL_ALPHA_LC = _mupdf.PDF_PAGE_LABEL_ALPHA_LC
3449: (0)           class pdf_write_options(object):
3450: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3451: (4)               __repr__ = _swig_repr
3452: (4)               do_incremental = property(_mupdf.pdf_write_options_do_incremental_get,
_mupdf.pdf_write_options_do_incremental_set)
3453: (4)               do_pretty = property(_mupdf.pdf_write_options_do_pretty_get,
_mupdf.pdf_write_options_do_pretty_set)
3454: (4)               do_ascii = property(_mupdf.pdf_write_options_do_ascii_get,
_mupdf.pdf_write_options_do_ascii_set)
3455: (4)               do_compress = property(_mupdf.pdf_write_options_do_compress_get,
_mupdf.pdf_write_options_do_compress_set)
3456: (4)               do_compress_images =
property(_mupdf.pdf_write_options_do_compress_images_get,
_mupdf.pdf_write_options_do_compress_images_set)
3457: (4)               do_compress_fonts =
property(_mupdf.pdf_write_options_do_compress_fonts_get,
_mupdf.pdf_write_options_do_compress_fonts_set)
3458: (4)               do_decompress = property(_mupdf.pdf_write_options_do_decompress_get,
_mupdf.pdf_write_options_do_decompress_set)
3459: (4)               do_garbage = property(_mupdf.pdf_write_options_do_garbage_get,
_mupdf.pdf_write_options_do_garbage_set)
3460: (4)               do_linear = property(_mupdf.pdf_write_options_do_linear_get,
_mupdf.pdf_write_options_do_linear_set)
3461: (4)               do_clean = property(_mupdf.pdf_write_options_do_clean_get,
```

```
_mupdf.pdf_write_options_do_clean_set)
3462: (4)               do_sanitize = property(_mupdf.pdf_write_options_do_sanitize_get,
_mupdf.pdf_write_options_do_sanitize_set)
3463: (4)               do_appearance = property(_mupdf.pdf_write_options_do_appearance_get,
_mupdf.pdf_write_options_do_appearance_set)
3464: (4)               do_encrypt = property(_mupdf.pdf_write_options_do_encrypt_get,
_mupdf.pdf_write_options_do_encrypt_set)
3465: (4)               dont_regenerate_id =
property(_mupdf.pdf_write_options_dont_regenerate_id_get,
_mupdf.pdf_write_options_dont_regenerate_id_set)
3466: (4)               permissions = property(_mupdf.pdf_write_options_permissions_get,
_mupdf.pdf_write_options_permissions_set)
3467: (4)               opwd_utf8 = property(_mupdf.pdf_write_options_opwd_utf8_get,
_mupdf.pdf_write_options_opwd_utf8_set)
3468: (4)               upwd_utf8 = property(_mupdf.pdf_write_options_upwd_utf8_get,
_mupdf.pdf_write_options_upwd_utf8_set)
3469: (4)               do_snapshot = property(_mupdf.pdf_write_options_do_snapshot_get,
_mupdf.pdf_write_options_do_snapshot_set)
3470: (4)               do_preserve_metadata =
property(_mupdf.pdf_write_options_do_preserve_metadata_get,
_mupdf.pdf_write_options_do_preserve_metadata_set)
3471: (4)               do_use_objstms = property(_mupdf.pdf_write_options_do_use_objstms_get,
_mupdf.pdf_write_options_do_use_objstms_set)
3472: (4)               compression_effort =
property(_mupdf.pdf_write_options_compression_effort_get,
_mupdf.pdf_write_options_compression_effort_set)
3473: (4)               def __init__(self):
3474: (8)                   _mupdf.pdf_write_options_swiginit(self,
_mupdf.new_pdf_write_options())
3475: (4)               __swig_destroy__ = _mupdf.delete_pdf_write_options
3476: (0)           _mupdf.pdf_write_options_swigregister(pdf_write_options)
3477: (0)           PDF_TOK_ERROR = _mupdf.PDF_TOK_ERROR
3478: (0)           PDF_TOK_EOF = _mupdf.PDF_TOK_EOF
3479: (0)           PDF_TOK_OPEN_ARRAY = _mupdf.PDF_TOK_OPEN_ARRAY
3480: (0)           PDF_TOK_CLOSE_ARRAY = _mupdf.PDF_TOK_CLOSE_ARRAY
3481: (0)           PDF_TOK_OPEN_DICT = _mupdf.PDF_TOK_OPEN_DICT
3482: (0)           PDF_TOK_CLOSE_DICT = _mupdf.PDF_TOK_CLOSE_DICT
3483: (0)           PDF_TOK_OPEN_BRACE = _mupdf.PDF_TOK_OPEN_BRACE
3484: (0)           PDF_TOK_CLOSE_BRACE = _mupdf.PDF_TOK_CLOSE_BRACE
3485: (0)           PDF_TOK_NAME = _mupdf.PDF_TOK_NAME
3486: (0)           PDF_TOK_INT = _mupdf.PDF_TOK_INT
3487: (0)           PDF_TOK_REAL = _mupdf.PDF_TOK_REAL
3488: (0)           PDF_TOK_STRING = _mupdf.PDF_TOK_STRING
3489: (0)           PDF_TOK_KEYWORD = _mupdf.PDF_TOK_KEYWORD
3490: (0)           PDF_TOK_R = _mupdf.PDF_TOK_R
3491: (0)           PDF_TOK_TRUE = _mupdf.PDF_TOK_TRUE
3492: (0)           PDF_TOK_FALSE = _mupdf.PDF_TOK_FALSE
3493: (0)           PDF_TOK_NULL = _mupdf.PDF_TOK_NULL
3494: (0)           PDF_TOK_OBJ = _mupdf.PDF_TOK_OBJ
3495: (0)           PDF_TOK_ENDOBJ = _mupdf.PDF_TOK_ENDOBJ
3496: (0)           PDF_TOK_STREAM = _mupdf.PDF_TOK_STREAM
3497: (0)           PDF_TOK_ENDSTREAM = _mupdf.PDF_TOK_ENDSTREAM
3498: (0)           PDF_TOK_XREF = _mupdf.PDF_TOK_XREF
3499: (0)           PDF_TOK_TRAILER = _mupdf.PDF_TOK_TRAILER
3500: (0)           PDF_TOK_STARTXREF = _mupdf.PDF_TOK_STARTXREF
3501: (0)           PDF_TOK_NEWOBJ = _mupdf.PDF_TOK_NEWOBJ
3502: (0)           PDF_NUM_TOKENS = _mupdf.PDF_NUM_TOKENS
3503: (0)           class pdf_xref_entry(object):
3504: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3505: (4)               __repr__ = _swig_repr
3506: (4)               type = property(_mupdf.pdf_xref_entry_type_get,
_mupdf.pdf_xref_entry_type_set)
3507: (4)               marked = property(_mupdf.pdf_xref_entry_marked_get,
_mupdf.pdf_xref_entry_marked_set)
3508: (4)               gen = property(_mupdf.pdf_xref_entry_gen_get,
_mupdf.pdf_xref_entry_gen_set)
3509: (4)               num = property(_mupdf.pdf_xref_entry_num_get,
_mupdf.pdf_xref_entry_num_set)
```

```
3510: (4)               ofs = property(_mupdf.pdf_xref_entry_ofs_get,
_mupdf.pdf_xref_entry_ofs_set)
3511: (4)               stm_ofs = property(_mupdf.pdf_xref_entry_stm_ofs_get,
_mupdf.pdf_xref_entry_stm_ofs_set)
3512: (4)               stm_buf = property(_mupdf.pdf_xref_entry_stm_buf_get,
_mupdf.pdf_xref_entry_stm_buf_set)
3513: (4)               obj = property(_mupdf.pdf_xref_entry_obj_get,
_mupdf.pdf_xref_entry_obj_set)
3514: (4)               def __init__(self):
3515: (8)                   _mupdf.pdf_xref_entry_swiginit(self, _mupdf.new_pdf_xref_entry())
3516: (4)               __swig_destroy__ = _mupdf.delete_pdf_xref_entry
3517: (0)           _mupdf.pdf_xref_entry_swigregister(pdf_xref_entry)
3518: (0)           pdf_default_write_options = cvar.pdf_default_write_options
3519: (0)           class pdf_xref_subsec(object):
3520: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3521: (4)               __repr__ = _swig_repr
3522: (4)               next = property(_mupdf.pdf_xref_subsec_next_get,
_mupdf.pdf_xref_subsec_next_set)
3523: (4)               len = property(_mupdf.pdf_xref_subsec_len_get,
_mupdf.pdf_xref_subsec_len_set)
3524: (4)               start = property(_mupdf.pdf_xref_subsec_start_get,
_mupdf.pdf_xref_subsec_start_set)
3525: (4)               table = property(_mupdf.pdf_xref_subsec_table_get,
_mupdf.pdf_xref_subsec_table_set)
3526: (4)               def __init__(self):
3527: (8)                   _mupdf.pdf_xref_subsec_swiginit(self, _mupdf.new_pdf_xref_subsec())
3528: (4)               __swig_destroy__ = _mupdf.delete_pdf_xref_subsec
3529: (0)           _mupdf.pdf_xref_subsec_swigregister(pdf_xref_subsec)
3530: (0)           class pdf_xref(object):
3531: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3532: (4)               __repr__ = _swig_repr
3533: (4)               num_objects = property(_mupdf.pdf_xref_num_objects_get,
_mupdf.pdf_xref_num_objects_set)
3534: (4)               subsec = property(_mupdf.pdf_xref_subsec_get, _mupdf.pdf_xref_subsec_set)
3535: (4)               trailer = property(_mupdf.pdf_xref_trailer_get,
_mupdf.pdf_xref_trailer_set)
3536: (4)               pre_repair_trailer = property(_mupdf.pdf_xref_pre_repair_trailer_get,
_mupdf.pdf_xref_pre_repair_trailer_set)
3537: (4)               unsaved_sigs = property(_mupdf.pdf_xref_unsaved_sigs_get,
_mupdf.pdf_xref_unsaved_sigs_set)
3538: (4)               unsaved_sigs_end = property(_mupdf.pdf_xref_unsaved_sigs_end_get,
_mupdf.pdf_xref_unsaved_sigs_end_set)
3539: (4)               end_ofs = property(_mupdf.pdf_xref_end_ofs_get,
_mupdf.pdf_xref_end_ofs_set)
3540: (4)               def __init__(self):
3541: (8)                   _mupdf.pdf_xref_swiginit(self, _mupdf.new_pdf_xref())
3542: (4)               __swig_destroy__ = _mupdf.delete_pdf_xref
3543: (0)           _mupdf.pdf_xref_swigregister(pdf_xref)
3544: (0)           PDF_ENCRYPT_KEEP = _mupdf.PDF_ENCRYPT_KEEP
3545: (0)           PDF_ENCRYPT_NONE = _mupdf.PDF_ENCRYPT_NONE
3546: (0)           PDF_ENCRYPT_RC4_40 = _mupdf.PDF_ENCRYPT_RC4_40
3547: (0)           PDF_ENCRYPT_RC4_128 = _mupdf.PDF_ENCRYPT_RC4_128
3548: (0)           PDF_ENCRYPT_AES_128 = _mupdf.PDF_ENCRYPT_AES_128
3549: (0)           PDF_ENCRYPT_AES_256 = _mupdf.PDF_ENCRYPT_AES_256
3550: (0)           PDF_ENCRYPT_UNKNOWN = _mupdf.PDF_ENCRYPT_UNKNOWN
3551: (0)           PDF_PERM_PRINT = _mupdf.PDF_PERM_PRINT
3552: (0)           PDF_PERM_MODIFY = _mupdf.PDF_PERM_MODIFY
3553: (0)           PDF_PERM_COPY = _mupdf.PDF_PERM_COPY
3554: (0)           PDF_PERM_ANNOTATE = _mupdf.PDF_PERM_ANNOTATE
3555: (0)           PDF_PERM_FORM = _mupdf.PDF_PERM_FORM
3556: (0)           PDF_PERM_ACCESSIBILITY = _mupdf.PDF_PERM_ACCESSIBILITY
3557: (0)           PDF_PERM_ASSEMBLE = _mupdf.PDF_PERM_ASSEMBLE
3558: (0)           PDF_PERM_PRINT_HQ = _mupdf.PDF_PERM_PRINT_HQ
3559: (0)           PDF_MRANGE_CAP = _mupdf.PDF_MRANGE_CAP
3560: (0)           class pdf_range(object):
3561: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
3562: (4)                __repr__ = _swig_repr
3563: (4)                low = property(_mupdf.pdf_range_low_get, _mupdf.pdf_range_low_set)
3564: (4)                high = property(_mupdf.pdf_range_high_get, _mupdf.pdf_range_high_set)
3565: (4)                out = property(_mupdf.pdf_range_out_get, _mupdf.pdf_range_out_set)
3566: (4)                def __init__(self):
3567: (8)                    _mupdf.pdf_range_swiginit(self, _mupdf.new_pdf_range())
3568: (4)                __swig_destroy__ = _mupdf.delete_pdf_range
3569: (0)            _mupdf.pdf_range_swigregister(pdf_range)
3570: (0)            class pdf_xrange(object):
3571: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3572: (4)                __repr__ = _swig_repr
3573: (4)                low = property(_mupdf.pdf_xrange_low_get, _mupdf.pdf_xrange_low_set)
3574: (4)                high = property(_mupdf.pdf_xrange_high_get, _mupdf.pdf_xrange_high_set)
3575: (4)                out = property(_mupdf.pdf_xrange_out_get, _mupdf.pdf_xrange_out_set)
3576: (4)                def __init__(self):
3577: (8)                    _mupdf.pdf_xrange_swiginit(self, _mupdf.new_pdf_xrange())
3578: (4)                __swig_destroy__ = _mupdf.delete_pdf_xrange
3579: (0)            _mupdf.pdf_xrange_swigregister(pdf_xrange)
3580: (0)            class pdf_mrange(object):
3581: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3582: (4)                __repr__ = _swig_repr
3583: (4)                low = property(_mupdf.pdf_mrange_low_get, _mupdf.pdf_mrange_low_set)
3584: (4)                out = property(_mupdf.pdf_mrange_out_get, _mupdf.pdf_mrange_out_set)
3585: (4)                def __init__(self):
3586: (8)                    _mupdf.pdf_mrange_swiginit(self, _mupdf.new_pdf_mrange())
3587: (4)                __swig_destroy__ = _mupdf.delete_pdf_mrange
3588: (0)            _mupdf.pdf_mrange_swigregister(pdf_mrange)
3589: (0)            class pdf_cmap(object):
3590: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3591: (4)                __repr__ = _swig_repr
3592: (4)                storable = property(_mupdf.pdf_cmap_storable_get,
_mupdf.pdf_cmap_storable_set)
3593: (4)                cmap_name = property(_mupdf.pdf_cmap_cmap_name_get,
_mupdf.pdf_cmap_cmap_name_set)
3594: (4)                usecmap_name = property(_mupdf.pdf_cmap_usecmap_name_get,
_mupdf.pdf_cmap_usecmap_name_set)
3595: (4)                usecmap = property(_mupdf.pdf_cmap_usecmap_get,
_mupdf.pdf_cmap_usecmap_set)
3596: (4)                wmode = property(_mupdf.pdf_cmap_wmode_get, _mupdf.pdf_cmap_wmode_set)
3597: (4)                codespace_len = property(_mupdf.pdf_cmap_codespace_len_get,
_mupdf.pdf_cmap_codespace_len_set)
3598: (4)                rlen = property(_mupdf.pdf_cmap_rlen_get, _mupdf.pdf_cmap_rlen_set)
3599: (4)                rcap = property(_mupdf.pdf_cmap_rcap_get, _mupdf.pdf_cmap_rcap_set)
3600: (4)                ranges = property(_mupdf.pdf_cmap_ranges_get, _mupdf.pdf_cmap_ranges_set)
3601: (4)                xlen = property(_mupdf.pdf_cmap_xlen_get, _mupdf.pdf_cmap_xlen_set)
3602: (4)                xcap = property(_mupdf.pdf_cmap_xcap_get, _mupdf.pdf_cmap_xcap_set)
3603: (4)                xranges = property(_mupdf.pdf_cmap_xranges_get,
_mupdf.pdf_cmap_xranges_set)
3604: (4)                mlen = property(_mupdf.pdf_cmap_mlen_get, _mupdf.pdf_cmap_mlen_set)
3605: (4)                mcap = property(_mupdf.pdf_cmap_mcap_get, _mupdf.pdf_cmap_mcap_set)
3606: (4)                mranges = property(_mupdf.pdf_cmap_mranges_get,
_mupdf.pdf_cmap_mranges_set)
3607: (4)                dlen = property(_mupdf.pdf_cmap_dlen_get, _mupdf.pdf_cmap_dlen_set)
3608: (4)                dcap = property(_mupdf.pdf_cmap_dcap_get, _mupdf.pdf_cmap_dcap_set)
3609: (4)                dict = property(_mupdf.pdf_cmap_dict_get, _mupdf.pdf_cmap_dict_set)
3610: (4)                tlen = property(_mupdf.pdf_cmap_tlen_get, _mupdf.pdf_cmap_tlen_set)
3611: (4)                tcap = property(_mupdf.pdf_cmap_tcap_get, _mupdf.pdf_cmap_tcap_set)
3612: (4)                ttop = property(_mupdf.pdf_cmap_ttop_get, _mupdf.pdf_cmap_ttop_set)
3613: (4)                tree = property(_mupdf.pdf_cmap_tree_get, _mupdf.pdf_cmap_tree_set)
3614: (4)                def __init__(self):
3615: (8)                    _mupdf.pdf_cmap_swiginit(self, _mupdf.new_pdf_cmap())
3616: (4)                __swig_destroy__ = _mupdf.delete_pdf_cmap
3617: (0)            _mupdf.pdf_cmap_swigregister(pdf_cmap)
3618: (0)            PDF_FD_FIXED_PITCH = _mupdf.PDF_FD_FIXED_PITCH
3619: (0)            PDF_FD_SERIF = _mupdf.PDF_FD_SERIF
3620: (0)            PDF_FD_SYMBOLIC = _mupdf.PDF_FD_SYMBOLIC
```

```
3621: (0)           PDF_FD_SCRIPT = _mupdf.PDF_FD_SCRIPT
3622: (0)           PDF_FD_NONSYMBOLIC = _mupdf.PDF_FD_NONSYMBOLIC
3623: (0)           PDF_FD_ITALIC = _mupdf.PDF_FD_ITALIC
3624: (0)           PDF_FD_ALL_CAP = _mupdf.PDF_FD_ALL_CAP
3625: (0)           PDF_FD_SMALL_CAP = _mupdf.PDF_FD_SMALL_CAP
3626: (0)           PDF_FD_FORCE_BOLD = _mupdf.PDF_FD_FORCE_BOLD
3627: (0)           class pdf_hmtx(object):
3628: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3629: (4)               __repr__ = _swig_repr
3630: (4)               lo = property(_mupdf.pdf_hmtx_lo_get, _mupdf.pdf_hmtx_lo_set)
3631: (4)               hi = property(_mupdf.pdf_hmtx_hi_get, _mupdf.pdf_hmtx_hi_set)
3632: (4)               w = property(_mupdf.pdf_hmtx_w_get, _mupdf.pdf_hmtx_w_set)
3633: (4)               def __init__(self):
3634: (8)                   _mupdf.pdf_hmtx_swiginit(self, _mupdf.new_pdf_hmtx())
3635: (4)               __swig_destroy__ = _mupdf.delete_pdf_hmtx
3636: (0)           _mupdf.pdf_hmtx_swigregister(pdf_hmtx)
3637: (0)           class pdf_vmtx(object):
3638: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3639: (4)               __repr__ = _swig_repr
3640: (4)               lo = property(_mupdf.pdf_vmtx_lo_get, _mupdf.pdf_vmtx_lo_set)
3641: (4)               hi = property(_mupdf.pdf_vmtx_hi_get, _mupdf.pdf_vmtx_hi_set)
3642: (4)               x = property(_mupdf.pdf_vmtx_x_get, _mupdf.pdf_vmtx_x_set)
3643: (4)               y = property(_mupdf.pdf_vmtx_y_get, _mupdf.pdf_vmtx_y_set)
3644: (4)               w = property(_mupdf.pdf_vmtx_w_get, _mupdf.pdf_vmtx_w_set)
3645: (4)               def __init__(self):
3646: (8)                   _mupdf.pdf_vmtx_swiginit(self, _mupdf.new_pdf_vmtx())
3647: (4)               __swig_destroy__ = _mupdf.delete_pdf_vmtx
3648: (0)           _mupdf.pdf_vmtx_swigregister(pdf_vmtx)
3649: (0)           class pdf_font_desc(object):
3650: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3651: (4)               __repr__ = _swig_repr
3652: (4)               storable = property(_mupdf.pdf_font_desc_storable_get,
_mupdf.pdf_font_desc_storable_set)
3653: (4)               size = property(_mupdf.pdf_font_desc_size_get,
_mupdf.pdf_font_desc_size_set)
3654: (4)               font = property(_mupdf.pdf_font_desc_font_get,
_mupdf.pdf_font_desc_font_set)
3655: (4)               flags = property(_mupdf.pdf_font_desc_flags_get,
_mupdf.pdf_font_desc_flags_set)
3656: (4)               italic_angle = property(_mupdf.pdf_font_desc_italic_angle_get,
_mupdf.pdf_font_desc_italic_angle_set)
3657: (4)               ascent = property(_mupdf.pdf_font_desc_ascent_get,
_mupdf.pdf_font_desc_ascent_set)
3658: (4)               descent = property(_mupdf.pdf_font_desc_descent_get,
_mupdf.pdf_font_desc_descent_set)
3659: (4)               cap_height = property(_mupdf.pdf_font_desc_cap_height_get,
_mupdf.pdf_font_desc_cap_height_set)
3660: (4)               x_height = property(_mupdf.pdf_font_desc_x_height_get,
_mupdf.pdf_font_desc_x_height_set)
3661: (4)               missing_width = property(_mupdf.pdf_font_desc_missing_width_get,
_mupdf.pdf_font_desc_missing_width_set)
3662: (4)               encoding = property(_mupdf.pdf_font_desc_encoding_get,
_mupdf.pdf_font_desc_encoding_set)
3663: (4)               to_ttf_cmap = property(_mupdf.pdf_font_desc_to_ttf_cmap_get,
_mupdf.pdf_font_desc_to_ttf_cmap_set)
3664: (4)               cid_to_gid_len = property(_mupdf.pdf_font_desc_cid_to_gid_len_get,
_mupdf.pdf_font_desc_cid_to_gid_len_set)
3665: (4)               cid_to_gid = property(_mupdf.pdf_font_desc_cid_to_gid_get,
_mupdf.pdf_font_desc_cid_to_gid_set)
3666: (4)               to_unicode = property(_mupdf.pdf_font_desc_to_unicode_get,
_mupdf.pdf_font_desc_to_unicode_set)
3667: (4)               cid_to_ucs_len = property(_mupdf.pdf_font_desc_cid_to_ucs_len_get,
_mupdf.pdf_font_desc_cid_to_ucs_len_set)
3668: (4)               cid_to_ucs = property(_mupdf.pdf_font_desc_cid_to_ucs_get,
_mupdf.pdf_font_desc_cid_to_ucs_set)
3669: (4)               wmode = property(_mupdf.pdf_font_desc_wmode_get,
```

```
_mupdf.pdf_font_desc_wmode_set)
3670: (4)               hmtx_len = property(_mupdf.pdf_font_desc_hmtx_len_get,
_mupdf.pdf_font_desc_hmtx_len_set)
3671: (4)               hmtx_cap = property(_mupdf.pdf_font_desc_hmtx_cap_get,
_mupdf.pdf_font_desc_hmtx_cap_set)
3672: (4)               dhmtx = property(_mupdf.pdf_font_desc_dhmtx_get,
_mupdf.pdf_font_desc_dhmtx_set)
3673: (4)               hmtx = property(_mupdf.pdf_font_desc_hmtx_get,
_mupdf.pdf_font_desc_hmtx_set)
3674: (4)               vmtx_len = property(_mupdf.pdf_font_desc_vmtx_len_get,
_mupdf.pdf_font_desc_vmtx_len_set)
3675: (4)               vmtx_cap = property(_mupdf.pdf_font_desc_vmtx_cap_get,
_mupdf.pdf_font_desc_vmtx_cap_set)
3676: (4)               dvmtx = property(_mupdf.pdf_font_desc_dvmtx_get,
_mupdf.pdf_font_desc_dvmtx_set)
3677: (4)               vmtx = property(_mupdf.pdf_font_desc_vmtx_get,
_mupdf.pdf_font_desc_vmtx_set)
3678: (4)               is_embedded = property(_mupdf.pdf_font_desc_is_embedded_get,
_mupdf.pdf_font_desc_is_embedded_set)
3679: (4)               t3loading = property(_mupdf.pdf_font_desc_t3loading_get,
_mupdf.pdf_font_desc_t3loading_set)
3680: (4)               def __init__(self):
3681: (8)                   _mupdf.pdf_font_desc_swiginit(self, _mupdf.new_pdf_font_desc())
3682: (4)               __swig_destroy__ = _mupdf.delete_pdf_font_desc
3683: (0)           _mupdf.pdf_font_desc_swigregister(pdf_font_desc)
3684: (0)           PDF_SIMPLE_FONT_RESOURCE = _mupdf.PDF_SIMPLE_FONT_RESOURCE
3685: (0)           PDF_CID_FONT_RESOURCE = _mupdf.PDF_CID_FONT_RESOURCE
3686: (0)           PDF_CJK_FONT_RESOURCE = _mupdf.PDF_CJK_FONT_RESOURCE
3687: (0)           PDF_SIMPLE_ENCODING_LATIN = _mupdf.PDF_SIMPLE_ENCODING_LATIN
3688: (0)           PDF_SIMPLE_ENCODING_GREEK = _mupdf.PDF_SIMPLE_ENCODING_GREEK
3689: (0)           PDF_SIMPLE_ENCODING_CYRILLIC = _mupdf.PDF_SIMPLE_ENCODING_CYRILLIC
3690: (0)           class pdf_font_resource_key(object):
3691: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3692: (4)               __repr__ = _swig_repr
3693: (4)               digest = property(_mupdf.pdf_font_resource_key_digest_get,
_mupdf.pdf_font_resource_key_digest_set)
3694: (4)               type = property(_mupdf.pdf_font_resource_key_type_get,
_mupdf.pdf_font_resource_key_type_set)
3695: (4)               encoding = property(_mupdf.pdf_font_resource_key_encoding_get,
_mupdf.pdf_font_resource_key_encoding_set)
3696: (4)               local_xref = property(_mupdf.pdf_font_resource_key_local_xref_get,
_mupdf.pdf_font_resource_key_local_xref_set)
3697: (4)               def __init__(self):
3698: (8)                   _mupdf.pdf_font_resource_key_swiginit(self,
_mupdf.new_pdf_font_resource_key())
3699: (4)               __swig_destroy__ = _mupdf.delete_pdf_font_resource_key
3700: (0)           _mupdf.pdf_font_resource_key_swigregister(pdf_font_resource_key)
3701: (0)           class pdf_pattern(object):
3702: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3703: (4)               __repr__ = _swig_repr
3704: (4)               storable = property(_mupdf.pdf_pattern_storable_get,
_mupdf.pdf_pattern_storable_set)
3705: (4)               ismask = property(_mupdf.pdf_pattern_ismask_get,
_mupdf.pdf_pattern_ismask_set)
3706: (4)               xstep = property(_mupdf.pdf_pattern_xstep_get,
_mupdf.pdf_pattern_xstep_set)
3707: (4)               ystep = property(_mupdf.pdf_pattern_ystep_get,
_mupdf.pdf_pattern_ystep_set)
3708: (4)               matrix = property(_mupdf.pdf_pattern_matrix_get,
_mupdf.pdf_pattern_matrix_set)
3709: (4)               bbox = property(_mupdf.pdf_pattern_bbox_get, _mupdf.pdf_pattern_bbox_set)
3710: (4)               document = property(_mupdf.pdf_pattern_document_get,
_mupdf.pdf_pattern_document_set)
3711: (4)               resources = property(_mupdf.pdf_pattern_resources_get,
_mupdf.pdf_pattern_resources_set)
3712: (4)               contents = property(_mupdf.pdf_pattern_contents_get,
_mupdf.pdf_pattern_contents_set)
```

```
3713: (4)               id = property(_mupdf.pdf_pattern_id_get, _mupdf.pdf_pattern_id_set)
3714: (4)               def __init__(self):
3715: (8)                   _mupdf.pdf_pattern_swiginit(self, _mupdf.new_pdf_pattern())
3716: (4)               __swig_destroy__ = _mupdf.delete_pdf_pattern
3717: (0)           _mupdf.pdf_pattern_swigregister(pdf_pattern)
3718: (0)           PDF_PROCESSOR_REQUIRES_DECODED_IMAGES =
_mupdf.PDF_PROCESSOR_REQUIRES_DECODED_IMAGES
3719: (0)           class pdf_processor(object):
3720: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3721: (4)               __repr__ = _swig_repr
3722: (4)               refs = property(_mupdf.pdf_processor_refs_get,
_mupdf.pdf_processor_refs_set)
3723: (4)               closed = property(_mupdf.pdf_processor_closed_get,
_mupdf.pdf_processor_closed_set)
3724: (4)               close_processor = property(_mupdf.pdf_processor_close_processor_get,
_mupdf.pdf_processor_close_processor_set)
3725: (4)               drop_processor = property(_mupdf.pdf_processor_drop_processor_get,
_mupdf.pdf_processor_drop_processor_set)
3726: (4)               reset_processor = property(_mupdf.pdf_processor_reset_processor_get,
_mupdf.pdf_processor_reset_processor_set)
3727: (4)               push_resources = property(_mupdf.pdf_processor_push_resources_get,
_mupdf.pdf_processor_push_resources_set)
3728: (4)               pop_resources = property(_mupdf.pdf_processor_pop_resources_get,
_mupdf.pdf_processor_pop_resources_set)
3729: (4)               op_w = property(_mupdf.pdf_processor_op_w_get,
_mupdf.pdf_processor_op_w_set)
3730: (4)               op_j = property(_mupdf.pdf_processor_op_j_get,
_mupdf.pdf_processor_op_j_set)
3731: (4)               op_J = property(_mupdf.pdf_processor_op_J_get,
_mupdf.pdf_processor_op_J_set)
3732: (4)               op_M = property(_mupdf.pdf_processor_op_M_get,
_mupdf.pdf_processor_op_M_set)
3733: (4)               op_d = property(_mupdf.pdf_processor_op_d_get,
_mupdf.pdf_processor_op_d_set)
3734: (4)               op_ri = property(_mupdf.pdf_processor_op_ri_get,
_mupdf.pdf_processor_op_ri_set)
3735: (4)               op_i = property(_mupdf.pdf_processor_op_i_get,
_mupdf.pdf_processor_op_i_set)
3736: (4)               op_gs_begin = property(_mupdf.pdf_processor_op_gs_begin_get,
_mupdf.pdf_processor_op_gs_begin_set)
3737: (4)               op_gs_BM = property(_mupdf.pdf_processor_op_gs_BM_get,
_mupdf.pdf_processor_op_gs_BM_set)
3738: (4)               op_gs_ca = property(_mupdf.pdf_processor_op_gs_ca_get,
_mupdf.pdf_processor_op_gs_ca_set)
3739: (4)               op_gs_CA = property(_mupdf.pdf_processor_op_gs_CA_get,
_mupdf.pdf_processor_op_gs_CA_set)
3740: (4)               op_gs_SMask = property(_mupdf.pdf_processor_op_gs_SMask_get,
_mupdf.pdf_processor_op_gs_SMask_set)
3741: (4)               op_gs_end = property(_mupdf.pdf_processor_op_gs_end_get,
_mupdf.pdf_processor_op_gs_end_set)
3742: (4)               op_q = property(_mupdf.pdf_processor_op_q_get,
_mupdf.pdf_processor_op_q_set)
3743: (4)               op_Q = property(_mupdf.pdf_processor_op_Q_get,
_mupdf.pdf_processor_op_Q_set)
3744: (4)               op_cm = property(_mupdf.pdf_processor_op_cm_get,
_mupdf.pdf_processor_op_cm_set)
3745: (4)               op_m = property(_mupdf.pdf_processor_op_m_get,
_mupdf.pdf_processor_op_m_set)
3746: (4)               op_l = property(_mupdf.pdf_processor_op_l_get,
_mupdf.pdf_processor_op_l_set)
3747: (4)               op_c = property(_mupdf.pdf_processor_op_c_get,
_mupdf.pdf_processor_op_c_set)
3748: (4)               op_v = property(_mupdf.pdf_processor_op_v_get,
_mupdf.pdf_processor_op_v_set)
3749: (4)               op_y = property(_mupdf.pdf_processor_op_y_get,
_mupdf.pdf_processor_op_y_set)
3750: (4)               op_h = property(_mupdf.pdf_processor_op_h_get,
_mupdf.pdf_processor_op_h_set)
```

```
3751: (4)               op_re = property(_mupdf.pdf_processor_op_re_get,
_mupdf.pdf_processor_op_re_set)
3752: (4)               op_S = property(_mupdf.pdf_processor_op_S_get,
_mupdf.pdf_processor_op_S_set)
3753: (4)               op_s = property(_mupdf.pdf_processor_op_s_get,
_mupdf.pdf_processor_op_s_set)
3754: (4)               op_F = property(_mupdf.pdf_processor_op_F_get,
_mupdf.pdf_processor_op_F_set)
3755: (4)               op_f = property(_mupdf.pdf_processor_op_f_get,
_mupdf.pdf_processor_op_f_set)
3756: (4)               op_fstar = property(_mupdf.pdf_processor_op_fstar_get,
_mupdf.pdf_processor_op_fstar_set)
3757: (4)               op_B = property(_mupdf.pdf_processor_op_B_get,
_mupdf.pdf_processor_op_B_set)
3758: (4)               op_Bstar = property(_mupdf.pdf_processor_op_Bstar_get,
_mupdf.pdf_processor_op_Bstar_set)
3759: (4)               op_b = property(_mupdf.pdf_processor_op_b_get,
_mupdf.pdf_processor_op_b_set)
3760: (4)               op_bstar = property(_mupdf.pdf_processor_op_bstar_get,
_mupdf.pdf_processor_op_bstar_set)
3761: (4)               op_n = property(_mupdf.pdf_processor_op_n_get,
_mupdf.pdf_processor_op_n_set)
3762: (4)               op_W = property(_mupdf.pdf_processor_op_W_get,
_mupdf.pdf_processor_op_W_set)
3763: (4)               op_Wstar = property(_mupdf.pdf_processor_op_Wstar_get,
_mupdf.pdf_processor_op_Wstar_set)
3764: (4)               op_BT = property(_mupdf.pdf_processor_op_BT_get,
_mupdf.pdf_processor_op_BT_set)
3765: (4)               op_ET = property(_mupdf.pdf_processor_op_ET_get,
_mupdf.pdf_processor_op_ET_set)
3766: (4)               op_Tc = property(_mupdf.pdf_processor_op_Tc_get,
_mupdf.pdf_processor_op_Tc_set)
3767: (4)               op_Tw = property(_mupdf.pdf_processor_op_Tw_get,
_mupdf.pdf_processor_op_Tw_set)
3768: (4)               op_Tz = property(_mupdf.pdf_processor_op_Tz_get,
_mupdf.pdf_processor_op_Tz_set)
3769: (4)               op_TL = property(_mupdf.pdf_processor_op_TL_get,
_mupdf.pdf_processor_op_TL_set)
3770: (4)               op_Tf = property(_mupdf.pdf_processor_op_Tf_get,
_mupdf.pdf_processor_op_Tf_set)
3771: (4)               op_Tr = property(_mupdf.pdf_processor_op_Tr_get,
_mupdf.pdf_processor_op_Tr_set)
3772: (4)               op_Ts = property(_mupdf.pdf_processor_op_Ts_get,
_mupdf.pdf_processor_op_Ts_set)
3773: (4)               op_Td = property(_mupdf.pdf_processor_op_Td_get,
_mupdf.pdf_processor_op_Td_set)
3774: (4)               op_TD = property(_mupdf.pdf_processor_op_TD_get,
_mupdf.pdf_processor_op_TD_set)
3775: (4)               op_Tm = property(_mupdf.pdf_processor_op_Tm_get,
_mupdf.pdf_processor_op_Tm_set)
3776: (4)               op_Tstar = property(_mupdf.pdf_processor_op_Tstar_get,
_mupdf.pdf_processor_op_Tstar_set)
3777: (4)               op_TJ = property(_mupdf.pdf_processor_op_TJ_get,
_mupdf.pdf_processor_op_TJ_set)
3778: (4)               op_Tj = property(_mupdf.pdf_processor_op_Tj_get,
_mupdf.pdf_processor_op_Tj_set)
3779: (4)               op_squote = property(_mupdf.pdf_processor_op_squote_get,
_mupdf.pdf_processor_op_squote_set)
3780: (4)               op_dquote = property(_mupdf.pdf_processor_op_dquote_get,
_mupdf.pdf_processor_op_dquote_set)
3781: (4)               op_d0 = property(_mupdf.pdf_processor_op_d0_get,
_mupdf.pdf_processor_op_d0_set)
3782: (4)               op_d1 = property(_mupdf.pdf_processor_op_d1_get,
_mupdf.pdf_processor_op_d1_set)
3783: (4)               op_CS = property(_mupdf.pdf_processor_op_CS_get,
_mupdf.pdf_processor_op_CS_set)
3784: (4)               op_cs = property(_mupdf.pdf_processor_op_cs_get,
_mupdf.pdf_processor_op_cs_set)
3785: (4)               op_SC_pattern = property(_mupdf.pdf_processor_op_SC_pattern_get,
```

```
_mupdf.pdf_processor_op_SC_pattern_set)
3786: (4)               op_sc_pattern = property(_mupdf.pdf_processor_op_sc_pattern_get,
_mupdf.pdf_processor_op_sc_pattern_set)
3787: (4)               op_SC_shade = property(_mupdf.pdf_processor_op_SC_shade_get,
_mupdf.pdf_processor_op_SC_shade_set)
3788: (4)               op_sc_shade = property(_mupdf.pdf_processor_op_sc_shade_get,
_mupdf.pdf_processor_op_sc_shade_set)
3789: (4)               op_SC_color = property(_mupdf.pdf_processor_op_SC_color_get,
_mupdf.pdf_processor_op_SC_color_set)
3790: (4)               op_sc_color = property(_mupdf.pdf_processor_op_sc_color_get,
_mupdf.pdf_processor_op_sc_color_set)
3791: (4)               op_G = property(_mupdf.pdf_processor_op_G_get,
_mupdf.pdf_processor_op_G_set)
3792: (4)               op_g = property(_mupdf.pdf_processor_op_g_get,
_mupdf.pdf_processor_op_g_set)
3793: (4)               op_RG = property(_mupdf.pdf_processor_op_RG_get,
_mupdf.pdf_processor_op_RG_set)
3794: (4)               op_rg = property(_mupdf.pdf_processor_op_rg_get,
_mupdf.pdf_processor_op_rg_set)
3795: (4)               op_K = property(_mupdf.pdf_processor_op_K_get,
_mupdf.pdf_processor_op_K_set)
3796: (4)               op_k = property(_mupdf.pdf_processor_op_k_get,
_mupdf.pdf_processor_op_k_set)
3797: (4)               op_BI = property(_mupdf.pdf_processor_op_BI_get,
_mupdf.pdf_processor_op_BI_set)
3798: (4)               op_sh = property(_mupdf.pdf_processor_op_sh_get,
_mupdf.pdf_processor_op_sh_set)
3799: (4)               op_Do_image = property(_mupdf.pdf_processor_op_Do_image_get,
_mupdf.pdf_processor_op_Do_image_set)
3800: (4)               op_Do_form = property(_mupdf.pdf_processor_op_Do_form_get,
_mupdf.pdf_processor_op_Do_form_set)
3801: (4)               op_MP = property(_mupdf.pdf_processor_op_MP_get,
_mupdf.pdf_processor_op_MP_set)
3802: (4)               op_DP = property(_mupdf.pdf_processor_op_DP_get,
_mupdf.pdf_processor_op_DP_set)
3803: (4)               op_BMC = property(_mupdf.pdf_processor_op_BMC_get,
_mupdf.pdf_processor_op_BMC_set)
3804: (4)               op_BDC = property(_mupdf.pdf_processor_op_BDC_get,
_mupdf.pdf_processor_op_BDC_set)
3805: (4)               op_EMC = property(_mupdf.pdf_processor_op_EMC_get,
_mupdf.pdf_processor_op_EMC_set)
3806: (4)               op_BX = property(_mupdf.pdf_processor_op_BX_get,
_mupdf.pdf_processor_op_BX_set)
3807: (4)               op_EX = property(_mupdf.pdf_processor_op_EX_get,
_mupdf.pdf_processor_op_EX_set)
3808: (4)               op_gs_OP = property(_mupdf.pdf_processor_op_gs_OP_get,
_mupdf.pdf_processor_op_gs_OP_set)
3809: (4)               op_gs_op = property(_mupdf.pdf_processor_op_gs_op_get,
_mupdf.pdf_processor_op_gs_op_set)
3810: (4)               op_gs_OPM = property(_mupdf.pdf_processor_op_gs_OPM_get,
_mupdf.pdf_processor_op_gs_OPM_set)
3811: (4)               op_gs_UseBlackPtComp =
property(_mupdf.pdf_processor_op_gs_UseBlackPtComp_get,
_mupdf.pdf_processor_op_gs_UseBlackPtComp_set)
3812: (4)               op_END = property(_mupdf.pdf_processor_op_END_get,
_mupdf.pdf_processor_op_END_set)
3813: (4)               usage = property(_mupdf.pdf_processor_usage_get,
_mupdf.pdf_processor_usage_set)
3814: (4)               hidden = property(_mupdf.pdf_processor_hidden_get,
_mupdf.pdf_processor_hidden_set)
3815: (4)               requirements = property(_mupdf.pdf_processor_requirements_get,
_mupdf.pdf_processor_requirements_set)
3816: (4)               def __init__(self):
3817: (8)                   _mupdf.pdf_processor_swiginit(self, _mupdf.new_pdf_processor())
3818: (4)               __swig_destroy__ = _mupdf.delete_pdf_processor
3819: (0)           _mupdf.pdf_processor_swigregister(pdf_processor)
3820: (0)           class pdf_csi(object):
3821: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
3822: (4)                __repr__ = _swig_repr
3823: (4)                doc = property(_mupdf.pdf_csi_doc_get, _mupdf.pdf_csi_doc_set)
3824: (4)                rdb = property(_mupdf.pdf_csi_rdb_get, _mupdf.pdf_csi_rdb_set)
3825: (4)                buf = property(_mupdf.pdf_csi_buf_get, _mupdf.pdf_csi_buf_set)
3826: (4)                cookie = property(_mupdf.pdf_csi_cookie_get, _mupdf.pdf_csi_cookie_set)
3827: (4)                gstate = property(_mupdf.pdf_csi_gstate_get, _mupdf.pdf_csi_gstate_set)
3828: (4)                xbalance = property(_mupdf.pdf_csi_xbalance_get,
_mupdf.pdf_csi_xbalance_set)
3829: (4)                in_text = property(_mupdf.pdf_csi_in_text_get, _mupdf.pdf_csi_in_text_set)
3830: (4)                d1_rect = property(_mupdf.pdf_csi_d1_rect_get, _mupdf.pdf_csi_d1_rect_set)
3831: (4)                obj = property(_mupdf.pdf_csi_obj_get, _mupdf.pdf_csi_obj_set)
3832: (4)                name = property(_mupdf.pdf_csi_name_get, _mupdf.pdf_csi_name_set)
3833: (4)                string = property(_mupdf.pdf_csi_string_get, _mupdf.pdf_csi_string_set)
3834: (4)                string_len = property(_mupdf.pdf_csi_string_len_get,
_mupdf.pdf_csi_string_len_set)
3835: (4)                top = property(_mupdf.pdf_csi_top_get, _mupdf.pdf_csi_top_set)
3836: (4)                stack = property(_mupdf.pdf_csi_stack_get, _mupdf.pdf_csi_stack_set)
3837: (4)                def __init__(self):
3838: (8)                    _mupdf.pdf_csi_swiginit(self, _mupdf.new_pdf_csi())
3839: (4)                __swig_destroy__ = _mupdf.delete_pdf_csi
3840: (0)            _mupdf.pdf_csi_swigregister(pdf_csi)
3841: (0)            class pdf_filter_factory(object):
3842: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3843: (4)                __repr__ = _swig_repr
3844: (4)                filter = property(_mupdf.pdf_filter_factory_filter_get,
_mupdf.pdf_filter_factory_filter_set)
3845: (4)                options = property(_mupdf.pdf_filter_factory_options_get,
_mupdf.pdf_filter_factory_options_set)
3846: (4)                def __init__(self):
3847: (8)                    _mupdf.pdf_filter_factory_swiginit(self,
_mupdf.new_pdf_filter_factory())
3848: (4)                __swig_destroy__ = _mupdf.delete_pdf_filter_factory
3849: (0)            _mupdf.pdf_filter_factory_swigregister(pdf_filter_factory)
3850: (0)            class pdf_filter_options(object):
3851: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3852: (4)                __repr__ = _swig_repr
3853: (4)                recurse = property(_mupdf.pdf_filter_options_recurse_get,
_mupdf.pdf_filter_options_recurse_set)
3854: (4)                instance_forms = property(_mupdf.pdf_filter_options_instance_forms_get,
_mupdf.pdf_filter_options_instance_forms_set)
3855: (4)                ascii = property(_mupdf.pdf_filter_options_ascii_get,
_mupdf.pdf_filter_options_ascii_set)
3856: (4)                no_update = property(_mupdf.pdf_filter_options_no_update_get,
_mupdf.pdf_filter_options_no_update_set)
3857: (4)                opaque = property(_mupdf.pdf_filter_options_opaque_get,
_mupdf.pdf_filter_options_opaque_set)
3858: (4)                complete = property(_mupdf.pdf_filter_options_complete_get,
_mupdf.pdf_filter_options_complete_set)
3859: (4)                filters = property(_mupdf.pdf_filter_options_filters_get,
_mupdf.pdf_filter_options_filters_set)
3860: (4)                newlines = property(_mupdf.pdf_filter_options_newlines_get,
_mupdf.pdf_filter_options_newlines_set)
3861: (4)                def __init__(self):
3862: (8)                    _mupdf.pdf_filter_options_swiginit(self,
_mupdf.new_pdf_filter_options())
3863: (4)                __swig_destroy__ = _mupdf.delete_pdf_filter_options
3864: (0)            _mupdf.pdf_filter_options_swigregister(pdf_filter_options)
3865: (0)            FZ_CULL_PATH_FILL = _mupdf.FZ_CULL_PATH_FILL
3866: (0)            FZ_CULL_PATH_STROKE = _mupdf.FZ_CULL_PATH_STROKE
3867: (0)            FZ_CULL_PATH_FILL_STROKE = _mupdf.FZ_CULL_PATH_FILL_STROKE
3868: (0)            FZ_CULL_CLIP_PATH = _mupdf.FZ_CULL_CLIP_PATH
3869: (0)            FZ_CULL_GLYPH = _mupdf.FZ_CULL_GLYPH
3870: (0)            FZ_CULL_IMAGE = _mupdf.FZ_CULL_IMAGE
3871: (0)            FZ_CULL_SHADING = _mupdf.FZ_CULL_SHADING
3872: (0)            class pdf_sanitize_filter_options(object):
3873: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
3874: (4)                __repr__ = _swig_repr
3875: (4)                opaque = property(_mupdf.pdf_sanitize_filter_options_opaque_get,
_mupdf.pdf_sanitize_filter_options_opaque_set)
3876: (4)                image_filter =
property(_mupdf.pdf_sanitize_filter_options_image_filter_get,
_mupdf.pdf_sanitize_filter_options_image_filter_set)
3877: (4)                text_filter = property(_mupdf.pdf_sanitize_filter_options_text_filter_get,
_mupdf.pdf_sanitize_filter_options_text_filter_set)
3878: (4)                after_text_object =
property(_mupdf.pdf_sanitize_filter_options_after_text_object_get,
_mupdf.pdf_sanitize_filter_options_after_text_object_set)
3879: (4)                culler = property(_mupdf.pdf_sanitize_filter_options_culler_get,
_mupdf.pdf_sanitize_filter_options_culler_set)
3880: (4)                def __init__(self):
3881: (8)                    _mupdf.pdf_sanitize_filter_options_swiginit(self,
_mupdf.new_pdf_sanitize_filter_options())
3882: (4)                __swig_destroy__ = _mupdf.delete_pdf_sanitize_filter_options
3883: (0)            _mupdf.pdf_sanitize_filter_options_swigregister(pdf_sanitize_filter_options)
3884: (0)            class pdf_color_filter_options(object):
3885: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3886: (4)                __repr__ = _swig_repr
3887: (4)                opaque = property(_mupdf.pdf_color_filter_options_opaque_get,
_mupdf.pdf_color_filter_options_opaque_set)
3888: (4)                color_rewrite =
property(_mupdf.pdf_color_filter_options_color_rewrite_get,
_mupdf.pdf_color_filter_options_color_rewrite_set)
3889: (4)                image_rewrite =
property(_mupdf.pdf_color_filter_options_image_rewrite_get,
_mupdf.pdf_color_filter_options_image_rewrite_set)
3890: (4)                shade_rewrite =
property(_mupdf.pdf_color_filter_options_shade_rewrite_get,
_mupdf.pdf_color_filter_options_shade_rewrite_set)
3891: (4)                repeated_image_rewrite =
property(_mupdf.pdf_color_filter_options_repeated_image_rewrite_get,
_mupdf.pdf_color_filter_options_repeated_image_rewrite_set)
3892: (4)                def __init__(self):
3893: (8)                    _mupdf.pdf_color_filter_options_swiginit(self,
_mupdf.new_pdf_color_filter_options())
3894: (4)                __swig_destroy__ = _mupdf.delete_pdf_color_filter_options
3895: (0)            _mupdf.pdf_color_filter_options_swigregister(pdf_color_filter_options)
3896: (0)            class pdf_text_state(object):
3897: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3898: (4)                __repr__ = _swig_repr
3899: (4)                char_space = property(_mupdf.pdf_text_state_char_space_get,
_mupdf.pdf_text_state_char_space_set)
3900: (4)                word_space = property(_mupdf.pdf_text_state_word_space_get,
_mupdf.pdf_text_state_word_space_set)
3901: (4)                scale = property(_mupdf.pdf_text_state_scale_get,
_mupdf.pdf_text_state_scale_set)
3902: (4)                leading = property(_mupdf.pdf_text_state_leading_get,
_mupdf.pdf_text_state_leading_set)
3903: (4)                font = property(_mupdf.pdf_text_state_font_get,
_mupdf.pdf_text_state_font_set)
3904: (4)                fontname = property(_mupdf.pdf_text_state_fontname_get,
_mupdf.pdf_text_state_fontname_set)
3905: (4)                size = property(_mupdf.pdf_text_state_size_get,
_mupdf.pdf_text_state_size_set)
3906: (4)                render = property(_mupdf.pdf_text_state_render_get,
_mupdf.pdf_text_state_render_set)
3907: (4)                rise = property(_mupdf.pdf_text_state_rise_get,
_mupdf.pdf_text_state_rise_set)
3908: (4)                def __init__(self):
3909: (8)                    _mupdf.pdf_text_state_swiginit(self, _mupdf.new_pdf_text_state())
3910: (4)                __swig_destroy__ = _mupdf.delete_pdf_text_state
3911: (0)            _mupdf.pdf_text_state_swigregister(pdf_text_state)
3912: (0)            class pdf_text_object_state(object):
3913: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
3914: (4)               __repr__ = _swig_repr
3915: (4)               text = property(_mupdf.pdf_text_object_state_text_get,
_mupdf.pdf_text_object_state_text_set)
3916: (4)               text_bbox = property(_mupdf.pdf_text_object_state_text_bbox_get,
_mupdf.pdf_text_object_state_text_bbox_set)
3917: (4)               tlm = property(_mupdf.pdf_text_object_state_tlm_get,
_mupdf.pdf_text_object_state_tlm_set)
3918: (4)               tm = property(_mupdf.pdf_text_object_state_tm_get,
_mupdf.pdf_text_object_state_tm_set)
3919: (4)               text_mode = property(_mupdf.pdf_text_object_state_text_mode_get,
_mupdf.pdf_text_object_state_text_mode_set)
3920: (4)               cid = property(_mupdf.pdf_text_object_state_cid_get,
_mupdf.pdf_text_object_state_cid_set)
3921: (4)               gid = property(_mupdf.pdf_text_object_state_gid_get,
_mupdf.pdf_text_object_state_gid_set)
3922: (4)               char_bbox = property(_mupdf.pdf_text_object_state_char_bbox_get,
_mupdf.pdf_text_object_state_char_bbox_set)
3923: (4)               fontdesc = property(_mupdf.pdf_text_object_state_fontdesc_get,
_mupdf.pdf_text_object_state_fontdesc_set)
3924: (4)               char_tx = property(_mupdf.pdf_text_object_state_char_tx_get,
_mupdf.pdf_text_object_state_char_tx_set)
3925: (4)               char_ty = property(_mupdf.pdf_text_object_state_char_ty_get,
_mupdf.pdf_text_object_state_char_ty_set)
3926: (4)               def __init__(self):
3927: (8)                   _mupdf.pdf_text_object_state_swiginit(self,
_mupdf.new_pdf_text_object_state())
3928: (4)               __swig_destroy__ = _mupdf.delete_pdf_text_object_state
3929: (0)           _mupdf.pdf_text_object_state_swigregister(pdf_text_object_state)
3930: (0)           PDF_REDACT_IMAGE_NONE = _mupdf.PDF_REDACT_IMAGE_NONE
3931: (0)           PDF_REDACT_IMAGE_REMOVE = _mupdf.PDF_REDACT_IMAGE_REMOVE
3932: (0)           PDF_REDACT_IMAGE_PIXELS = _mupdf.PDF_REDACT_IMAGE_PIXELS
3933: (0)           PDF_REDACT_IMAGE_REMOVE_UNLESS_INVISIBLE =
_mupdf.PDF_REDACT_IMAGE_REMOVE_UNLESS_INVISIBLE
3934: (0)           PDF_REDACT_LINE_ART_NONE = _mupdf.PDF_REDACT_LINE_ART_NONE
3935: (0)           PDF_REDACT_LINE_ART_REMOVE_IF_COVERED =
_mupdf.PDF_REDACT_LINE_ART_REMOVE_IF_COVERED
3936: (0)           PDF_REDACT_LINE_ART_REMOVE_IF_TOUCHED =
_mupdf.PDF_REDACT_LINE_ART_REMOVE_IF_TOUCHED
3937: (0)           PDF_REDACT_TEXT_REMOVE = _mupdf.PDF_REDACT_TEXT_REMOVE
3938: (0)           PDF_REDACT_TEXT_NONE = _mupdf.PDF_REDACT_TEXT_NONE
3939: (0)           class pdf_redact_options(object):
3940: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3941: (4)               __repr__ = _swig_repr
3942: (4)               black_boxes = property(_mupdf.pdf_redact_options_black_boxes_get,
_mupdf.pdf_redact_options_black_boxes_set)
3943: (4)               image_method = property(_mupdf.pdf_redact_options_image_method_get,
_mupdf.pdf_redact_options_image_method_set)
3944: (4)               line_art = property(_mupdf.pdf_redact_options_line_art_get,
_mupdf.pdf_redact_options_line_art_set)
3945: (4)               text = property(_mupdf.pdf_redact_options_text_get,
_mupdf.pdf_redact_options_text_set)
3946: (4)               def __init__(self):
3947: (8)                   _mupdf.pdf_redact_options_swiginit(self,
_mupdf.new_pdf_redact_options())
3948: (4)               __swig_destroy__ = _mupdf.delete_pdf_redact_options
3949: (0)           _mupdf.pdf_redact_options_swigregister(pdf_redact_options)
3950: (0)           class pdf_page(object):
3951: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
3952: (4)               __repr__ = _swig_repr
3953: (4)               super = property(_mupdf.pdf_page_super_get, _mupdf.pdf_page_super_set)
3954: (4)               doc = property(_mupdf.pdf_page_doc_get, _mupdf.pdf_page_doc_set)
3955: (4)               obj = property(_mupdf.pdf_page_obj_get, _mupdf.pdf_page_obj_set)
3956: (4)               transparency = property(_mupdf.pdf_page_transparency_get,
_mupdf.pdf_page_transparency_set)
3957: (4)               overprint = property(_mupdf.pdf_page_overprint_get,
_mupdf.pdf_page_overprint_set)
```

```
3958: (4)               links = property(_mupdf.pdf_page_links_get, _mupdf.pdf_page_links_set)
3959: (4)               annots = property(_mupdf.pdf_page_annots_get, _mupdf.pdf_page_annots_set)
3960: (4)               annot_tailp = property(_mupdf.pdf_page_annot_tailp_get,
_mupdf.pdf_page_annot_tailp_set)
3961: (4)               widgets = property(_mupdf.pdf_page_widgets_get,
_mupdf.pdf_page_widgets_set)
3962: (4)               widget_tailp = property(_mupdf.pdf_page_widget_tailp_get,
_mupdf.pdf_page_widget_tailp_set)
3963: (4)               def __init__(self):
3964: (8)                   _mupdf.pdf_page_swiginit(self, _mupdf.new_pdf_page())
3965: (4)               __swig_destroy__ = _mupdf.delete_pdf_page
3966: (0)           _mupdf.pdf_page_swigregister(pdf_page)
3967: (0)           PDF_ANNOT_TEXT = _mupdf.PDF_ANNOT_TEXT
3968: (0)           PDF_ANNOT_LINK = _mupdf.PDF_ANNOT_LINK
3969: (0)           PDF_ANNOT_FREE_TEXT = _mupdf.PDF_ANNOT_FREE_TEXT
3970: (0)           PDF_ANNOT_LINE = _mupdf.PDF_ANNOT_LINE
3971: (0)           PDF_ANNOT_SQUARE = _mupdf.PDF_ANNOT_SQUARE
3972: (0)           PDF_ANNOT_CIRCLE = _mupdf.PDF_ANNOT_CIRCLE
3973: (0)           PDF_ANNOT_POLYGON = _mupdf.PDF_ANNOT_POLYGON
3974: (0)           PDF_ANNOT_POLY_LINE = _mupdf.PDF_ANNOT_POLY_LINE
3975: (0)           PDF_ANNOT_HIGHLIGHT = _mupdf.PDF_ANNOT_HIGHLIGHT
3976: (0)           PDF_ANNOT_UNDERLINE = _mupdf.PDF_ANNOT_UNDERLINE
3977: (0)           PDF_ANNOT_SQUIGGLY = _mupdf.PDF_ANNOT_SQUIGGLY
3978: (0)           PDF_ANNOT_STRIKE_OUT = _mupdf.PDF_ANNOT_STRIKE_OUT
3979: (0)           PDF_ANNOT_REDACT = _mupdf.PDF_ANNOT_REDACT
3980: (0)           PDF_ANNOT_STAMP = _mupdf.PDF_ANNOT_STAMP
3981: (0)           PDF_ANNOT_CARET = _mupdf.PDF_ANNOT_CARET
3982: (0)           PDF_ANNOT_INK = _mupdf.PDF_ANNOT_INK
3983: (0)           PDF_ANNOT_POPUP = _mupdf.PDF_ANNOT_POPUP
3984: (0)           PDF_ANNOT_FILE_ATTACHMENT = _mupdf.PDF_ANNOT_FILE_ATTACHMENT
3985: (0)           PDF_ANNOT_SOUND = _mupdf.PDF_ANNOT_SOUND
3986: (0)           PDF_ANNOT_MOVIE = _mupdf.PDF_ANNOT_MOVIE
3987: (0)           PDF_ANNOT_RICH_MEDIA = _mupdf.PDF_ANNOT_RICH_MEDIA
3988: (0)           PDF_ANNOT_WIDGET = _mupdf.PDF_ANNOT_WIDGET
3989: (0)           PDF_ANNOT_SCREEN = _mupdf.PDF_ANNOT_SCREEN
3990: (0)           PDF_ANNOT_PRINTER_MARK = _mupdf.PDF_ANNOT_PRINTER_MARK
3991: (0)           PDF_ANNOT_TRAP_NET = _mupdf.PDF_ANNOT_TRAP_NET
3992: (0)           PDF_ANNOT_WATERMARK = _mupdf.PDF_ANNOT_WATERMARK
3993: (0)           PDF_ANNOT_3D = _mupdf.PDF_ANNOT_3D
3994: (0)           PDF_ANNOT_PROJECTION = _mupdf.PDF_ANNOT_PROJECTION
3995: (0)           PDF_ANNOT_UNKNOWN = _mupdf.PDF_ANNOT_UNKNOWN
3996: (0)           PDF_ANNOT_IS_INVISIBLE = _mupdf.PDF_ANNOT_IS_INVISIBLE
3997: (0)           PDF_ANNOT_IS_HIDDEN = _mupdf.PDF_ANNOT_IS_HIDDEN
3998: (0)           PDF_ANNOT_IS_PRINT = _mupdf.PDF_ANNOT_IS_PRINT
3999: (0)           PDF_ANNOT_IS_NO_ZOOM = _mupdf.PDF_ANNOT_IS_NO_ZOOM
4000: (0)           PDF_ANNOT_IS_NO_ROTATE = _mupdf.PDF_ANNOT_IS_NO_ROTATE
4001: (0)           PDF_ANNOT_IS_NO_VIEW = _mupdf.PDF_ANNOT_IS_NO_VIEW
4002: (0)           PDF_ANNOT_IS_READ_ONLY = _mupdf.PDF_ANNOT_IS_READ_ONLY
4003: (0)           PDF_ANNOT_IS_LOCKED = _mupdf.PDF_ANNOT_IS_LOCKED
4004: (0)           PDF_ANNOT_IS_TOGGLE_NO_VIEW = _mupdf.PDF_ANNOT_IS_TOGGLE_NO_VIEW
4005: (0)           PDF_ANNOT_IS_LOCKED_CONTENTS = _mupdf.PDF_ANNOT_IS_LOCKED_CONTENTS
4006: (0)           PDF_ANNOT_LE_NONE = _mupdf.PDF_ANNOT_LE_NONE
4007: (0)           PDF_ANNOT_LE_SQUARE = _mupdf.PDF_ANNOT_LE_SQUARE
4008: (0)           PDF_ANNOT_LE_CIRCLE = _mupdf.PDF_ANNOT_LE_CIRCLE
4009: (0)           PDF_ANNOT_LE_DIAMOND = _mupdf.PDF_ANNOT_LE_DIAMOND
4010: (0)           PDF_ANNOT_LE_OPEN_ARROW = _mupdf.PDF_ANNOT_LE_OPEN_ARROW
4011: (0)           PDF_ANNOT_LE_CLOSED_ARROW = _mupdf.PDF_ANNOT_LE_CLOSED_ARROW
4012: (0)           PDF_ANNOT_LE_BUTT = _mupdf.PDF_ANNOT_LE_BUTT
4013: (0)           PDF_ANNOT_LE_R_OPEN_ARROW = _mupdf.PDF_ANNOT_LE_R_OPEN_ARROW
4014: (0)           PDF_ANNOT_LE_R_CLOSED_ARROW = _mupdf.PDF_ANNOT_LE_R_CLOSED_ARROW
4015: (0)           PDF_ANNOT_LE_SLASH = _mupdf.PDF_ANNOT_LE_SLASH
4016: (0)           PDF_ANNOT_Q_LEFT = _mupdf.PDF_ANNOT_Q_LEFT
4017: (0)           PDF_ANNOT_Q_CENTER = _mupdf.PDF_ANNOT_Q_CENTER
4018: (0)           PDF_ANNOT_Q_RIGHT = _mupdf.PDF_ANNOT_Q_RIGHT
4019: (0)           PDF_ANNOT_IT_DEFAULT = _mupdf.PDF_ANNOT_IT_DEFAULT
4020: (0)           PDF_ANNOT_IT_FREETEXT_CALLOUT = _mupdf.PDF_ANNOT_IT_FREETEXT_CALLOUT
4021: (0)           PDF_ANNOT_IT_FREETEXT_TYPEWRITER = _mupdf.PDF_ANNOT_IT_FREETEXT_TYPEWRITER
4022: (0)           PDF_ANNOT_IT_LINE_ARROW = _mupdf.PDF_ANNOT_IT_LINE_ARROW
4023: (0)           PDF_ANNOT_IT_LINE_DIMENSION = _mupdf.PDF_ANNOT_IT_LINE_DIMENSION
```

```
4024: (0)           PDF_ANNOT_IT_POLYLINE_DIMENSION = _mupdf.PDF_ANNOT_IT_POLYLINE_DIMENSION
4025: (0)           PDF_ANNOT_IT_POLYGON_CLOUD = _mupdf.PDF_ANNOT_IT_POLYGON_CLOUD
4026: (0)           PDF_ANNOT_IT_POLYGON_DIMENSION = _mupdf.PDF_ANNOT_IT_POLYGON_DIMENSION
4027: (0)           PDF_ANNOT_IT_STAMP_IMAGE = _mupdf.PDF_ANNOT_IT_STAMP_IMAGE
4028: (0)           PDF_ANNOT_IT_STAMP_SNAPSHOT = _mupdf.PDF_ANNOT_IT_STAMP_SNAPSHOT
4029: (0)           PDF_ANNOT_IT_UNKNOWN = _mupdf.PDF_ANNOT_IT_UNKNOWN
4030: (0)           PDF_BORDER_STYLE_SOLID = _mupdf.PDF_BORDER_STYLE_SOLID
4031: (0)           PDF_BORDER_STYLE_DASHED = _mupdf.PDF_BORDER_STYLE_DASHED
4032: (0)           PDF_BORDER_STYLE_BEVELED = _mupdf.PDF_BORDER_STYLE_BEVELED
4033: (0)           PDF_BORDER_STYLE_INSET = _mupdf.PDF_BORDER_STYLE_INSET
4034: (0)           PDF_BORDER_STYLE_UNDERLINE = _mupdf.PDF_BORDER_STYLE_UNDERLINE
4035: (0)           PDF_BORDER_EFFECT_NONE = _mupdf.PDF_BORDER_EFFECT_NONE
4036: (0)           PDF_BORDER_EFFECT_CLOUDY = _mupdf.PDF_BORDER_EFFECT_CLOUDY
4037: (0)           class pdf_embedded_file_params(object):
4038: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4039: (4)               __repr__ = _swig_repr
4040: (4)               filename = property(_mupdf.pdf_embedded_file_params_filename_get,
_mupdf.pdf_embedded_file_params_filename_set)
4041: (4)               mimetype = property(_mupdf.pdf_embedded_file_params_mimetype_get,
_mupdf.pdf_embedded_file_params_mimetype_set)
4042: (4)               size = property(_mupdf.pdf_embedded_file_params_size_get,
_mupdf.pdf_embedded_file_params_size_set)
4043: (4)               created = property(_mupdf.pdf_embedded_file_params_created_get,
_mupdf.pdf_embedded_file_params_created_set)
4044: (4)               modified = property(_mupdf.pdf_embedded_file_params_modified_get,
_mupdf.pdf_embedded_file_params_modified_set)
4045: (4)               def __init__(self):
4046: (8)                   _mupdf.pdf_embedded_file_params_swiginit(self,
_mupdf.new_pdf_embedded_file_params())
4047: (4)               __swig_destroy__ = _mupdf.delete_pdf_embedded_file_params
4048: (0)           _mupdf.pdf_embedded_file_params_swigregister(pdf_embedded_file_params)
4049: (0)           PDF_WIDGET_TYPE_UNKNOWN = _mupdf.PDF_WIDGET_TYPE_UNKNOWN
4050: (0)           PDF_WIDGET_TYPE_BUTTON = _mupdf.PDF_WIDGET_TYPE_BUTTON
4051: (0)           PDF_WIDGET_TYPE_CHECKBOX = _mupdf.PDF_WIDGET_TYPE_CHECKBOX
4052: (0)           PDF_WIDGET_TYPE_COMBOBOX = _mupdf.PDF_WIDGET_TYPE_COMBOBOX
4053: (0)           PDF_WIDGET_TYPE_LISTBOX = _mupdf.PDF_WIDGET_TYPE_LISTBOX
4054: (0)           PDF_WIDGET_TYPE_RADIOBUTTON = _mupdf.PDF_WIDGET_TYPE_RADIOBUTTON
4055: (0)           PDF_WIDGET_TYPE_SIGNATURE = _mupdf.PDF_WIDGET_TYPE_SIGNATURE
4056: (0)           PDF_WIDGET_TYPE_TEXT = _mupdf.PDF_WIDGET_TYPE_TEXT
4057: (0)           PDF_WIDGET_TX_FORMAT_NONE = _mupdf.PDF_WIDGET_TX_FORMAT_NONE
4058: (0)           PDF_WIDGET_TX_FORMAT_NUMBER = _mupdf.PDF_WIDGET_TX_FORMAT_NUMBER
4059: (0)           PDF_WIDGET_TX_FORMAT_SPECIAL = _mupdf.PDF_WIDGET_TX_FORMAT_SPECIAL
4060: (0)           PDF_WIDGET_TX_FORMAT_DATE = _mupdf.PDF_WIDGET_TX_FORMAT_DATE
4061: (0)           PDF_WIDGET_TX_FORMAT_TIME = _mupdf.PDF_WIDGET_TX_FORMAT_TIME
4062: (0)           PDF_FIELD_IS_READ_ONLY = _mupdf.PDF_FIELD_IS_READ_ONLY
4063: (0)           PDF_FIELD_IS_REQUIRED = _mupdf.PDF_FIELD_IS_REQUIRED
4064: (0)           PDF_FIELD_IS_NO_EXPORT = _mupdf.PDF_FIELD_IS_NO_EXPORT
4065: (0)           PDF_TX_FIELD_IS_MULTILINE = _mupdf.PDF_TX_FIELD_IS_MULTILINE
4066: (0)           PDF_TX_FIELD_IS_PASSWORD = _mupdf.PDF_TX_FIELD_IS_PASSWORD
4067: (0)           PDF_TX_FIELD_IS_FILE_SELECT = _mupdf.PDF_TX_FIELD_IS_FILE_SELECT
4068: (0)           PDF_TX_FIELD_IS_DO_NOT_SPELL_CHECK = _mupdf.PDF_TX_FIELD_IS_DO_NOT_SPELL_CHECK
4069: (0)           PDF_TX_FIELD_IS_DO_NOT_SCROLL = _mupdf.PDF_TX_FIELD_IS_DO_NOT_SCROLL
4070: (0)           PDF_TX_FIELD_IS_COMB = _mupdf.PDF_TX_FIELD_IS_COMB
4071: (0)           PDF_TX_FIELD_IS_RICH_TEXT = _mupdf.PDF_TX_FIELD_IS_RICH_TEXT
4072: (0)           PDF_BTN_FIELD_IS_NO_TOGGLE_TO_OFF = _mupdf.PDF_BTN_FIELD_IS_NO_TOGGLE_TO_OFF
4073: (0)           PDF_BTN_FIELD_IS_RADIO = _mupdf.PDF_BTN_FIELD_IS_RADIO
4074: (0)           PDF_BTN_FIELD_IS_PUSHBUTTON = _mupdf.PDF_BTN_FIELD_IS_PUSHBUTTON
4075: (0)           PDF_BTN_FIELD_IS_RADIOS_IN_UNISON = _mupdf.PDF_BTN_FIELD_IS_RADIOS_IN_UNISON
4076: (0)           PDF_CH_FIELD_IS_COMBO = _mupdf.PDF_CH_FIELD_IS_COMBO
4077: (0)           PDF_CH_FIELD_IS_EDIT = _mupdf.PDF_CH_FIELD_IS_EDIT
4078: (0)           PDF_CH_FIELD_IS_SORT = _mupdf.PDF_CH_FIELD_IS_SORT
4079: (0)           PDF_CH_FIELD_IS_MULTI_SELECT = _mupdf.PDF_CH_FIELD_IS_MULTI_SELECT
4080: (0)           PDF_CH_FIELD_IS_DO_NOT_SPELL_CHECK = _mupdf.PDF_CH_FIELD_IS_DO_NOT_SPELL_CHECK
4081: (0)           PDF_CH_FIELD_IS_COMMIT_ON_SEL_CHANGE =
_mupdf.PDF_CH_FIELD_IS_COMMIT_ON_SEL_CHANGE
4082: (0)           class pdf_pkcs7_distinguished_name(object):
4083: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
4084: (4)                __repr__ = _swig_repr
4085: (4)                cn = property(_mupdf.pdf_pkcs7_distinguished_name_cn_get,
_mupdf.pdf_pkcs7_distinguished_name_cn_set)
4086: (4)                o = property(_mupdf.pdf_pkcs7_distinguished_name_o_get,
_mupdf.pdf_pkcs7_distinguished_name_o_set)
4087: (4)                ou = property(_mupdf.pdf_pkcs7_distinguished_name_ou_get,
_mupdf.pdf_pkcs7_distinguished_name_ou_set)
4088: (4)                email = property(_mupdf.pdf_pkcs7_distinguished_name_email_get,
_mupdf.pdf_pkcs7_distinguished_name_email_set)
4089: (4)                c = property(_mupdf.pdf_pkcs7_distinguished_name_c_get,
_mupdf.pdf_pkcs7_distinguished_name_c_set)
4090: (4)                def __init__(self):
4091: (8)                    _mupdf.pdf_pkcs7_distinguished_name_swiginit(self,
_mupdf.new_pdf_pkcs7_distinguished_name())
4092: (4)                __swig_destroy__ = _mupdf.delete_pdf_pkcs7_distinguished_name
4093: (0)            _mupdf.pdf_pkcs7_distinguished_name_swigregister(pdf_pkcs7_distinguished_name)
4094: (0)            PDF_SIGNATURE_ERROR_OKAY = _mupdf.PDF_SIGNATURE_ERROR_OKAY
4095: (0)            PDF_SIGNATURE_ERROR_NO_SIGNATURES = _mupdf.PDF_SIGNATURE_ERROR_NO_SIGNATURES
4096: (0)            PDF_SIGNATURE_ERROR_NO_CERTIFICATE = _mupdf.PDF_SIGNATURE_ERROR_NO_CERTIFICATE
4097: (0)            PDF_SIGNATURE_ERROR_DIGEST_FAILURE = _mupdf.PDF_SIGNATURE_ERROR_DIGEST_FAILURE
4098: (0)            PDF_SIGNATURE_ERROR_SELF_SIGNED = _mupdf.PDF_SIGNATURE_ERROR_SELF_SIGNED
4099: (0)            PDF_SIGNATURE_ERROR_SELF_SIGNED_IN_CHAIN =
_mupdf.PDF_SIGNATURE_ERROR_SELF_SIGNED_IN_CHAIN
4100: (0)            PDF_SIGNATURE_ERROR_NOT_TRUSTED = _mupdf.PDF_SIGNATURE_ERROR_NOT_TRUSTED
4101: (0)            PDF_SIGNATURE_ERROR_UNKNOWN = _mupdf.PDF_SIGNATURE_ERROR_UNKNOWN
4102: (0)            class pdf_pkcs7_signer(object):
4103: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4104: (4)                __repr__ = _swig_repr
4105: (4)                keep = property(_mupdf.pdf_pkcs7_signer_keep_get,
_mupdf.pdf_pkcs7_signer_keep_set)
4106: (4)                drop = property(_mupdf.pdf_pkcs7_signer_drop_get,
_mupdf.pdf_pkcs7_signer_drop_set)
4107: (4)                get_signing_name = property(_mupdf.pdf_pkcs7_signer_get_signing_name_get,
_mupdf.pdf_pkcs7_signer_get_signing_name_set)
4108: (4)                max_digest_size = property(_mupdf.pdf_pkcs7_signer_max_digest_size_get,
_mupdf.pdf_pkcs7_signer_max_digest_size_set)
4109: (4)                create_digest = property(_mupdf.pdf_pkcs7_signer_create_digest_get,
_mupdf.pdf_pkcs7_signer_create_digest_set)
4110: (4)                def __init__(self):
4111: (8)                    _mupdf.pdf_pkcs7_signer_swiginit(self, _mupdf.new_pdf_pkcs7_signer())
4112: (4)                __swig_destroy__ = _mupdf.delete_pdf_pkcs7_signer
4113: (0)            _mupdf.pdf_pkcs7_signer_swigregister(pdf_pkcs7_signer)
4114: (0)            class pdf_pkcs7_verifier(object):
4115: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4116: (4)                __repr__ = _swig_repr
4117: (4)                drop = property(_mupdf.pdf_pkcs7_verifier_drop_get,
_mupdf.pdf_pkcs7_verifier_drop_set)
4118: (4)                check_certificate =
property(_mupdf.pdf_pkcs7_verifier_check_certificate_get,
_mupdf.pdf_pkcs7_verifier_check_certificate_set)
4119: (4)                check_digest = property(_mupdf.pdf_pkcs7_verifier_check_digest_get,
_mupdf.pdf_pkcs7_verifier_check_digest_set)
4120: (4)                get_signatory = property(_mupdf.pdf_pkcs7_verifier_get_signatory_get,
_mupdf.pdf_pkcs7_verifier_get_signatory_set)
4121: (4)                def __init__(self):
4122: (8)                    _mupdf.pdf_pkcs7_verifier_swiginit(self,
_mupdf.new_pdf_pkcs7_verifier())
4123: (4)                __swig_destroy__ = _mupdf.delete_pdf_pkcs7_verifier
4124: (0)            _mupdf.pdf_pkcs7_verifier_swigregister(pdf_pkcs7_verifier)
4125: (0)            PDF_SIGNATURE_SHOW_LABELS = _mupdf.PDF_SIGNATURE_SHOW_LABELS
4126: (0)            PDF_SIGNATURE_SHOW_DN = _mupdf.PDF_SIGNATURE_SHOW_DN
4127: (0)            PDF_SIGNATURE_SHOW_DATE = _mupdf.PDF_SIGNATURE_SHOW_DATE
4128: (0)            PDF_SIGNATURE_SHOW_TEXT_NAME = _mupdf.PDF_SIGNATURE_SHOW_TEXT_NAME
4129: (0)            PDF_SIGNATURE_SHOW_GRAPHIC_NAME = _mupdf.PDF_SIGNATURE_SHOW_GRAPHIC_NAME
4130: (0)            PDF_SIGNATURE_SHOW_LOGO = _mupdf.PDF_SIGNATURE_SHOW_LOGO
4131: (0)            class pdf_keystroke_event(object):
4132: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
4133: (4)               __repr__ = _swig_repr
4134: (4)               value = property(_mupdf.pdf_keystroke_event_value_get,
_mupdf.pdf_keystroke_event_value_set)
4135: (4)               change = property(_mupdf.pdf_keystroke_event_change_get,
_mupdf.pdf_keystroke_event_change_set)
4136: (4)               selStart = property(_mupdf.pdf_keystroke_event_selStart_get,
_mupdf.pdf_keystroke_event_selStart_set)
4137: (4)               selEnd = property(_mupdf.pdf_keystroke_event_selEnd_get,
_mupdf.pdf_keystroke_event_selEnd_set)
4138: (4)               willCommit = property(_mupdf.pdf_keystroke_event_willCommit_get,
_mupdf.pdf_keystroke_event_willCommit_set)
4139: (4)               newChange = property(_mupdf.pdf_keystroke_event_newChange_get,
_mupdf.pdf_keystroke_event_newChange_set)
4140: (4)               newValue = property(_mupdf.pdf_keystroke_event_newValue_get,
_mupdf.pdf_keystroke_event_newValue_set)
4141: (4)               def __init__(self):
4142: (8)                   _mupdf.pdf_keystroke_event_swiginit(self,
_mupdf.new_pdf_keystroke_event())
4143: (4)               __swig_destroy__ = _mupdf.delete_pdf_keystroke_event
4144: (0)           _mupdf.pdf_keystroke_event_swigregister(pdf_keystroke_event)
4145: (0)           class pdf_doc_event(object):
4146: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4147: (4)               __repr__ = _swig_repr
4148: (4)               type = property(_mupdf.pdf_doc_event_type_get,
_mupdf.pdf_doc_event_type_set)
4149: (4)               def __init__(self):
4150: (8)                   _mupdf.pdf_doc_event_swiginit(self, _mupdf.new_pdf_doc_event())
4151: (4)               __swig_destroy__ = _mupdf.delete_pdf_doc_event
4152: (0)           _mupdf.pdf_doc_event_swigregister(pdf_doc_event)
4153: (0)           PDF_DOCUMENT_EVENT_ALERT = _mupdf.PDF_DOCUMENT_EVENT_ALERT
4154: (0)           PDF_DOCUMENT_EVENT_PRINT = _mupdf.PDF_DOCUMENT_EVENT_PRINT
4155: (0)           PDF_DOCUMENT_EVENT_LAUNCH_URL = _mupdf.PDF_DOCUMENT_EVENT_LAUNCH_URL
4156: (0)           PDF_DOCUMENT_EVENT_MAIL_DOC = _mupdf.PDF_DOCUMENT_EVENT_MAIL_DOC
4157: (0)           PDF_DOCUMENT_EVENT_SUBMIT = _mupdf.PDF_DOCUMENT_EVENT_SUBMIT
4158: (0)           PDF_DOCUMENT_EVENT_EXEC_MENU_ITEM = _mupdf.PDF_DOCUMENT_EVENT_EXEC_MENU_ITEM
4159: (0)           class pdf_alert_event(object):
4160: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4161: (4)               __repr__ = _swig_repr
4162: (4)               doc = property(_mupdf.pdf_alert_event_doc_get,
_mupdf.pdf_alert_event_doc_set)
4163: (4)               message = property(_mupdf.pdf_alert_event_message_get,
_mupdf.pdf_alert_event_message_set)
4164: (4)               icon_type = property(_mupdf.pdf_alert_event_icon_type_get,
_mupdf.pdf_alert_event_icon_type_set)
4165: (4)               button_group_type = property(_mupdf.pdf_alert_event_button_group_type_get,
_mupdf.pdf_alert_event_button_group_type_set)
4166: (4)               title = property(_mupdf.pdf_alert_event_title_get,
_mupdf.pdf_alert_event_title_set)
4167: (4)               has_check_box = property(_mupdf.pdf_alert_event_has_check_box_get,
_mupdf.pdf_alert_event_has_check_box_set)
4168: (4)               check_box_message = property(_mupdf.pdf_alert_event_check_box_message_get,
_mupdf.pdf_alert_event_check_box_message_set)
4169: (4)               initially_checked = property(_mupdf.pdf_alert_event_initially_checked_get,
_mupdf.pdf_alert_event_initially_checked_set)
4170: (4)               finally_checked = property(_mupdf.pdf_alert_event_finally_checked_get,
_mupdf.pdf_alert_event_finally_checked_set)
4171: (4)               button_pressed = property(_mupdf.pdf_alert_event_button_pressed_get,
_mupdf.pdf_alert_event_button_pressed_set)
4172: (4)               def __init__(self):
4173: (8)                   _mupdf.pdf_alert_event_swiginit(self, _mupdf.new_pdf_alert_event())
4174: (4)               __swig_destroy__ = _mupdf.delete_pdf_alert_event
4175: (0)           _mupdf.pdf_alert_event_swigregister(pdf_alert_event)
4176: (0)           PDF_ALERT_ICON_ERROR = _mupdf.PDF_ALERT_ICON_ERROR
4177: (0)           PDF_ALERT_ICON_WARNING = _mupdf.PDF_ALERT_ICON_WARNING
4178: (0)           PDF_ALERT_ICON_QUESTION = _mupdf.PDF_ALERT_ICON_QUESTION
4179: (0)           PDF_ALERT_ICON_STATUS = _mupdf.PDF_ALERT_ICON_STATUS
```

```
4180: (0)           PDF_ALERT_BUTTON_GROUP_OK = _mupdf.PDF_ALERT_BUTTON_GROUP_OK
4181: (0)           PDF_ALERT_BUTTON_GROUP_OK_CANCEL = _mupdf.PDF_ALERT_BUTTON_GROUP_OK_CANCEL
4182: (0)           PDF_ALERT_BUTTON_GROUP_YES_NO = _mupdf.PDF_ALERT_BUTTON_GROUP_YES_NO
4183: (0)           PDF_ALERT_BUTTON_GROUP_YES_NO_CANCEL =
_mupdf.PDF_ALERT_BUTTON_GROUP_YES_NO_CANCEL
4184: (0)           PDF_ALERT_BUTTON_NONE = _mupdf.PDF_ALERT_BUTTON_NONE
4185: (0)           PDF_ALERT_BUTTON_OK = _mupdf.PDF_ALERT_BUTTON_OK
4186: (0)           PDF_ALERT_BUTTON_CANCEL = _mupdf.PDF_ALERT_BUTTON_CANCEL
4187: (0)           PDF_ALERT_BUTTON_NO = _mupdf.PDF_ALERT_BUTTON_NO
4188: (0)           PDF_ALERT_BUTTON_YES = _mupdf.PDF_ALERT_BUTTON_YES
4189: (0)           class pdf_launch_url_event(object):
4190: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4191: (4)               __repr__ = _swig_repr
4192: (4)               url = property(_mupdf.pdf_launch_url_event_url_get,
_mupdf.pdf_launch_url_event_url_set)
4193: (4)               new_frame = property(_mupdf.pdf_launch_url_event_new_frame_get,
_mupdf.pdf_launch_url_event_new_frame_set)
4194: (4)               def __init__(self):
4195: (8)                   _mupdf.pdf_launch_url_event_swiginit(self,
_mupdf.new_pdf_launch_url_event())
4196: (4)               __swig_destroy__ = _mupdf.delete_pdf_launch_url_event
4197: (0)           _mupdf.pdf_launch_url_event_swigregister(pdf_launch_url_event)
4198: (0)           class pdf_mail_doc_event(object):
4199: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4200: (4)               __repr__ = _swig_repr
4201: (4)               ask_user = property(_mupdf.pdf_mail_doc_event_ask_user_get,
_mupdf.pdf_mail_doc_event_ask_user_set)
4202: (4)               to = property(_mupdf.pdf_mail_doc_event_to_get,
_mupdf.pdf_mail_doc_event_to_set)
4203: (4)               cc = property(_mupdf.pdf_mail_doc_event_cc_get,
_mupdf.pdf_mail_doc_event_cc_set)
4204: (4)               bcc = property(_mupdf.pdf_mail_doc_event_bcc_get,
_mupdf.pdf_mail_doc_event_bcc_set)
4205: (4)               subject = property(_mupdf.pdf_mail_doc_event_subject_get,
_mupdf.pdf_mail_doc_event_subject_set)
4206: (4)               message = property(_mupdf.pdf_mail_doc_event_message_get,
_mupdf.pdf_mail_doc_event_message_set)
4207: (4)               def __init__(self):
4208: (8)                   _mupdf.pdf_mail_doc_event_swiginit(self,
_mupdf.new_pdf_mail_doc_event())
4209: (4)               __swig_destroy__ = _mupdf.delete_pdf_mail_doc_event
4210: (0)           _mupdf.pdf_mail_doc_event_swigregister(pdf_mail_doc_event)
4211: (0)           FZ_SUBSAMPLE_AVERAGE = _mupdf.FZ_SUBSAMPLE_AVERAGE
4212: (0)           FZ_SUBSAMPLE_BICUBIC = _mupdf.FZ_SUBSAMPLE_BICUBIC
4213: (0)           FZ_RECOMPRESS_NEVER = _mupdf.FZ_RECOMPRESS_NEVER
4214: (0)           FZ_RECOMPRESS_SAME = _mupdf.FZ_RECOMPRESS_SAME
4215: (0)           FZ_RECOMPRESS_LOSSLESS = _mupdf.FZ_RECOMPRESS_LOSSLESS
4216: (0)           FZ_RECOMPRESS_JPEG = _mupdf.FZ_RECOMPRESS_JPEG
4217: (0)           FZ_RECOMPRESS_J2K = _mupdf.FZ_RECOMPRESS_J2K
4218: (0)           FZ_RECOMPRESS_FAX = _mupdf.FZ_RECOMPRESS_FAX
4219: (0)           class pdf_image_rewriter_options(object):
4220: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4221: (4)               __repr__ = _swig_repr
4222: (4)               color_lossless_image_subsample_method =
property(_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_method_get,
_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_method_set)
4223: (4)               color_lossy_image_subsample_method =
property(_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_method_get,
_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_method_set)
4224: (4)               color_lossless_image_subsample_threshold =
property(_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_threshold_get,
_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_threshold_set)
4225: (4)               color_lossless_image_subsample_to =
property(_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_to_get,
_mupdf.pdf_image_rewriter_options_color_lossless_image_subsample_to_set)
4226: (4)               color_lossy_image_subsample_threshold =
```

```
property(_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_threshold_get,
_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_threshold_set)
4227: (4)               color_lossy_image_subsample_to =
property(_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_to_get,
_mupdf.pdf_image_rewriter_options_color_lossy_image_subsample_to_set)
4228: (4)               color_lossless_image_recompress_method =
property(_mupdf.pdf_image_rewriter_options_color_lossless_image_recompress_method_get,
_mupdf.pdf_image_rewriter_options_color_lossless_image_recompress_method_set)
4229: (4)               color_lossy_image_recompress_method =
property(_mupdf.pdf_image_rewriter_options_color_lossy_image_recompress_method_get,
_mupdf.pdf_image_rewriter_options_color_lossy_image_recompress_method_set)
4230: (4)               color_lossy_image_recompress_quality =
property(_mupdf.pdf_image_rewriter_options_color_lossy_image_recompress_quality_get,
_mupdf.pdf_image_rewriter_options_color_lossy_image_recompress_quality_set)
4231: (4)               color_lossless_image_recompress_quality =
property(_mupdf.pdf_image_rewriter_options_color_lossless_image_recompress_quality_get,
_mupdf.pdf_image_rewriter_options_color_lossless_image_recompress_quality_set)
4232: (4)               gray_lossless_image_subsample_method =
property(_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_method_get,
_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_method_set)
4233: (4)               gray_lossy_image_subsample_method =
property(_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_method_get,
_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_method_set)
4234: (4)               gray_lossless_image_subsample_threshold =
property(_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_threshold_get,
_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_threshold_set)
4235: (4)               gray_lossless_image_subsample_to =
property(_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_to_get,
_mupdf.pdf_image_rewriter_options_gray_lossless_image_subsample_to_set)
4236: (4)               gray_lossy_image_subsample_threshold =
property(_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_threshold_get,
_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_threshold_set)
4237: (4)               gray_lossy_image_subsample_to =
property(_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_to_get,
_mupdf.pdf_image_rewriter_options_gray_lossy_image_subsample_to_set)
4238: (4)               gray_lossless_image_recompress_method =
property(_mupdf.pdf_image_rewriter_options_gray_lossless_image_recompress_method_get,
_mupdf.pdf_image_rewriter_options_gray_lossless_image_recompress_method_set)
4239: (4)               gray_lossy_image_recompress_method =
property(_mupdf.pdf_image_rewriter_options_gray_lossy_image_recompress_method_get,
_mupdf.pdf_image_rewriter_options_gray_lossy_image_recompress_method_set)
4240: (4)               gray_lossy_image_recompress_quality =
property(_mupdf.pdf_image_rewriter_options_gray_lossy_image_recompress_quality_get,
_mupdf.pdf_image_rewriter_options_gray_lossy_image_recompress_quality_set)
4241: (4)               gray_lossless_image_recompress_quality =
property(_mupdf.pdf_image_rewriter_options_gray_lossless_image_recompress_quality_get,
_mupdf.pdf_image_rewriter_options_gray_lossless_image_recompress_quality_set)
4242: (4)               bitonal_image_subsample_method =
property(_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_method_get,
_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_method_set)
4243: (4)               bitonal_image_subsample_threshold =
property(_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_threshold_get,
_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_threshold_set)
4244: (4)               bitonal_image_subsample_to =
property(_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_to_get,
_mupdf.pdf_image_rewriter_options_bitonal_image_subsample_to_set)
4245: (4)               bitonal_image_recompress_method =
property(_mupdf.pdf_image_rewriter_options_bitonal_image_recompress_method_get,
_mupdf.pdf_image_rewriter_options_bitonal_image_recompress_method_set)
4246: (4)               bitonal_image_recompress_quality =
property(_mupdf.pdf_image_rewriter_options_bitonal_image_recompress_quality_get,
_mupdf.pdf_image_rewriter_options_bitonal_image_recompress_quality_set)
4247: (4)               def __init__(self):
4248: (8)                   _mupdf.pdf_image_rewriter_options_swiginit(self,
_mupdf.new_pdf_image_rewriter_options())
4249: (4)               __swig_destroy__ = _mupdf.delete_pdf_image_rewriter_options
4250: (0)           _mupdf.pdf_image_rewriter_options_swigregister(pdf_image_rewriter_options)
4251: (0)           class pdf_clean_options(object):
4252: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
4253: (4)               __repr__ = _swig_repr
4254: (4)               write = property(_mupdf.pdf_clean_options_write_get,
_mupdf.pdf_clean_options_write_set)
4255: (4)               image = property(_mupdf.pdf_clean_options_image_get,
_mupdf.pdf_clean_options_image_set)
4256: (4)               subset_fonts = property(_mupdf.pdf_clean_options_subset_fonts_get,
_mupdf.pdf_clean_options_subset_fonts_set)
4257: (4)               def __init__(self):
4258: (8)                   _mupdf.pdf_clean_options_swiginit(self,
_mupdf.new_pdf_clean_options())
4259: (4)               __swig_destroy__ = _mupdf.delete_pdf_clean_options
4260: (0)           _mupdf.pdf_clean_options_swigregister(pdf_clean_options)
4261: (0)           class fz_search_page2_hit(object):
4262: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4263: (4)               __repr__ = _swig_repr
4264: (4)               quad = property(_mupdf.fz_search_page2_hit_quad_get,
_mupdf.fz_search_page2_hit_quad_set)
4265: (4)               mark = property(_mupdf.fz_search_page2_hit_mark_get,
_mupdf.fz_search_page2_hit_mark_set)
4266: (4)               def __init__(self):
4267: (8)                   _mupdf.fz_search_page2_hit_swiginit(self,
_mupdf.new_fz_search_page2_hit())
4268: (4)               __swig_destroy__ = _mupdf.delete_fz_search_page2_hit
4269: (0)           _mupdf.fz_search_page2_hit_swigregister(fz_search_page2_hit)
4270: (0)           class fz_install_load_system_font_funcs_args(object):
4271: (4)               r"""
4272: (4)               Extra struct containing fz_install_load_system_font_funcs()'s args,
4273: (4)               which we wrap with virtual_fnptrs set to allow use from Python/C# via
4274: (4)               Swig Directors.
4275: (4)               """
4276: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
4277: (4)               __repr__ = _swig_repr
4278: (4)               f = property(_mupdf.fz_install_load_system_font_funcs_args_f_get,
_mupdf.fz_install_load_system_font_funcs_args_f_set)
4279: (4)               f_cjk = property(_mupdf.fz_install_load_system_font_funcs_args_f_cjk_get,
_mupdf.fz_install_load_system_font_funcs_args_f_cjk_set)
4280: (4)               f_fallback =
property(_mupdf.fz_install_load_system_font_funcs_args_f_fallback_get,
_mupdf.fz_install_load_system_font_funcs_args_f_fallback_set)
4281: (4)               def __init__(self):
4282: (8)                   _mupdf.fz_install_load_system_font_funcs_args_swiginit(self,
_mupdf.new_fz_install_load_system_font_funcs_args())
4283: (4)               __swig_destroy__ = _mupdf.delete_fz_install_load_system_font_funcs_args
4284: (0)
_mupdf.fz_install_load_system_font_funcs_args_swigregister(fz_install_load_system_font_funcs_args)
4285: (0)           def ll_fz_aa_level():
4286: (4)               r"""
4287: (4)               Low-level wrapper for `::fz_aa_level()`.
4288: (4)               Get the number of bits of antialiasing we are
4289: (4)               using (for graphics). Between 0 and 8.
4290: (4)               """
4291: (4)               return _mupdf.ll_fz_aa_level()
4292: (0)           def ll_fz_abs(f):
4293: (4)               r"""
4294: (4)               Low-level wrapper for `::fz_abs()`.
4295: (4)               Some standard math functions, done as static inlines for speed.
4296: (4)               People with compilers that do not adequately implement inline
4297: (4)               may like to reimplement these using macros.
4298: (4)               """
4299: (4)               return _mupdf.ll_fz_abs(f)
4300: (0)           def ll_fz_absi(i):
4301: (4)               r""" Low-level wrapper for `::fz_absi()`."""
4302: (4)               return _mupdf.ll_fz_absi(i)
4303: (0)           def ll_fz_add_layout_char(block, x, w, p):
4304: (4)               r"""
4305: (4)               Low-level wrapper for `::fz_add_layout_char()`.
```

```
4306: (4)               Add a new char to the line at the end of the layout block.
4307: (4)               """
4308: (4)               return _mupdf.ll_fz_add_layout_char(block, x, w, p)
4309: (0)           def ll_fz_add_layout_line(block, x, y, h, p):
4310: (4)               r"""
4311: (4)               Low-level wrapper for `::fz_add_layout_line()`.
4312: (4)               Add a new line to the end of the layout block.
4313: (4)               """
4314: (4)               return _mupdf.ll_fz_add_layout_line(block, x, y, h, p)
4315: (0)           def ll_fz_add_separation(sep, name, cs, cs_channel):
4316: (4)               r"""
4317: (4)               Low-level wrapper for `::fz_add_separation()`.
4318: (4)               Add a separation (null terminated name, colorspace)
4319: (4)               """
4320: (4)               return _mupdf.ll_fz_add_separation(sep, name, cs, cs_channel)
4321: (0)           def ll_fz_add_separation_equivalents(sep, rgba, cmyk, name):
4322: (4)               r"""
4323: (4)               Low-level wrapper for `::fz_add_separation_equivalents()`.
4324: (4)               Add a separation with equivalents (null terminated name,
4325: (4)               colorspace)
4326: (4)               (old, deprecated)
4327: (4)               """
4328: (4)               return _mupdf.ll_fz_add_separation_equivalents(sep, rgba, cmyk, name)
4329: (0)           def ll_fz_adjust_rect_for_stroke(rect, stroke, ctm):
4330: (4)               r"""
4331: (4)               Low-level wrapper for `::fz_adjust_rect_for_stroke()`.
4332: (4)               Given a rectangle (assumed to be the bounding box for a path),
4333: (4)               expand it to allow for the expansion of the bbox that would be
4334: (4)               seen by stroking the path with the given stroke state and
4335: (4)               transform.
4336: (4)               """
4337: (4)               return _mupdf.ll_fz_adjust_rect_for_stroke(rect, stroke, ctm)
4338: (0)           def ll_fz_advance_glyph(font, glyph, wmode):
4339: (4)               r"""
4340: (4)               Low-level wrapper for `::fz_advance_glyph()`.
4341: (4)               Return the advance for a given glyph.
4342: (4)               font: The font to look for the glyph in.
4343: (4)               glyph: The glyph to find the advance for.
4344: (4)               wmode: 1 for vertical mode, 0 for horizontal.
4345: (4)               Returns the advance for the glyph.
4346: (4)               """
4347: (4)               return _mupdf.ll_fz_advance_glyph(font, glyph, wmode)
4348: (0)           def ll_fz_aes_crypt_cbc(ctx, mode, length, iv, input, output):
4349: (4)               r"""
4350: (4)               Low-level wrapper for `::fz_aes_crypt_cbc()`.
4351: (4)               AES block processing. Encrypts or Decrypts (according to mode,
4352: (4)               which must match what was initially set up) length bytes (which
4353: (4)               must be a multiple of 16), using (and modifying) the insertion
4354: (4)               vector iv, reading from input, and writing to output.
4355: (4)               Never throws an exception.
4356: (4)               """
4357: (4)               return _mupdf.ll_fz_aes_crypt_cbc(ctx, mode, length, iv, input, output)
4358: (0)           def ll_fz_aes_setkey_dec(ctx, key, keysize):
4359: (4)               r"""
4360: (4)               Low-level wrapper for `::fz_aes_setkey_dec()`.
4361: (4)               AES decryption intialisation. Fills in the supplied context
4362: (4)               and prepares for decryption using the given key.
4363: (4)               Returns non-zero for error (key size other than 128/192/256).
4364: (4)               Never throws an exception.
4365: (4)               """
4366: (4)               return _mupdf.ll_fz_aes_setkey_dec(ctx, key, keysize)
4367: (0)           def ll_fz_aes_setkey_enc(ctx, key, keysize):
4368: (4)               r"""
4369: (4)               Low-level wrapper for `::fz_aes_setkey_enc()`.
4370: (4)               AES encryption intialisation. Fills in the supplied context
4371: (4)               and prepares for encryption using the given key.
4372: (4)               Returns non-zero for error (key size other than 128/192/256).
4373: (4)               Never throws an exception.
4374: (4)               """
```

```
4375: (4)                return _mupdf.ll_fz_aes_setkey_enc(ctx, key, keysize)
4376: (0)            def ll_fz_alpha_from_gray(gray):
4377: (4)                r""" Low-level wrapper for `::fz_alpha_from_gray()`."""
4378: (4)                return _mupdf.ll_fz_alpha_from_gray(gray)
4379: (0)            def ll_fz_append_base64(out, data, size, newline):
4380: (4)                r"""
4381: (4)                Low-level wrapper for `::fz_append_base64()`.
4382: (4)                Write a base64 encoded data block, optionally with periodic newlines.
4383: (4)                """
4384: (4)                return _mupdf.ll_fz_append_base64(out, data, size, newline)
4385: (0)            def ll_fz_append_base64_buffer(out, data, newline):
4386: (4)                r"""
4387: (4)                Low-level wrapper for `::fz_append_base64_buffer()`.
4388: (4)                Append a base64 encoded fz_buffer, optionally with periodic newlines.
4389: (4)                """
4390: (4)                return _mupdf.ll_fz_append_base64_buffer(out, data, newline)
4391: (0)            def ll_fz_append_bits(buf, value, count):
4392: (4)                r""" Low-level wrapper for `::fz_append_bits()`."""
4393: (4)                return _mupdf.ll_fz_append_bits(buf, value, count)
4394: (0)            def ll_fz_append_bits_pad(buf):
4395: (4)                r""" Low-level wrapper for `::fz_append_bits_pad()`."""
4396: (4)                return _mupdf.ll_fz_append_bits_pad(buf)
4397: (0)            def ll_fz_append_buffer(destination, source):
4398: (4)                r"""
4399: (4)                Low-level wrapper for `::fz_append_buffer()`.
4400: (4)                Append the contents of the source buffer onto the end of the
4401: (4)                destination buffer, extending automatically as required.
4402: (4)                Ownership of buffers does not change.
4403: (4)                """
4404: (4)                return _mupdf.ll_fz_append_buffer(destination, source)
4405: (0)            def ll_fz_append_byte(buf, c):
4406: (4)                r""" Low-level wrapper for `::fz_append_byte()`."""
4407: (4)                return _mupdf.ll_fz_append_byte(buf, c)
4408: (0)            def ll_fz_append_data(buf, data, len):
4409: (4)                r"""
4410: (4)                Low-level wrapper for `::fz_append_data()`.
4411: (4)                fz_append_*: Append data to a buffer.
4412: (4)                The buffer will automatically grow as required.
4413: (4)                """
4414: (4)                return _mupdf.ll_fz_append_data(buf, data, len)
4415: (0)            def ll_fz_append_image_as_data_uri(out, image):
4416: (4)                r""" Low-level wrapper for `::fz_append_image_as_data_uri()`."""
4417: (4)                return _mupdf.ll_fz_append_image_as_data_uri(out, image)
4418: (0)            def ll_fz_append_int16_be(buf, x):
4419: (4)                r""" Low-level wrapper for `::fz_append_int16_be()`."""
4420: (4)                return _mupdf.ll_fz_append_int16_be(buf, x)
4421: (0)            def ll_fz_append_int16_le(buf, x):
4422: (4)                r""" Low-level wrapper for `::fz_append_int16_le()`."""
4423: (4)                return _mupdf.ll_fz_append_int16_le(buf, x)
4424: (0)            def ll_fz_append_int32_be(buf, x):
4425: (4)                r""" Low-level wrapper for `::fz_append_int32_be()`."""
4426: (4)                return _mupdf.ll_fz_append_int32_be(buf, x)
4427: (0)            def ll_fz_append_int32_le(buf, x):
4428: (4)                r""" Low-level wrapper for `::fz_append_int32_le()`."""
4429: (4)                return _mupdf.ll_fz_append_int32_le(buf, x)
4430: (0)            def ll_fz_append_pdf_string(buffer, text):
4431: (4)                r"""
4432: (4)                Low-level wrapper for `::fz_append_pdf_string()`.
4433: (4)                fz_append_pdf_string: Append a string with PDF syntax quotes and
4434: (4)                escapes.
4435: (4)                The buffer will automatically grow as required.
4436: (4)                """
4437: (4)                return _mupdf.ll_fz_append_pdf_string(buffer, text)
4438: (0)            def ll_fz_append_pixmap_as_data_uri(out, pixmap):
4439: (4)                r""" Low-level wrapper for `::fz_append_pixmap_as_data_uri()`."""
4440: (4)                return _mupdf.ll_fz_append_pixmap_as_data_uri(out, pixmap)
4441: (0)            def ll_fz_append_rune(buf, c):
4442: (4)                r""" Low-level wrapper for `::fz_append_rune()`."""
4443: (4)                return _mupdf.ll_fz_append_rune(buf, c)
```

```
4444: (0)           def ll_fz_append_string(buf, data):
4445: (4)               r""" Low-level wrapper for `::fz_append_string()`."""
4446: (4)               return _mupdf.ll_fz_append_string(buf, data)
4447: (0)           def ll_fz_arc4_encrypt(state, dest, src, len):
4448: (4)               r"""
4449: (4)               Low-level wrapper for `::fz_arc4_encrypt()`.
4450: (4)               RC4 block encrypt operation; encrypt src into dst (both of
4451: (4)               length len) updating the RC4 state as we go.
4452: (4)               Never throws an exception.
4453: (4)               """
4454: (4)               return _mupdf.ll_fz_arc4_encrypt(state, dest, src, len)
4455: (0)           def ll_fz_arc4_final(state):
4456: (4)               r"""
4457: (4)               Low-level wrapper for `::fz_arc4_final()`.
4458: (4)               RC4 finalization. Zero the context.
4459: (4)               Never throws an exception.
4460: (4)               """
4461: (4)               return _mupdf.ll_fz_arc4_final(state)
4462: (0)           def ll_fz_arc4_init(state, key, len):
4463: (4)               r"""
4464: (4)               Low-level wrapper for `::fz_arc4_init()`.
4465: (4)               RC4 initialization. Begins an RC4 operation, writing a new
4466: (4)               context.
4467: (4)               Never throws an exception.
4468: (4)               """
4469: (4)               return _mupdf.ll_fz_arc4_init(state, key, len)
4470: (0)           def ll_fz_archive_format(arch):
4471: (4)               r"""
4472: (4)               Low-level wrapper for `::fz_archive_format()`.
4473: (4)               Return a pointer to a string describing the format of the
4474: (4)               archive.
4475: (4)               The lifetime of the string is unspecified (in current
4476: (4)               implementations the string will persist until the archive
4477: (4)               is closed, but this is not guaranteed).
4478: (4)               """
4479: (4)               return _mupdf.ll_fz_archive_format(arch)
4480: (0)           def ll_fz_atof(s):
4481: (4)               r"""
4482: (4)               Low-level wrapper for `::fz_atof()`.
4483: (4)               Range checking atof
4484: (4)               """
4485: (4)               return _mupdf.ll_fz_atof(s)
4486: (0)           def ll_fz_atoi(s):
4487: (4)               r"""
4488: (4)               Low-level wrapper for `::fz_atoi()`.
4489: (4)               atoi that copes with NULL
4490: (4)               """
4491: (4)               return _mupdf.ll_fz_atoi(s)
4492: (0)           def ll_fz_atoi64(s):
4493: (4)               r"""
4494: (4)               Low-level wrapper for `::fz_atoi64()`.
4495: (4)               64bit atoi that copes with NULL
4496: (4)               """
4497: (4)               return _mupdf.ll_fz_atoi64(s)
4498: (0)           def ll_fz_authenticate_password(doc, password):
4499: (4)               r"""
4500: (4)               Low-level wrapper for `::fz_authenticate_password()`.
4501: (4)               Test if the given password can decrypt the document.
4502: (4)               password: The password string to be checked. Some document
4503: (4)               specifications do not specify any particular text encoding, so
4504: (4)               neither do we.
4505: (4)               Returns 0 for failure to authenticate, non-zero for success.
4506: (4)               For PDF documents, further information can be given by examining
4507: (4)               the bits in the return code.
4508: (8)                   Bit 0 => No password required
4509: (8)                   Bit 1 => User password authenticated
4510: (8)                   Bit 2 => Owner password authenticated
4511: (4)               """
4512: (4)               return _mupdf.ll_fz_authenticate_password(doc, password)
```

```
4513: (0)           def ll_fz_available(stm, max):
4514: (4)               r"""
4515: (4)               Low-level wrapper for `::fz_available()`.
4516: (4)               Ask how many bytes are available immediately from
4517: (4)               a given stream.
4518: (4)               stm: The stream to read from.
4519: (4)               max: A hint for the underlying stream; the maximum number of
4520: (4)               bytes that we are sure we will want to read. If you do not know
4521: (4)               this number, give 1.
4522: (4)               Returns the number of bytes immediately available between the
4523: (4)               read and write pointers. This number is guaranteed only to be 0
4524: (4)               if we have hit EOF. The number of bytes returned here need have
4525: (4)               no relation to max (could be larger, could be smaller).
4526: (4)               """
4527: (4)               return _mupdf.ll_fz_available(stm, max)
4528: (0)           def ll_fz_base_colorspace(cs):
4529: (4)               r"""
4530: (4)               Low-level wrapper for `::fz_base_colorspace()`.
4531: (4)               Get the 'base' colorspace for a colorspace.
4532: (4)               For indexed colorspaces, this is the colorspace the index
4533: (4)               decodes into. For all other colorspaces, it is the colorspace
4534: (4)               itself.
4535: (4)               The returned colorspace is 'borrowed' (i.e. no additional
4536: (4)               references are taken or dropped).
4537: (4)               """
4538: (4)               return _mupdf.ll_fz_base_colorspace(cs)
4539: (0)           def ll_fz_basename(path):
4540: (4)               r"""
4541: (4)               Low-level wrapper for `::fz_basename()`.
4542: (4)               Find the filename component in a path.
4543: (4)               """
4544: (4)               return _mupdf.ll_fz_basename(path)
4545: (0)           def ll_fz_begin_group(dev, area, cs, isolated, knockout, blendmode, alpha):
4546: (4)               r""" Low-level wrapper for `::fz_begin_group()`."""
4547: (4)               return _mupdf.ll_fz_begin_group(dev, area, cs, isolated, knockout,
blendmode, alpha)
4548: (0)           def ll_fz_begin_layer(dev, layer_name):
4549: (4)               r""" Low-level wrapper for `::fz_begin_layer()`."""
4550: (4)               return _mupdf.ll_fz_begin_layer(dev, layer_name)
4551: (0)           def ll_fz_begin_mask(dev, area, luminosity, colorspace, bc, color_params):
4552: (4)               r""" Low-level wrapper for `::fz_begin_mask()`."""
4553: (4)               return _mupdf.ll_fz_begin_mask(dev, area, luminosity, colorspace, bc,
color_params)
4554: (0)           def ll_fz_begin_metatext(dev, meta, text):
4555: (4)               r""" Low-level wrapper for `::fz_begin_metatext()`."""
4556: (4)               return _mupdf.ll_fz_begin_metatext(dev, meta, text)
4557: (0)           def ll_fz_begin_page(wri, mediabox):
4558: (4)               r"""
4559: (4)               Low-level wrapper for `::fz_begin_page()`.
4560: (4)               Called to start the process of writing a page to
4561: (4)               a document.
4562: (4)               mediabox: page size rectangle in points.
4563: (4)               Returns a borrowed fz_device to write page contents to. This
4564: (4)               should be kept if required, and only dropped if it was kept.
4565: (4)               """
4566: (4)               return _mupdf.ll_fz_begin_page(wri, mediabox)
4567: (0)           def ll_fz_begin_structure(dev, standard, raw, idx):
4568: (4)               r""" Low-level wrapper for `::fz_begin_structure()`."""
4569: (4)               return _mupdf.ll_fz_begin_structure(dev, standard, raw, idx)
4570: (0)           def ll_fz_begin_tile(dev, area, view, xstep, ystep, ctm):
4571: (4)               r""" Low-level wrapper for `::fz_begin_tile()`."""
4572: (4)               return _mupdf.ll_fz_begin_tile(dev, area, view, xstep, ystep, ctm)
4573: (0)           def ll_fz_begin_tile_id(dev, area, view, xstep, ystep, ctm, id):
4574: (4)               r""" Low-level wrapper for `::fz_begin_tile_id()`."""
4575: (4)               return _mupdf.ll_fz_begin_tile_id(dev, area, view, xstep, ystep, ctm, id)
4576: (0)           def ll_fz_bidi_fragment_text(text, textlen, baseDir, callback, arg, flags):
4577: (4)               r"""
4578: (5)                Low-level wrapper for `::fz_bidi_fragment_text()`.
4579: (4)               This function has out-params. Python/C# wrappers look like:
```

```
4580: (8)                   `ll_fz_bidi_fragment_text(const uint32_t *text, size_t textlen,
::fz_bidi_fragment_fn *callback, void *arg, int flags)` => ::fz_bidi_direction baseDir
4581: (8)                   Partitions the given Unicode sequence into one or more
4582: (8)                   unidirectional fragments and invokes the given callback
4583: (8)                   function for each fragment.
4584: (8)                   For example, if directionality of text is:
4585: (16)                          0123456789
4586: (16)                          rrlllrrrrr,
4587: (8)                   we'll invoke callback with:
4588: (16)                          &text[0], length == 2
4589: (16)                          &text[2], length == 3
4590: (16)                          &text[5], length == 5
4591: (8)                   :type text: int, in
4592: (4)               :param text:    start of Unicode sequence
4593: (12)                      :type textlen: int, in
4594: (4)               :param textlen:   number of Unicodes to analyse
4595: (12)                      :type baseDir: int, in
4596: (4)               :param baseDir:   direction of paragraph (specify FZ_BIDI_NEUTRAL to force
auto-detection)
4597: (12)                      :type callback: ::fz_bidi_fragment_fn, in
4598: (4)               :param callback:  function to be called for each fragment
4599: (12)                      :type arg: void, in
4600: (4)               :param arg:    data to be passed to the callback function
4601: (12)                      :type flags: int, in
4602: (4)               :param flags:     flags to control operation (see fz_bidi_flags above)
4603: (4)               """
4604: (4)               return _mupdf.ll_fz_bidi_fragment_text(text, textlen, baseDir, callback,
arg, flags)
4605: (0)           def ll_fz_bitmap_details(bitmap, w, h, n, stride):
4606: (4)               r"""
4607: (5)                Low-level wrapper for `::fz_bitmap_details()`.
4608: (4)               This function has out-params. Python/C# wrappers look like:
4609: (8)                   `ll_fz_bitmap_details(::fz_bitmap *bitmap)` => `(int w, int h, int n,
int stride)`
4610: (8)                   Retrieve details of a given bitmap.
4611: (8)                   bitmap: The bitmap to query.
4612: (8)                   w: Pointer to storage to retrieve width (or NULL).
4613: (8)                   h: Pointer to storage to retrieve height (or NULL).
4614: (8)                   n: Pointer to storage to retrieve number of color components (or
4615: (8)                   NULL).
4616: (8)                   stride: Pointer to storage to retrieve bitmap stride (or NULL).
4617: (4)               """
4618: (4)               return _mupdf.ll_fz_bitmap_details(bitmap, w, h, n, stride)
4619: (0)           def ll_fz_blendmode_name(blendmode):
4620: (4)               r"""
4621: (4)               Low-level wrapper for `::fz_blendmode_name()`.
4622: (4)               Map from enumeration to blend mode string.
4623: (4)               The string is static, with arbitrary lifespan.
4624: (4)               """
4625: (4)               return _mupdf.ll_fz_blendmode_name(blendmode)
4626: (0)           def ll_fz_bound_display_list(list):
4627: (4)               r"""
4628: (4)               Low-level wrapper for `::fz_bound_display_list()`.
4629: (4)               Return the bounding box of the page recorded in a display list.
4630: (4)               """
4631: (4)               return _mupdf.ll_fz_bound_display_list(list)
4632: (0)           def ll_fz_bound_glyph(font, gid, trm):
4633: (4)               r"""
4634: (4)               Low-level wrapper for `::fz_bound_glyph()`.
4635: (4)               Return a bbox for a given glyph in a font.
4636: (4)               font: The font to look for the glyph in.
4637: (4)               gid: The glyph to bound.
4638: (4)               trm: The matrix to apply to the glyph before bounding.
4639: (4)               Returns rectangle by value containing the bounds of the given
4640: (4)               glyph.
4641: (4)               """
4642: (4)               return _mupdf.ll_fz_bound_glyph(font, gid, trm)
4643: (0)           def ll_fz_bound_page(page):
4644: (4)               r"""
```

```
4645: (4)               Low-level wrapper for `::fz_bound_page()`.
4646: (4)               Determine the size of a page at 72 dpi.
4647: (4)               """
4648: (4)               return _mupdf.ll_fz_bound_page(page)
4649: (0)           def ll_fz_bound_page_box(page, box):
4650: (4)               r""" Low-level wrapper for `::fz_bound_page_box()`."""
4651: (4)               return _mupdf.ll_fz_bound_page_box(page, box)
4652: (0)           def ll_fz_bound_path(path, stroke, ctm):
4653: (4)               r"""
4654: (4)               Low-level wrapper for `::fz_bound_path()`.
4655: (4)               Return a bounding rectangle for a path.
4656: (4)               path: The path to bound.
4657: (4)               stroke: If NULL, the bounding rectangle given is for
4658: (4)               the filled path. If non-NULL the bounding rectangle
4659: (4)               given is for the path stroked with the given attributes.
4660: (4)               ctm: The matrix to apply to the path during stroking.
4661: (4)               r: Pointer to a fz_rect which will be used to hold
4662: (4)               the result.
4663: (4)               Returns r, updated to contain the bounding rectangle.
4664: (4)               """
4665: (4)               return _mupdf.ll_fz_bound_path(path, stroke, ctm)
4666: (0)           def ll_fz_bound_shade(shade, ctm):
4667: (4)               r"""
4668: (4)               Low-level wrapper for `::fz_bound_shade()`.
4669: (4)               Bound a given shading.
4670: (4)               shade: The shade to bound.
4671: (4)               ctm: The transform to apply to the shade before bounding.
4672: (4)               r: Pointer to storage to put the bounds in.
4673: (4)               Returns r, updated to contain the bounds for the shading.
4674: (4)               """
4675: (4)               return _mupdf.ll_fz_bound_shade(shade, ctm)
4676: (0)           def ll_fz_bound_text(text, stroke, ctm):
4677: (4)               r"""
4678: (4)               Low-level wrapper for `::fz_bound_text()`.
4679: (4)               Find the bounds of a given text object.
4680: (4)               text: The text object to find the bounds of.
4681: (4)               stroke: Pointer to the stroke attributes (for stroked
4682: (4)               text), or NULL (for filled text).
4683: (4)               ctm: The matrix in use.
4684: (4)               r: pointer to storage for the bounds.
4685: (4)               Returns a pointer to r, which is updated to contain the
4686: (4)               bounding box for the text object.
4687: (4)               """
4688: (4)               return _mupdf.ll_fz_bound_text(text, stroke, ctm)
4689: (0)           def ll_fz_box_type_from_string(name):
4690: (4)               r""" Low-level wrapper for `::fz_box_type_from_string()`."""
4691: (4)               return _mupdf.ll_fz_box_type_from_string(name)
4692: (0)           def ll_fz_buffer_extract(buf, data):
4693: (4)               r"""
4694: (5)                Low-level wrapper for `::fz_buffer_extract()`.
4695: (4)               This function has out-params. Python/C# wrappers look like:
4696: (8)                   `ll_fz_buffer_extract(::fz_buffer *buf)` => `(size_t, unsigned char
*data)`
4697: (8)                   Take ownership of buffer contents.
4698: (8)                   Performs the same task as fz_buffer_storage, but ownership of
4699: (8)                   the data buffer returns with this call. The buffer is left
4700: (8)                   empty.
4701: (8)                   Note: Bad things may happen if this is called on a buffer with
4702: (8)                   multiple references that is being used from multiple threads.
4703: (8)                   data: Pointer to place to retrieve data pointer.
4704: (8)                   Returns length of stream.
4705: (4)               """
4706: (4)               return _mupdf.ll_fz_buffer_extract(buf, data)
4707: (0)           def ll_fz_buffer_storage(buf, datap):
4708: (4)               r"""
4709: (5)                Low-level wrapper for `::fz_buffer_storage()`.
4710: (4)               This function has out-params. Python/C# wrappers look like:
4711: (8)                   `ll_fz_buffer_storage(::fz_buffer *buf)` => `(size_t, unsigned char
*datap)`
```

```
4712: (8)                   Retrieve internal memory of buffer.
4713: (8)                   datap: Output parameter that will be pointed to the data.
4714: (8)                   Returns the current size of the data in bytes.
4715: (4)               """
4716: (4)               return _mupdf.ll_fz_buffer_storage(buf, datap)
4717: (0)           def ll_fz_calloc(count, size):
4718: (4)               r"""
4719: (4)               Low-level wrapper for `::fz_calloc()`.
4720: (4)               Allocate array of memory of count entries of size bytes.
4721: (4)               Clears the memory to zero.
4722: (4)               Throws exception in the event of failure to allocate.
4723: (4)               """
4724: (4)               return _mupdf.ll_fz_calloc(count, size)
4725: (0)           def ll_fz_calloc_no_throw(count, size):
4726: (4)               r"""
4727: (4)               Low-level wrapper for `::fz_calloc_no_throw()`.
4728: (4)               fz_calloc equivalent that returns NULL rather than throwing
4729: (4)               exceptions.
4730: (4)               """
4731: (4)               return _mupdf.ll_fz_calloc_no_throw(count, size)
4732: (0)           def ll_fz_caught():
4733: (4)               r"""
4734: (4)               Low-level wrapper for `::fz_caught()`.
4735: (4)               Within an fz_catch() block, retrieve the error code for
4736: (4)               the current exception.
4737: (4)               This assumes no intervening use of fz_try/fz_catch.
4738: (4)               """
4739: (4)               return _mupdf.ll_fz_caught()
4740: (0)           def ll_fz_caught_errno():
4741: (4)               r""" Low-level wrapper for `::fz_caught_errno()`."""
4742: (4)               return _mupdf.ll_fz_caught_errno()
4743: (0)           def ll_fz_caught_message():
4744: (4)               r"""
4745: (4)               Low-level wrapper for `::fz_caught_message()`.
4746: (4)               Within an fz_catch() block, retrieve the formatted message
4747: (4)               string for the current exception.
4748: (4)               This assumes no intervening use of fz_try/fz_catch.
4749: (4)               """
4750: (4)               return _mupdf.ll_fz_caught_message()
4751: (0)           def ll_fz_chartorune(rune, str):
4752: (4)               r"""
4753: (5)                Low-level wrapper for `::fz_chartorune()`.
4754: (4)               This function has out-params. Python/C# wrappers look like:
4755: (8)                   `ll_fz_chartorune(const char *str)` => `(int, int rune)`
4756: (8)                   UTF8 decode a single rune from a sequence of chars.
4757: (8)                   rune: Pointer to an int to assign the decoded 'rune' to.
4758: (8)                   str: Pointer to a UTF8 encoded string.
4759: (8)                   Returns the number of bytes consumed.
4760: (4)               """
4761: (4)               return _mupdf.ll_fz_chartorune(rune, str)
4762: (0)           def ll_fz_clamp(x, min, max):
4763: (4)               r""" Low-level wrapper for `::fz_clamp()`."""
4764: (4)               return _mupdf.ll_fz_clamp(x, min, max)
4765: (0)           def ll_fz_clamp64(x, min, max):
4766: (4)               r""" Low-level wrapper for `::fz_clamp64()`."""
4767: (4)               return _mupdf.ll_fz_clamp64(x, min, max)
4768: (0)           def ll_fz_clamp_color(cs, _in, out):
4769: (4)               r"""
4770: (5)                Low-level wrapper for `::fz_clamp_color()`.
4771: (4)               This function has out-params. Python/C# wrappers look like:
4772: (8)                   `ll_fz_clamp_color(::fz_colorspace *cs, const float *in)` => float out
4773: (8)                   Clamp the samples in a color to the correct ranges for a
4774: (8)                   given colorspace.
4775: (4)               """
4776: (4)               return _mupdf.ll_fz_clamp_color(cs, _in, out)
4777: (0)           def ll_fz_clamp_location(doc, loc):
4778: (4)               r"""
4779: (4)               Low-level wrapper for `::fz_clamp_location()`.
4780: (4)               Clamps a location into valid chapter/page range. (First clamps
```

```
(4)             the chapter into range, then the page into range).
(4)             """
(4)             return _mupdf.ll_fz_clamp_location(doc, loc)
(0)         def ll_fz_clampd(x, min, max):
(4)             r""" Low-level wrapper for `::fz_clampd()`."""
(4)             return _mupdf.ll_fz_clampd(x, min, max)
(0)         def ll_fz_clampi(x, min, max):
(4)             r""" Low-level wrapper for `::fz_clampi()`."""
(4)             return _mupdf.ll_fz_clampi(x, min, max)
(0)         def ll_fz_clampp(x, min, max):
(4)             r""" Low-level wrapper for `::fz_clampp()`."""
(4)             return _mupdf.ll_fz_clampp(x, min, max)
(0)         def ll_fz_cleanname(name):
(4)             r"""
(4)             Low-level wrapper for `::fz_cleanname()`.
(4)             rewrite path to the shortest string that names the same path.
(4)             Eliminates multiple and trailing slashes, interprets "." and
(4)             "..". Overwrites the string in place.
(4)             """
(4)             return _mupdf.ll_fz_cleanname(name)
(0)         def ll_fz_cleanname_strdup(name):
(4)             r"""
(4)             Low-level wrapper for `::fz_cleanname_strdup()`.
(4)             rewrite path to the shortest string that names the same path.
(4)             Eliminates multiple and trailing slashes, interprets "." and
(4)             "..". Allocates a new string that the caller must free.
(4)             """
(4)             return _mupdf.ll_fz_cleanname_strdup(name)
(0)         def ll_fz_clear_bitmap(bit):
(4)             r"""
(4)             Low-level wrapper for `::fz_clear_bitmap()`.
(4)             Set the entire bitmap to 0.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_clear_bitmap(bit)
(0)         def ll_fz_clear_buffer(buf):
(4)             r"""
(4)             Low-level wrapper for `::fz_clear_buffer()`.
(4)             Empties the buffer. Storage is not freed, but is held ready
(4)             to be reused as the buffer is refilled.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_clear_buffer(buf)
(0)         def ll_fz_clear_pixmap(pix):
(4)             r"""
(4)             Low-level wrapper for `::fz_clear_pixmap()`.
(4)             Sets all components (including alpha) of
(4)             all pixels in a pixmap to 0.
(4)             pix: The pixmap to clear.
(4)             """
(4)             return _mupdf.ll_fz_clear_pixmap(pix)
(0)         def ll_fz_clear_pixmap_rect_with_value(pix, value, r):
(4)             r"""
(4)             Low-level wrapper for `::fz_clear_pixmap_rect_with_value()`.
(4)             Clears a subrect of a pixmap with the given value.
(4)             pix: The pixmap to clear.
(4)             value: Values in the range 0 to 255 are valid. Each component
(4)             sample for each pixel in the pixmap will be set to this value,
(4)             while alpha will always be set to 255 (non-transparent).
(4)             r: the rectangle.
(4)             """
(4)             return _mupdf.ll_fz_clear_pixmap_rect_with_value(pix, value, r)
(0)         def ll_fz_clear_pixmap_with_value(pix, value):
(4)             r"""
(4)             Low-level wrapper for `::fz_clear_pixmap_with_value()`.
(4)             Clears a pixmap with the given value.
(4)             pix: The pixmap to clear.
(4)             value: Values in the range 0 to 255 are valid. Each component
(4)             sample for each pixel in the pixmap will be set to this value,
```

```
4850: (4)               while alpha will always be set to 255 (non-transparent).
4851: (4)               This function is horrible, and should be removed from the
4852: (4)               API and replaced with a less magic one.
4853: (4)               """
4854: (4)               return _mupdf.ll_fz_clear_pixmap_with_value(pix, value)
4855: (0)           def ll_fz_clip_image_mask(dev, image, ctm, scissor):
4856: (4)               r""" Low-level wrapper for `::fz_clip_image_mask()`."""
4857: (4)               return _mupdf.ll_fz_clip_image_mask(dev, image, ctm, scissor)
4858: (0)           def ll_fz_clip_path(dev, path, even_odd, ctm, scissor):
4859: (4)               r""" Low-level wrapper for `::fz_clip_path()`."""
4860: (4)               return _mupdf.ll_fz_clip_path(dev, path, even_odd, ctm, scissor)
4861: (0)           def ll_fz_clip_stroke_path(dev, path, stroke, ctm, scissor):
4862: (4)               r""" Low-level wrapper for `::fz_clip_stroke_path()`."""
4863: (4)               return _mupdf.ll_fz_clip_stroke_path(dev, path, stroke, ctm, scissor)
4864: (0)           def ll_fz_clip_stroke_text(dev, text, stroke, ctm, scissor):
4865: (4)               r""" Low-level wrapper for `::fz_clip_stroke_text()`."""
4866: (4)               return _mupdf.ll_fz_clip_stroke_text(dev, text, stroke, ctm, scissor)
4867: (0)           def ll_fz_clip_text(dev, text, ctm, scissor):
4868: (4)               r""" Low-level wrapper for `::fz_clip_text()`."""
4869: (4)               return _mupdf.ll_fz_clip_text(dev, text, ctm, scissor)
4870: (0)           def ll_fz_clone_buffer(buf):
4871: (4)               r"""
4872: (4)               Low-level wrapper for `::fz_clone_buffer()`.
4873: (4)               Make a new buffer, containing a copy of the data used in
4874: (4)               the original.
4875: (4)               """
4876: (4)               return _mupdf.ll_fz_clone_buffer(buf)
4877: (0)           def ll_fz_clone_context():
4878: (4)               r"""
4879: (4)               Low-level wrapper for `::fz_clone_context()`.
4880: (4)               Make a clone of an existing context.
4881: (4)               This function is meant to be used in multi-threaded
4882: (4)               applications where each thread requires its own context, yet
4883: (4)               parts of the global state, for example caching, are shared.
4884: (4)               ctx: Context obtained from fz_new_context to make a copy of.
4885: (4)               ctx must have had locks and lock/functions setup when created.
4886: (4)               The two contexts will share the memory allocator, resource
4887: (4)               store, locks and lock/unlock functions. They will each have
4888: (4)               their own exception stacks though.
4889: (4)               May return NULL.
4890: (4)               """
4891: (4)               return _mupdf.ll_fz_clone_context()
4892: (0)           def ll_fz_clone_default_colorspaces(base):
4893: (4)               r"""
4894: (4)               Low-level wrapper for `::fz_clone_default_colorspaces()`.
4895: (4)               Returns a reference to a newly cloned default colorspaces
4896: (4)               structure.
4897: (4)               The new clone may safely be altered without fear of race
4898: (4)               conditions as the caller is the only reference holder.
4899: (4)               """
4900: (4)               return _mupdf.ll_fz_clone_default_colorspaces(base)
4901: (0)           def ll_fz_clone_path(path):
4902: (4)               r"""
4903: (4)               Low-level wrapper for `::fz_clone_path()`.
4904: (4)               Clone the data for a path.
4905: (4)               This is used in preference to fz_keep_path when a whole
4906: (4)               new copy of a path is required, rather than just a shared
4907: (4)               pointer. This probably indicates that the path is about to
4908: (4)               be modified.
4909: (4)               path: path to clone.
4910: (4)               Throws exceptions on failure to allocate.
4911: (4)               """
4912: (4)               return _mupdf.ll_fz_clone_path(path)
4913: (0)           def ll_fz_clone_pixmap(old):
4914: (4)               r"""
4915: (4)               Low-level wrapper for `::fz_clone_pixmap()`.
4916: (4)               Clone a pixmap, copying the pixels and associated data to new
4917: (4)               storage.
4918: (4)               The reference count of 'old' is unchanged.
```

```
4919: (4)                """
4920: (4)                return _mupdf.ll_fz_clone_pixmap(old)
4921: (0)            def ll_fz_clone_pixmap_area_with_different_seps(src, bbox, dcs, seps,
color_params, default_cs):
4922: (4)                r""" Low-level wrapper for
`::fz_clone_pixmap_area_with_different_seps()`."""
4923: (4)                return _mupdf.ll_fz_clone_pixmap_area_with_different_seps(src, bbox, dcs,
seps, color_params, default_cs)
4924: (0)            def ll_fz_clone_separations_for_overprint(seps):
4925: (4)                r"""
4926: (4)                Low-level wrapper for `::fz_clone_separations_for_overprint()`.
4927: (4)                Return a separations object with all the spots in the input
4928: (4)                separations object that are set to composite, reset to be
4929: (4)                enabled. If there ARE no spots in the object, this returns
4930: (4)                NULL. If the object already has all its spots enabled, then
4931: (4)                just returns another handle on the same object.
4932: (4)                """
4933: (4)                return _mupdf.ll_fz_clone_separations_for_overprint(seps)
4934: (0)            def ll_fz_clone_stroke_state(stroke):
4935: (4)                r"""
4936: (4)                Low-level wrapper for `::fz_clone_stroke_state()`.
4937: (4)                Create an identical stroke_state structure and return a
4938: (4)                reference to it.
4939: (4)                stroke: The stroke state reference to clone.
4940: (4)                Exceptions may be thrown in the event of a failure to
4941: (4)                allocate.
4942: (4)                """
4943: (4)                return _mupdf.ll_fz_clone_stroke_state(stroke)
4944: (0)            def ll_fz_close_band_writer(writer):
4945: (4)                r"""
4946: (4)                Low-level wrapper for `::fz_close_band_writer()`.
4947: (4)                Finishes up the output and closes the band writer. After this
4948: (4)                call no more headers or bands may be written.
4949: (4)                """
4950: (4)                return _mupdf.ll_fz_close_band_writer(writer)
4951: (0)            def ll_fz_close_device(dev):
4952: (4)                r"""
4953: (4)                Low-level wrapper for `::fz_close_device()`.
4954: (4)                Signal the end of input, and flush any buffered output.
4955: (4)                This is NOT called implicitly on fz_drop_device. This
4956: (4)                may throw exceptions.
4957: (4)                """
4958: (4)                return _mupdf.ll_fz_close_device(dev)
4959: (0)            def ll_fz_close_document_writer(wri):
4960: (4)                r"""
4961: (4)                Low-level wrapper for `::fz_close_document_writer()`.
4962: (4)                Called to end the process of writing
4963: (4)                pages to a document.
4964: (4)                This writes any file level trailers required. After this
4965: (4)                completes successfully the file is up to date and complete.
4966: (4)                """
4967: (4)                return _mupdf.ll_fz_close_document_writer(wri)
4968: (0)            def ll_fz_close_output(arg_1):
4969: (4)                r"""
4970: (4)                Low-level wrapper for `::fz_close_output()`.
4971: (4)                Flush pending output and close an output stream.
4972: (4)                """
4973: (4)                return _mupdf.ll_fz_close_output(arg_1)
4974: (0)            def ll_fz_close_zip_writer(zip):
4975: (4)                r"""
4976: (4)                Low-level wrapper for `::fz_close_zip_writer()`.
4977: (4)                Close the zip file for writing.
4978: (4)                This flushes any pending data to the file. This can throw
4979: (4)                exceptions.
4980: (4)                """
4981: (4)                return _mupdf.ll_fz_close_zip_writer(zip)
4982: (0)            def ll_fz_closepath(path):
4983: (4)                r"""
4984: (4)                Low-level wrapper for `::fz_closepath()`.
```

```
4985: (4)                Close the current subpath.
4986: (4)                path: The path to modify.
4987: (4)                Throws exceptions on failure to allocate, attempting to modify
4988: (4)                a packed path, and illegal path closes (i.e. closing a non open
4989: (4)                path).
4990: (4)                """
4991: (4)                return _mupdf.ll_fz_closepath(path)
4992: (0)            def ll_fz_colorspace_colorant(cs, n):
4993: (4)                r"""
4994: (4)                Low-level wrapper for `::fz_colorspace_colorant()`.
4995: (4)                Retrieve a the name for a colorant.
4996: (4)                Returns a pointer with the same lifespan as the colorspace.
4997: (4)                """
4998: (4)                return _mupdf.ll_fz_colorspace_colorant(cs, n)
4999: (0)            def ll_fz_colorspace_device_n_has_cmyk(cs):
5000: (4)                r"""
5001: (4)                Low-level wrapper for `::fz_colorspace_device_n_has_cmyk()`.
5002: (4)                True if DeviceN color space has cyan magenta yellow or black as
5003: (4)                one of its colorants.
5004: (4)                """
5005: (4)                return _mupdf.ll_fz_colorspace_device_n_has_cmyk(cs)
5006: (0)            def ll_fz_colorspace_device_n_has_only_cmyk(cs):
5007: (4)                r"""
5008: (4)                Low-level wrapper for `::fz_colorspace_device_n_has_only_cmyk()`.
5009: (4)                True if DeviceN color space has only colorants from the CMYK set.
5010: (4)                """
5011: (4)                return _mupdf.ll_fz_colorspace_device_n_has_only_cmyk(cs)
5012: (0)            def ll_fz_colorspace_is_cmyk(cs):
5013: (4)                r""" Low-level wrapper for `::fz_colorspace_is_cmyk()`."""
5014: (4)                return _mupdf.ll_fz_colorspace_is_cmyk(cs)
5015: (0)            def ll_fz_colorspace_is_device(cs):
5016: (4)                r""" Low-level wrapper for `::fz_colorspace_is_device()`."""
5017: (4)                return _mupdf.ll_fz_colorspace_is_device(cs)
5018: (0)            def ll_fz_colorspace_is_device_cmyk(cs):
5019: (4)                r""" Low-level wrapper for `::fz_colorspace_is_device_cmyk()`."""
5020: (4)                return _mupdf.ll_fz_colorspace_is_device_cmyk(cs)
5021: (0)            def ll_fz_colorspace_is_device_gray(cs):
5022: (4)                r""" Low-level wrapper for `::fz_colorspace_is_device_gray()`."""
5023: (4)                return _mupdf.ll_fz_colorspace_is_device_gray(cs)
5024: (0)            def ll_fz_colorspace_is_device_n(cs):
5025: (4)                r""" Low-level wrapper for `::fz_colorspace_is_device_n()`."""
5026: (4)                return _mupdf.ll_fz_colorspace_is_device_n(cs)
5027: (0)            def ll_fz_colorspace_is_gray(cs):
5028: (4)                r"""
5029: (4)                Low-level wrapper for `::fz_colorspace_is_gray()`.
5030: (4)                Tests for particular types of colorspaces
5031: (4)                """
5032: (4)                return _mupdf.ll_fz_colorspace_is_gray(cs)
5033: (0)            def ll_fz_colorspace_is_indexed(cs):
5034: (4)                r""" Low-level wrapper for `::fz_colorspace_is_indexed()`."""
5035: (4)                return _mupdf.ll_fz_colorspace_is_indexed(cs)
5036: (0)            def ll_fz_colorspace_is_lab(cs):
5037: (4)                r""" Low-level wrapper for `::fz_colorspace_is_lab()`."""
5038: (4)                return _mupdf.ll_fz_colorspace_is_lab(cs)
5039: (0)            def ll_fz_colorspace_is_lab_icc(cs):
5040: (4)                r""" Low-level wrapper for `::fz_colorspace_is_lab_icc()`."""
5041: (4)                return _mupdf.ll_fz_colorspace_is_lab_icc(cs)
5042: (0)            def ll_fz_colorspace_is_rgb(cs):
5043: (4)                r""" Low-level wrapper for `::fz_colorspace_is_rgb()`."""
5044: (4)                return _mupdf.ll_fz_colorspace_is_rgb(cs)
5045: (0)            def ll_fz_colorspace_is_subtractive(cs):
5046: (4)                r"""
5047: (4)                Low-level wrapper for `::fz_colorspace_is_subtractive()`.
5048: (4)                True for CMYK, Separation and DeviceN colorspaces.
5049: (4)                """
5050: (4)                return _mupdf.ll_fz_colorspace_is_subtractive(cs)
5051: (0)            def ll_fz_colorspace_n(cs):
5052: (4)                r"""
5053: (4)                Low-level wrapper for `::fz_colorspace_n()`.
```

```
5054: (4)               Query the number of colorants in a colorspace.
5055: (4)               """
5056: (4)               return _mupdf.ll_fz_colorspace_n(cs)
5057: (0)           def ll_fz_colorspace_name(cs):
5058: (4)               r"""
5059: (4)               Low-level wrapper for `::fz_colorspace_name()`.
5060: (4)               Query the name of a colorspace.
5061: (4)               The returned string has the same lifespan as the colorspace
5062: (4)               does. Caller should not free it.
5063: (4)               """
5064: (4)               return _mupdf.ll_fz_colorspace_name(cs)
5065: (0)           def ll_fz_colorspace_name_colorant(cs, n, name):
5066: (4)               r"""
5067: (4)               Low-level wrapper for `::fz_colorspace_name_colorant()`.
5068: (4)               Assign a name for a given colorant in a colorspace.
5069: (4)               Used while initially setting up a colorspace. The string is
5070: (4)               copied into local storage, so need not be retained by the
5071: (4)               caller.
5072: (4)               """
5073: (4)               return _mupdf.ll_fz_colorspace_name_colorant(cs, n, name)
5074: (0)           def ll_fz_colorspace_type(cs):
5075: (4)               r"""
5076: (4)               Low-level wrapper for `::fz_colorspace_type()`.
5077: (4)               Query the type of colorspace.
5078: (4)               """
5079: (4)               return _mupdf.ll_fz_colorspace_type(cs)
5080: (0)           def ll_fz_compare_separations(sep1, sep2):
5081: (4)               r"""
5082: (4)               Low-level wrapper for `::fz_compare_separations()`.
5083: (4)               Compare 2 separations structures (or NULLs).
5084: (4)               Return 0 if identical, non-zero if not identical.
5085: (4)               """
5086: (4)               return _mupdf.ll_fz_compare_separations(sep1, sep2)
5087: (0)           def ll_fz_compress_ccitt_fax_g3(data, columns, rows, stride):
5088: (4)               r"""
5089: (4)               Low-level wrapper for `::fz_compress_ccitt_fax_g3()`.
5090: (4)               Compress bitmap data as CCITT Group 3 1D fax image.
5091: (4)               Creates a stream assuming the default PDF parameters,
5092: (4)               except the number of columns.
5093: (4)               """
5094: (4)               return _mupdf.ll_fz_compress_ccitt_fax_g3(data, columns, rows, stride)
5095: (0)           def ll_fz_compress_ccitt_fax_g4(data, columns, rows, stride):
5096: (4)               r"""
5097: (4)               Low-level wrapper for `::fz_compress_ccitt_fax_g4()`.
5098: (4)               Compress bitmap data as CCITT Group 4 2D fax image.
5099: (4)               Creates a stream assuming the default PDF parameters, except
5100: (4)               K=-1 and the number of columns.
5101: (4)               """
5102: (4)               return _mupdf.ll_fz_compress_ccitt_fax_g4(data, columns, rows, stride)
5103: (0)           def ll_fz_compressed_buffer_size(buffer):
5104: (4)               r"""
5105: (4)               Low-level wrapper for `::fz_compressed_buffer_size()`.
5106: (4)               Return the storage size used for a buffer and its data.
5107: (4)               Used in implementing store handling.
5108: (4)               Never throws exceptions.
5109: (4)               """
5110: (4)               return _mupdf.ll_fz_compressed_buffer_size(buffer)
5111: (0)           def ll_fz_compressed_image_buffer(image):
5112: (4)               r"""
5113: (4)               Low-level wrapper for `::fz_compressed_image_buffer()`.
5114: (4)               Retrieve the underlying compressed data for an image.
5115: (4)               Returns a pointer to the underlying data buffer for an image,
5116: (4)               or NULL if this image is not based upon a compressed data
5117: (4)               buffer.
5118: (4)               This is not a reference counted structure, so no reference is
5119: (4)               returned. Lifespan is limited to that of the image itself.
5120: (4)               """
5121: (4)               return _mupdf.ll_fz_compressed_image_buffer(image)
5122: (0)           def ll_fz_compressed_image_type(image):
```

```
5123: (4)               r"""
5124: (4)               Low-level wrapper for `::fz_compressed_image_type()`.
5125: (4)               Return the type of a compressed image.
5126: (4)               Any non-compressed image will have the type returned as UNKNOWN.
5127: (4)               """
5128: (4)               return _mupdf.ll_fz_compressed_image_type(image)
5129: (0)           def ll_fz_concat(left, right):
5130: (4)               r"""
5131: (4)               Low-level wrapper for `::fz_concat()`.
5132: (4)               Multiply two matrices.
5133: (4)               The order of the two matrices are important since matrix
5134: (4)               multiplication is not commutative.
5135: (4)               Returns result.
5136: (4)               """
5137: (4)               return _mupdf.ll_fz_concat(left, right)
5138: (0)           def ll_fz_concat_push_drop(concat, chain):
5139: (4)               r"""
5140: (4)               Low-level wrapper for `::fz_concat_push_drop()`.
5141: (4)               Add a chained stream to the end of the concatenate filter.
5142: (4)               Ownership of chain is passed in.
5143: (4)               """
5144: (4)               return _mupdf.ll_fz_concat_push_drop(concat, chain)
5145: (0)           def ll_fz_contains_rect(a, b):
5146: (4)               r"""
5147: (4)               Low-level wrapper for `::fz_contains_rect()`.
5148: (4)               Test rectangle inclusion.
5149: (4)               Return true if a entirely contains b.
5150: (4)               """
5151: (4)               return _mupdf.ll_fz_contains_rect(a, b)
5152: (0)           def ll_fz_convert_color(ss, sv, ds, dv, _is, params):
5153: (4)               r"""
5154: (5)                Low-level wrapper for `::fz_convert_color()`.
5155: (4)               This function has out-params. Python/C# wrappers look like:
5156: (8)                   `ll_fz_convert_color(::fz_colorspace *ss, const float *sv,
::fz_colorspace *ds, ::fz_colorspace *is, ::fz_color_params params)` => float dv
5157: (8)                   Convert color values sv from colorspace ss into colorvalues dv
5158: (8)                   for colorspace ds, via an optional intervening space is,
5159: (8)                   respecting the given color_params.
5160: (4)               """
5161: (4)               return _mupdf.ll_fz_convert_color(ss, sv, ds, dv, _is, params)
5162: (0)           def ll_fz_convert_error(code):
5163: (4)               r"""
5164: (5)                Low-level wrapper for `::fz_convert_error()`.
5165: (4)               This function has out-params. Python/C# wrappers look like:
5166: (8)                   `ll_fz_convert_error()` => `(const char *, int code)`
5167: (4)               """
5168: (4)               return _mupdf.ll_fz_convert_error(code)
5169: (0)           def ll_fz_convert_indexed_pixmap_to_base(src):
5170: (4)               r"""
5171: (4)               Low-level wrapper for `::fz_convert_indexed_pixmap_to_base()`.
5172: (4)               Convert pixmap from indexed to base colorspace.
5173: (4)               This creates a new bitmap containing the converted pixmap data.
5174: (4)               """
5175: (4)               return _mupdf.ll_fz_convert_indexed_pixmap_to_base(src)
5176: (0)           def ll_fz_convert_pixmap(pix, cs_des, prf, default_cs, color_params,
keep_alpha):
5177: (4)               r"""
5178: (4)               Low-level wrapper for `::fz_convert_pixmap()`.
5179: (4)               Convert an existing pixmap to a desired
5180: (4)               colorspace. Other properties of the pixmap, such as resolution
5181: (4)               and position are copied to the converted pixmap.
5182: (4)               pix: The pixmap to convert.
5183: (4)               default_cs: If NULL pix->colorspace is used. It is possible that
5184: (4)               the data may need to be interpreted as one of the color spaces
5185: (4)               in default_cs.
5186: (4)               cs_des: Desired colorspace, may be NULL to denote alpha-only.
5187: (4)               prf: Proofing color space through which we need to convert.
5188: (4)               color_params: Parameters that may be used in conversion (e.g.
5189: (4)               ri).
```

```
5190: (4)               keep_alpha: If 0 any alpha component is removed, otherwise
5191: (4)               alpha is kept if present in the pixmap.
5192: (4)               """
5193: (4)               return _mupdf.ll_fz_convert_pixmap(pix, cs_des, prf, default_cs,
color_params, keep_alpha)
5194: (0)           def ll_fz_convert_separation_colors(src_cs, src_color, dst_seps, dst_cs,
dst_color, color_params):
5195: (4)               r"""
5196: (5)                Low-level wrapper for `::fz_convert_separation_colors()`.
5197: (4)               This function has out-params. Python/C# wrappers look like:
5198: (8)                   `ll_fz_convert_separation_colors(::fz_colorspace *src_cs, const float
*src_color, ::fz_separations *dst_seps, ::fz_colorspace *dst_cs, ::fz_color_params color_params)`
=> float dst_color
5199: (8)                   Convert a color given in terms of one colorspace,
5200: (8)                   to a color in terms of another colorspace/separations.
5201: (4)               """
5202: (4)               return _mupdf.ll_fz_convert_separation_colors(src_cs, src_color, dst_seps,
dst_cs, dst_color, color_params)
5203: (0)           def ll_fz_convert_separation_pixmap_to_base(src):
5204: (4)               r"""
5205: (4)               Low-level wrapper for `::fz_convert_separation_pixmap_to_base()`.
5206: (4)               Convert pixmap from DeviceN/Separation to base colorspace.
5207: (4)               This creates a new bitmap containing the converted pixmap data.
5208: (4)               """
5209: (4)               return _mupdf.ll_fz_convert_separation_pixmap_to_base(src)
5210: (0)           def ll_fz_copy_option(val, dest, maxlen):
5211: (4)               r"""
5212: (4)               Low-level wrapper for `::fz_copy_option()`.
5213: (4)               Copy an option (val) into a destination buffer (dest), of maxlen
5214: (4)               bytes.
5215: (4)               Returns the number of bytes (including terminator) that did not
5216: (4)               fit. If val is maxlen or greater bytes in size, it will be left
5217: (4)               unterminated.
5218: (4)               """
5219: (4)               return _mupdf.ll_fz_copy_option(val, dest, maxlen)
5220: (0)           def ll_fz_copy_pixmap_rect(dest, src, r, default_cs):
5221: (4)               r""" Low-level wrapper for `::fz_copy_pixmap_rect()`."""
5222: (4)               return _mupdf.ll_fz_copy_pixmap_rect(dest, src, r, default_cs)
5223: (0)           def ll_fz_copy_rectangle(page, area, crlf):
5224: (4)               r"""
5225: (4)               Low-level wrapper for `::fz_copy_rectangle()`.
5226: (4)               Return a newly allocated UTF-8 string with the text for a given
5227: (4)               selection rectangle.
5228: (4)               crlf: If true, write "\r\n" style line endings (otherwise "\n"
5229: (4)               only).
5230: (4)               """
5231: (4)               return _mupdf.ll_fz_copy_rectangle(page, area, crlf)
5232: (0)           def ll_fz_copy_selection(page, a, b, crlf):
5233: (4)               r"""
5234: (4)               Low-level wrapper for `::fz_copy_selection()`.
5235: (4)               Return a newly allocated UTF-8 string with the text for a given
5236: (4)               selection.
5237: (4)               crlf: If true, write "\r\n" style line endings (otherwise "\n"
5238: (4)               only).
5239: (4)               """
5240: (4)               return _mupdf.ll_fz_copy_selection(page, a, b, crlf)
5241: (0)           def ll_fz_count_active_separations(seps):
5242: (4)               r"""
5243: (4)               Low-level wrapper for `::fz_count_active_separations()`.
5244: (4)               Return the number of active separations.
5245: (4)               """
5246: (4)               return _mupdf.ll_fz_count_active_separations(seps)
5247: (0)           def ll_fz_count_archive_entries(arch):
5248: (4)               r"""
5249: (4)               Low-level wrapper for `::fz_count_archive_entries()`.
5250: (4)               Number of entries in archive.
5251: (4)               Will always return a value >= 0.
5252: (4)               May throw an exception if this type of archive cannot count the
5253: (4)               entries (such as a directory).
```

```
5254: (4)                """
5255: (4)                return _mupdf.ll_fz_count_archive_entries(arch)
5256: (0)            def ll_fz_count_chapter_pages(doc, chapter):
5257: (4)                r"""
5258: (4)                Low-level wrapper for `::fz_count_chapter_pages()`.
5259: (4)                Return the number of pages in a chapter.
5260: (4)                May return 0.
5261: (4)                """
5262: (4)                return _mupdf.ll_fz_count_chapter_pages(doc, chapter)
5263: (0)            def ll_fz_count_chapters(doc):
5264: (4)                r"""
5265: (4)                Low-level wrapper for `::fz_count_chapters()`.
5266: (4)                Return the number of chapters in the document.
5267: (4)                At least 1.
5268: (4)                """
5269: (4)                return _mupdf.ll_fz_count_chapters(doc)
5270: (0)            def ll_fz_count_pages(doc):
5271: (4)                r"""
5272: (4)                Low-level wrapper for `::fz_count_pages()`.
5273: (4)                Return the number of pages in document
5274: (4)                May return 0 for documents with no pages.
5275: (4)                """
5276: (4)                return _mupdf.ll_fz_count_pages(doc)
5277: (0)            def ll_fz_count_separations(sep):
5278: (4)                r""" Low-level wrapper for `::fz_count_separations()`."""
5279: (4)                return _mupdf.ll_fz_count_separations(sep)
5280: (0)            def ll_fz_create_link(page, bbox, uri):
5281: (4)                r"""
5282: (4)                Low-level wrapper for `::fz_create_link()`.
5283: (4)                Create a new link on a page.
5284: (4)                """
5285: (4)                return _mupdf.ll_fz_create_link(page, bbox, uri)
5286: (0)            def ll_fz_currentpoint(path):
5287: (4)                r"""
5288: (4)                Low-level wrapper for `::fz_currentpoint()`.
5289: (4)                Return the current point that a path has
5290: (4)                reached or (0,0) if empty.
5291: (4)                path: path to return the current point of.
5292: (4)                """
5293: (4)                return _mupdf.ll_fz_currentpoint(path)
5294: (0)            def ll_fz_curveto(path, x0, y0, x1, y1, x2, y2):
5295: (4)                r"""
5296: (4)                Low-level wrapper for `::fz_curveto()`.
5297: (4)                Append a 'curveto' command to an open path. (For a
5298: (4)                cubic bezier).
5299: (4)                path: The path to modify.
5300: (4)                x0, y0: The coordinates of the first control point for the
5301: (4)                curve.
5302: (4)                x1, y1: The coordinates of the second control point for the
5303: (4)                curve.
5304: (4)                x2, y2: The end coordinates for the curve.
5305: (4)                Throws exceptions on failure to allocate, or attempting to
5306: (4)                modify a packed path.
5307: (4)                """
5308: (4)                return _mupdf.ll_fz_curveto(path, x0, y0, x1, y1, x2, y2)
5309: (0)            def ll_fz_curvetov(path, x1, y1, x2, y2):
5310: (4)                r"""
5311: (4)                Low-level wrapper for `::fz_curvetov()`.
5312: (4)                Append a 'curvetov' command to an open path. (For a
5313: (4)                cubic bezier with the first control coordinate equal to
5314: (4)                the start point).
5315: (4)                path: The path to modify.
5316: (4)                x1, y1: The coordinates of the second control point for the
5317: (4)                curve.
5318: (4)                x2, y2: The end coordinates for the curve.
5319: (4)                Throws exceptions on failure to allocate, or attempting to
5320: (4)                modify a packed path.
5321: (4)                """
5322: (4)                return _mupdf.ll_fz_curvetov(path, x1, y1, x2, y2)
```

```
(0)         def ll_fz_curvetoy(path, x0, y0, x2, y2):
(4)             r"""
(4)             Low-level wrapper for `::fz_curvetoy()`.
(4)             Append a 'curvetoy' command to an open path. (For a
(4)             cubic bezier with the second control coordinate equal to
(4)             the end point).
(4)             path: The path to modify.
(4)             x0, y0: The coordinates of the first control point for the
(4)             curve.
(4)             x2, y2: The end coordinates for the curve (and the second
(4)             control coordinate).
(4)             Throws exceptions on failure to allocate, or attempting to
(4)             modify a packed path.
(4)             """
(4)             return _mupdf.ll_fz_curvetoy(path, x0, y0, x2, y2)
(0)         def ll_fz_debug_store(out):
(4)             r"""
(4)             Low-level wrapper for `::fz_debug_store()`.
(4)             Output debugging information for the current state of the store
(4)             to the given output channel.
(4)             """
(4)             return _mupdf.ll_fz_debug_store(out)
(0)         def ll_fz_debug_xml(item, level):
(4)             r"""
(4)             Low-level wrapper for `::fz_debug_xml()`.
(4)             Pretty-print an XML tree to stdout. (Deprecated, use
(4)             fz_output_xml in preference).
(4)             """
(4)             return _mupdf.ll_fz_debug_xml(item, level)
(0)         def ll_fz_decode_tile(pix, decode):
(4)             r""" Low-level wrapper for `::fz_decode_tile()`."""
(4)             return _mupdf.ll_fz_decode_tile(pix, decode)
(0)         def ll_fz_decode_uri(s):
(4)             r"""
(5)              Low-level wrapper for `::fz_decode_uri()`.
(4)             Return a new string representing the unencoded version of the given URI.
(4)             This decodes all escape sequences except those that would result in a
reserved
(4)             character that are part of the URI syntax (; / ? : @ & = + $ , #).
(4)             """
(4)             return _mupdf.ll_fz_decode_uri(s)
(0)         def ll_fz_decode_uri_component(s):
(4)             r"""
(5)              Low-level wrapper for `::fz_decode_uri_component()`.
(4)             Return a new string representing the unencoded version of the given URI
component.
(4)             This decodes all escape sequences!
(4)             """
(4)             return _mupdf.ll_fz_decode_uri_component(s)
(0)         def ll_fz_decomp_image_from_stream(stm, image, subarea, indexed, l2factor,
l2extra):
(4)             r"""
(5)              Low-level wrapper for `::fz_decomp_image_from_stream()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_fz_decomp_image_from_stream(::fz_stream *stm,
::fz_compressed_image *image, ::fz_irect *subarea, int indexed, int l2factor)` => `(fz_pixmap *,
int l2extra)`
(8)                 Decode a subarea of a compressed image. l2factor is the amount
(8)                 of subsampling inbuilt to the stream (i.e. performed by the
(8)                 decoder). If non NULL, l2extra is the extra amount of
(8)                 subsampling that should be performed by this routine. This will
(8)                 be updated on exit to the amount of subsampling that is still
(8)                 required to be done.
(8)                 Returns a kept reference.
(4)             """
(4)             return _mupdf.ll_fz_decomp_image_from_stream(stm, image, subarea, indexed,
l2factor, l2extra)
(0)         def ll_fz_decouple_type3_font(font, t3doc):
(4)             r""" Low-level wrapper for `::fz_decouple_type3_font()`."""
```

```
(4)             return _mupdf.ll_fz_decouple_type3_font(font, t3doc)
(0)         def ll_fz_default_cmyk(default_cs):
(4)             r""" Low-level wrapper for `::fz_default_cmyk()`."""
(4)             return _mupdf.ll_fz_default_cmyk(default_cs)
(0)         def ll_fz_default_error_callback(user, message):
(4)             r"""
(4)             Low-level wrapper for `::fz_default_error_callback()`.
(4)             FIXME: Better not to expose fz_default_error_callback, and
(4)             fz_default_warning callback and to allow 'NULL' to be used
(4)             int fz_set_xxxx_callback to mean "defaults".
(4)             FIXME: Do we need/want functions like
(4)             fz_error_callback(ctx, message) to allow callers to inject
(4)             stuff into the error/warning streams?
(4)             The default error callback. Declared publicly just so that the
(4)             error callback can be set back to this after it has been
(4)             overridden.
(4)             """
(4)             return _mupdf.ll_fz_default_error_callback(user, message)
(0)         def ll_fz_default_gray(default_cs):
(4)             r"""
(4)             Low-level wrapper for `::fz_default_gray()`.
(4)             Retrieve default colorspaces (typically page local).
(4)             If default_cs is non NULL, the default is retrieved from there,
(4)             otherwise the global default is retrieved.
(4)             These return borrowed references that should not be dropped,
(4)             unless they are kept first.
(4)             """
(4)             return _mupdf.ll_fz_default_gray(default_cs)
(0)         def ll_fz_default_halftone(num_comps):
(4)             r"""
(4)             Low-level wrapper for `::fz_default_halftone()`.
(4)             Create a 'default' halftone structure
(4)             for the given number of components.
(4)             num_comps: The number of components to use.
(4)             Returns a simple default halftone. The default halftone uses
(4)             the same halftone tile for each plane, which may not be ideal
(4)             for all purposes.
(4)             """
(4)             return _mupdf.ll_fz_default_halftone(num_comps)
(0)         def ll_fz_default_output_intent(default_cs):
(4)             r""" Low-level wrapper for `::fz_default_output_intent()`."""
(4)             return _mupdf.ll_fz_default_output_intent(default_cs)
(0)         def ll_fz_default_rgb(default_cs):
(4)             r""" Low-level wrapper for `::fz_default_rgb()`."""
(4)             return _mupdf.ll_fz_default_rgb(default_cs)
(0)         def ll_fz_default_warning_callback(user, message):
(4)             r"""
(4)             Low-level wrapper for `::fz_default_warning_callback()`.
(4)             The default warning callback. Declared publicly just so that
(4)             the warning callback can be set back to this after it has been
(4)             overridden.
(4)             """
(4)             return _mupdf.ll_fz_default_warning_callback(user, message)
(0)         def ll_fz_defer_reap_end():
(4)             r"""
(4)             Low-level wrapper for `::fz_defer_reap_end()`.
(4)             Decrement the defer reap count.
(4)             If the defer reap count returns to 0, and the store
(4)             has reapable objects in, a reap pass will begin.
(4)             Call this at the end of a process during which you
(4)             potentially might drop many reapable objects.
(4)             It is vital that every fz_defer_reap_start is matched
(4)             by a fz_defer_reap_end call.
(4)             """
(4)             return _mupdf.ll_fz_defer_reap_end()
(0)         def ll_fz_defer_reap_start():
(4)             r"""
(4)             Low-level wrapper for `::fz_defer_reap_start()`.
(4)             Increment the defer reap count.
```

```
5455: (4)               No reap operations will take place (except for those
5456: (4)               triggered by an immediate failed malloc) until the
5457: (4)               defer reap count returns to 0.
5458: (4)               Call this at the start of a process during which you
5459: (4)               potentially might drop many reapable objects.
5460: (4)               It is vital that every fz_defer_reap_start is matched
5461: (4)               by a fz_defer_reap_end call.
5462: (4)               """
5463: (4)               return _mupdf.ll_fz_defer_reap_start()
5464: (0)           def ll_fz_deflate(dest, compressed_length, source, source_length, level):
5465: (4)               r"""
5466: (5)                Low-level wrapper for `::fz_deflate()`.
5467: (4)               This function has out-params. Python/C# wrappers look like:
5468: (8)                   `ll_fz_deflate(unsigned char *dest, const unsigned char *source,
size_t source_length, ::fz_deflate_level level)` => size_t compressed_length
5469: (8)                   Compress source_length bytes of data starting
5470: (8)                   at source, into a buffer of length *destLen, starting at dest.
5471: (4)               compressed_length will be updated on exit to contain the size
5472: (8)                   actually used.
5473: (4)               """
5474: (4)               return _mupdf.ll_fz_deflate(dest, compressed_length, source,
source_length, level)
5475: (0)           def ll_fz_deflate_bound(size):
5476: (4)               r"""
5477: (4)               Low-level wrapper for `::fz_deflate_bound()`.
5478: (4)               Returns the upper bound on the
5479: (4)               size of flated data of length size.
5480: (4)               """
5481: (4)               return _mupdf.ll_fz_deflate_bound(size)
5482: (0)           def ll_fz_delete_link(page, link):
5483: (4)               r"""
5484: (4)               Low-level wrapper for `::fz_delete_link()`.
5485: (4)               Delete an existing link on a page.
5486: (4)               """
5487: (4)               return _mupdf.ll_fz_delete_link(page, link)
5488: (0)           def ll_fz_detach_xml(node):
5489: (4)               r"""
5490: (4)               Low-level wrapper for `::fz_detach_xml()`.
5491: (4)               Detach a node from the tree, unlinking it from its parent,
5492: (4)               and setting the document root to the node.
5493: (4)               """
5494: (4)               return _mupdf.ll_fz_detach_xml(node)
5495: (0)           def ll_fz_device_bgr():
5496: (4)               r""" Low-level wrapper for `::fz_device_bgr()`."""
5497: (4)               return _mupdf.ll_fz_device_bgr()
5498: (0)           def ll_fz_device_cmyk():
5499: (4)               r""" Low-level wrapper for `::fz_device_cmyk()`."""
5500: (4)               return _mupdf.ll_fz_device_cmyk()
5501: (0)           def ll_fz_device_current_scissor(dev):
5502: (4)               r"""
5503: (4)               Low-level wrapper for `::fz_device_current_scissor()`.
5504: (4)               Find current scissor region as tracked by the device.
5505: (4)               """
5506: (4)               return _mupdf.ll_fz_device_current_scissor(dev)
5507: (0)           def ll_fz_device_gray():
5508: (4)               r"""
5509: (4)               Low-level wrapper for `::fz_device_gray()`.
5510: (4)               Retrieve global default colorspaces.
5511: (4)               These return borrowed references that should not be dropped,
5512: (4)               unless they are kept first.
5513: (4)               """
5514: (4)               return _mupdf.ll_fz_device_gray()
5515: (0)           def ll_fz_device_lab():
5516: (4)               r""" Low-level wrapper for `::fz_device_lab()`."""
5517: (4)               return _mupdf.ll_fz_device_lab()
5518: (0)           def ll_fz_device_rgb():
5519: (4)               r""" Low-level wrapper for `::fz_device_rgb()`."""
5520: (4)               return _mupdf.ll_fz_device_rgb()
5521: (0)           def ll_fz_dirname(dir, path, dirsize):
```

```
5522: (4)               r"""
5523: (4)               Low-level wrapper for `::fz_dirname()`.
5524: (4)               extract the directory component from a path.
5525: (4)               """
5526: (4)               return _mupdf.ll_fz_dirname(dir, path, dirsize)
5527: (0)           def ll_fz_disable_device_hints(dev, hints):
5528: (4)               r"""
5529: (4)               Low-level wrapper for `::fz_disable_device_hints()`.
5530: (4)               Disable (clear) hint bits within the hint bitfield for a device.
5531: (4)               """
5532: (4)               return _mupdf.ll_fz_disable_device_hints(dev, hints)
5533: (0)           def ll_fz_disable_icc():
5534: (4)               r"""
5535: (4)               Low-level wrapper for `::fz_disable_icc()`.
5536: (4)               Disable icc profile based operation.
5537: (4)               """
5538: (4)               return _mupdf.ll_fz_disable_icc()
5539: (0)           def ll_fz_display_list_is_empty(list):
5540: (4)               r"""
5541: (4)               Low-level wrapper for `::fz_display_list_is_empty()`.
5542: (4)               Check for a display list being empty
5543: (4)               list: The list to check.
5544: (4)               Returns true if empty, false otherwise.
5545: (4)               """
5546: (4)               return _mupdf.ll_fz_display_list_is_empty(list)
5547: (0)           def ll_fz_div255(c, a):
5548: (4)               r"""
5549: (4)               Low-level wrapper for `::fz_div255()`.
5550: (4)               Undo alpha premultiplication.
5551: (4)               """
5552: (4)               return _mupdf.ll_fz_div255(c, a)
5553: (0)           def ll_fz_do_always():
5554: (4)               r""" Low-level wrapper for `::fz_do_always()`."""
5555: (4)               return _mupdf.ll_fz_do_always()
5556: (0)           def ll_fz_do_catch():
5557: (4)               r""" Low-level wrapper for `::fz_do_catch()`."""
5558: (4)               return _mupdf.ll_fz_do_catch()
5559: (0)           def ll_fz_do_try():
5560: (4)               r""" Low-level wrapper for `::fz_do_try()`."""
5561: (4)               return _mupdf.ll_fz_do_try()
5562: (0)           def ll_fz_document_handler_open(handler, stream, accel, dir, recognize_state):
5563: (4)               r"""
5564: (5)                Low-level wrapper for `::fz_document_handler_open()`.  Helper for calling
`fz_document_handler::open` function pointer via
5565: (4)               Swig from Python/C#.
5566: (4)               """
5567: (4)               return _mupdf.ll_fz_document_handler_open(handler, stream, accel, dir,
recognize_state)
5568: (0)           def ll_fz_document_handler_recognize(handler, magic):
5569: (4)               r"""
5570: (5)                Low-level wrapper for `::fz_document_handler_recognize()`.  Helper for
calling a `fz_document_handler::recognize` function
5571: (4)               pointer via Swig from Python/C#.
5572: (4)               """
5573: (4)               return _mupdf.ll_fz_document_handler_recognize(handler, magic)
5574: (0)           def ll_fz_document_output_intent(doc):
5575: (4)               r"""
5576: (4)               Low-level wrapper for `::fz_document_output_intent()`.
5577: (4)               Find the output intent colorspace if the document has defined
5578: (4)               one.
5579: (4)               Returns a borrowed reference that should not be dropped, unless
5580: (4)               it is kept first.
5581: (4)               """
5582: (4)               return _mupdf.ll_fz_document_output_intent(doc)
5583: (0)           def ll_fz_document_supports_accelerator(doc):
5584: (4)               r"""
5585: (4)               Low-level wrapper for `::fz_document_supports_accelerator()`.
5586: (4)               Query if the document supports the saving of accelerator data.
5587: (4)               """
```

```
5588: (4)                return _mupdf.ll_fz_document_supports_accelerator(doc)
5589: (0)            def ll_fz_dom_add_attribute(elt, att, value):
5590: (4)                r"""
5591: (4)                Low-level wrapper for `::fz_dom_add_attribute()`.
5592: (4)                Add an attribute to an element.
5593: (4)                Ownership of att and value remain with the caller.
5594: (4)                """
5595: (4)                return _mupdf.ll_fz_dom_add_attribute(elt, att, value)
5596: (0)            def ll_fz_dom_append_child(parent, child):
5597: (4)                r"""
5598: (4)                Low-level wrapper for `::fz_dom_append_child()`.
5599: (4)                Insert an element as the last child of a parent, unlinking the
5600: (4)                child from its current position if required.
5601: (4)                """
5602: (4)                return _mupdf.ll_fz_dom_append_child(parent, child)
5603: (0)            def ll_fz_dom_attribute(elt, att):
5604: (4)                r"""
5605: (4)                Low-level wrapper for `::fz_dom_attribute()`.
5606: (4)                Retrieve the value of a given attribute from a given element.
5607: (4)                Returns a borrowed pointer to the value or NULL if not found.
5608: (4)                """
5609: (4)                return _mupdf.ll_fz_dom_attribute(elt, att)
5610: (0)            def ll_fz_dom_body(dom):
5611: (4)                r"""
5612: (4)                Low-level wrapper for `::fz_dom_body()`.
5613: (4)                Return a borrowed reference for the 'body' element of
5614: (4)                the given DOM.
5615: (4)                """
5616: (4)                return _mupdf.ll_fz_dom_body(dom)
5617: (0)            def ll_fz_dom_clone(elt):
5618: (4)                r"""
5619: (4)                Low-level wrapper for `::fz_dom_clone()`.
5620: (4)                Clone an element (and its children).
5621: (4)                A borrowed reference to the clone is returned. The clone is not
5622: (4)                yet linked into the DOM.
5623: (4)                """
5624: (4)                return _mupdf.ll_fz_dom_clone(elt)
5625: (0)            def ll_fz_dom_create_element(dom, tag):
5626: (4)                r"""
5627: (4)                Low-level wrapper for `::fz_dom_create_element()`.
5628: (4)                Create an element of a given tag type for the given DOM.
5629: (4)                The element is not linked into the DOM yet.
5630: (4)                """
5631: (4)                return _mupdf.ll_fz_dom_create_element(dom, tag)
5632: (0)            def ll_fz_dom_create_text_node(dom, text):
5633: (4)                r"""
5634: (4)                Low-level wrapper for `::fz_dom_create_text_node()`.
5635: (4)                Create a text node for the given DOM.
5636: (4)                The element is not linked into the DOM yet.
5637: (4)                """
5638: (4)                return _mupdf.ll_fz_dom_create_text_node(dom, text)
5639: (0)            def ll_fz_dom_document_element(dom):
5640: (4)                r"""
5641: (4)                Low-level wrapper for `::fz_dom_document_element()`.
5642: (4)                Return a borrowed reference for the document (the top
5643: (4)                level element) of the DOM.
5644: (4)                """
5645: (4)                return _mupdf.ll_fz_dom_document_element(dom)
5646: (0)            def ll_fz_dom_find(elt, tag, att, match):
5647: (4)                r"""
5648: (4)                Low-level wrapper for `::fz_dom_find()`.
5649: (4)                Find the first element matching the requirements in a depth first
traversal from elt.
5650: (4)                The tagname must match tag, unless tag is NULL, when all tag names are
considered to match.
5651: (4)                If att is NULL, then all tags match.
5652: (4)                Otherwise:
5653: (8)                    If match is NULL, then only nodes that have an att attribute match.
5654: (8)                    If match is non-NULL, then only nodes that have an att attribute that
```

```
matches match match.
5655: (4)               Returns NULL (if no match found), or a borrowed reference to the first
matching element.
5656: (4)               """
5657: (4)               return _mupdf.ll_fz_dom_find(elt, tag, att, match)
5658: (0)           def ll_fz_dom_find_next(elt, tag, att, match):
5659: (4)               r"""
5660: (4)               Low-level wrapper for `::fz_dom_find_next()`.
5661: (4)               Find the next element matching the requirements.
5662: (4)               """
5663: (4)               return _mupdf.ll_fz_dom_find_next(elt, tag, att, match)
5664: (0)           def ll_fz_dom_first_child(elt):
5665: (4)               r"""
5666: (4)               Low-level wrapper for `::fz_dom_first_child()`.
5667: (4)               Return a borrowed reference to the first child of a node,
5668: (4)               or NULL if there isn't one.
5669: (4)               """
5670: (4)               return _mupdf.ll_fz_dom_first_child(elt)
5671: (0)           def ll_fz_dom_get_attribute(elt, i, att):
5672: (4)               r"""
5673: (5)                Low-level wrapper for `::fz_dom_get_attribute()`.
5674: (4)               This function has out-params. Python/C# wrappers look like:
5675: (8)                   `ll_fz_dom_get_attribute(::fz_xml *elt, int i)` => `(const char *,
const char *att)`
5676: (8)                   Enumerate through the attributes of an element.
5677: (8)                   Call with i=0,1,2,3... to enumerate attributes.
5678: (8)                   On return *att and the return value will be NULL if there are not
5679: (8)                   that many attributes to read. Otherwise, *att will be filled in
5680: (8)                   with a borrowed pointer to the attribute name, and the return
5681: (8)                   value will be a borrowed pointer to the value.
5682: (4)               """
5683: (4)               return _mupdf.ll_fz_dom_get_attribute(elt, i, att)
5684: (0)           def ll_fz_dom_insert_after(node, new_elt):
5685: (4)               r"""
5686: (4)               Low-level wrapper for `::fz_dom_insert_after()`.
5687: (4)               Insert an element (new_elt), after another element (node),
5688: (4)               unlinking the new_elt from its current position if required.
5689: (4)               """
5690: (4)               return _mupdf.ll_fz_dom_insert_after(node, new_elt)
5691: (0)           def ll_fz_dom_insert_before(node, new_elt):
5692: (4)               r"""
5693: (4)               Low-level wrapper for `::fz_dom_insert_before()`.
5694: (4)               Insert an element (new_elt), before another element (node),
5695: (4)               unlinking the new_elt from its current position if required.
5696: (4)               """
5697: (4)               return _mupdf.ll_fz_dom_insert_before(node, new_elt)
5698: (0)           def ll_fz_dom_next(elt):
5699: (4)               r"""
5700: (4)               Low-level wrapper for `::fz_dom_next()`.
5701: (4)               Return a borrowed reference to the next sibling of a node,
5702: (4)               or NULL if there isn't one.
5703: (4)               """
5704: (4)               return _mupdf.ll_fz_dom_next(elt)
5705: (0)           def ll_fz_dom_parent(elt):
5706: (4)               r"""
5707: (4)               Low-level wrapper for `::fz_dom_parent()`.
5708: (4)               Return a borrowed reference to the parent of a node,
5709: (4)               or NULL if there isn't one.
5710: (4)               """
5711: (4)               return _mupdf.ll_fz_dom_parent(elt)
5712: (0)           def ll_fz_dom_previous(elt):
5713: (4)               r"""
5714: (4)               Low-level wrapper for `::fz_dom_previous()`.
5715: (4)               Return a borrowed reference to the previous sibling of a node,
5716: (4)               or NULL if there isn't one.
5717: (4)               """
5718: (4)               return _mupdf.ll_fz_dom_previous(elt)
5719: (0)           def ll_fz_dom_remove(elt):
5720: (4)               r"""
```

```
5721: (4)               Low-level wrapper for `::fz_dom_remove()`.
5722: (4)               Remove an element from the DOM. The element can be added back elsewhere
5723: (4)               if required.
5724: (4)               No reference counting changes for the element.
5725: (4)               """
5726: (4)               return _mupdf.ll_fz_dom_remove(elt)
5727: (0)           def ll_fz_dom_remove_attribute(elt, att):
5728: (4)               r"""
5729: (4)               Low-level wrapper for `::fz_dom_remove_attribute()`.
5730: (4)               Remove an attribute from an element.
5731: (4)               """
5732: (4)               return _mupdf.ll_fz_dom_remove_attribute(elt, att)
5733: (0)           def ll_fz_draw_story(story, dev, ctm):
5734: (4)               r""" Low-level wrapper for `::fz_draw_story()`."""
5735: (4)               return _mupdf.ll_fz_draw_story(story, dev, ctm)
5736: (0)           def ll_fz_drop_archive(arch):
5737: (4)               r"""
5738: (4)               Low-level wrapper for `::fz_drop_archive()`.
5739: (4)               Drop a reference to an archive.
5740: (4)               When the last reference is dropped, this closes and releases
5741: (4)               any memory or filehandles associated with the archive.
5742: (4)               """
5743: (4)               return _mupdf.ll_fz_drop_archive(arch)
5744: (0)           def ll_fz_drop_band_writer(writer):
5745: (4)               r"""
5746: (4)               Low-level wrapper for `::fz_drop_band_writer()`.
5747: (4)               Drop the reference to the band writer, causing it to be
5748: (4)               destroyed.
5749: (4)               Never throws an exception.
5750: (4)               """
5751: (4)               return _mupdf.ll_fz_drop_band_writer(writer)
5752: (0)           def ll_fz_drop_bitmap(bit):
5753: (4)               r"""
5754: (4)               Low-level wrapper for `::fz_drop_bitmap()`.
5755: (4)               Drop a reference to the bitmap. When the reference count reaches
5756: (4)               zero, the bitmap will be destroyed.
5757: (4)               Never throws exceptions.
5758: (4)               """
5759: (4)               return _mupdf.ll_fz_drop_bitmap(bit)
5760: (0)           def ll_fz_drop_buffer(buf):
5761: (4)               r"""
5762: (4)               Low-level wrapper for `::fz_drop_buffer()`.
5763: (4)               Drop a reference to the buffer. When the reference count reaches
5764: (4)               zero, the buffer is destroyed.
5765: (4)               Never throws exceptions.
5766: (4)               """
5767: (4)               return _mupdf.ll_fz_drop_buffer(buf)
5768: (0)           def ll_fz_drop_colorspace(colorspace):
5769: (4)               r"""
5770: (4)               Low-level wrapper for `::fz_drop_colorspace()`.
5771: (4)               Drops a reference to the colorspace.
5772: (4)               When the reference count reaches zero, the colorspace is
5773: (4)               destroyed.
5774: (4)               """
5775: (4)               return _mupdf.ll_fz_drop_colorspace(colorspace)
5776: (0)           def ll_fz_drop_colorspace_imp(cs_):
5777: (4)               r""" Low-level wrapper for `::fz_drop_colorspace_imp()`."""
5778: (4)               return _mupdf.ll_fz_drop_colorspace_imp(cs_)
5779: (0)           def ll_fz_drop_compressed_buffer(buf):
5780: (4)               r"""
5781: (4)               Low-level wrapper for `::fz_drop_compressed_buffer()`.
5782: (4)               Drop a reference to a compressed buffer. Destroys the buffer
5783: (4)               and frees any storage/other references held by it.
5784: (4)               Never throws exceptions.
5785: (4)               """
5786: (4)               return _mupdf.ll_fz_drop_compressed_buffer(buf)
5787: (0)           def ll_fz_drop_context():
5788: (4)               r"""
5789: (4)               Low-level wrapper for `::fz_drop_context()`.
```

```
5790: (4)               Free a context and its global state.
5791: (4)               The context and all of its global state is freed, and any
5792: (4)               buffered warnings are flushed (see fz_flush_warnings). If NULL
5793: (4)               is passed in nothing will happen.
5794: (4)               Must not be called for a context that is being used in an active
5795: (4)               fz_try(), fz_always() or fz_catch() block.
5796: (4)               """
5797: (4)               return _mupdf.ll_fz_drop_context()
5798: (0)           def ll_fz_drop_default_colorspaces(default_cs):
5799: (4)               r"""
5800: (4)               Low-level wrapper for `::fz_drop_default_colorspaces()`.
5801: (4)               Drop a reference to the default colorspaces structure. When the
5802: (4)               reference count reaches 0, the references it holds internally
5803: (4)               to the underlying colorspaces will be dropped, and the structure
5804: (4)               will be destroyed.
5805: (4)               Never throws exceptions.
5806: (4)               """
5807: (4)               return _mupdf.ll_fz_drop_default_colorspaces(default_cs)
5808: (0)           def ll_fz_drop_device(dev):
5809: (4)               r"""
5810: (4)               Low-level wrapper for `::fz_drop_device()`.
5811: (4)               Reduce the reference count on a device. When the reference count
5812: (4)               reaches zero, the device and its resources will be freed.
5813: (4)               Don't forget to call fz_close_device before dropping the device,
5814: (4)               or you may get incomplete output!
5815: (4)               Never throws exceptions.
5816: (4)               """
5817: (4)               return _mupdf.ll_fz_drop_device(dev)
5818: (0)           def ll_fz_drop_display_list(list):
5819: (4)               r"""
5820: (4)               Low-level wrapper for `::fz_drop_display_list()`.
5821: (4)               Decrement the reference count for a display list. When the
5822: (4)               reference count reaches zero, all the references in the display
5823: (4)               list itself are dropped, and the display list is freed.
5824: (4)               Never throws exceptions.
5825: (4)               """
5826: (4)               return _mupdf.ll_fz_drop_display_list(list)
5827: (0)           def ll_fz_drop_document(doc):
5828: (4)               r"""
5829: (4)               Low-level wrapper for `::fz_drop_document()`.
5830: (4)               Decrement the document reference count. When the reference
5831: (4)               count reaches 0, the document and all it's references are
5832: (4)               freed.
5833: (4)               Never throws exceptions.
5834: (4)               """
5835: (4)               return _mupdf.ll_fz_drop_document(doc)
5836: (0)           def ll_fz_drop_document_writer(wri):
5837: (4)               r"""
5838: (4)               Low-level wrapper for `::fz_drop_document_writer()`.
5839: (4)               Called to discard a fz_document_writer.
5840: (4)               This may be called at any time during the process to release all
5841: (4)               the resources owned by the writer.
5842: (4)               Calling drop without having previously called close may leave
5843: (4)               the file in an inconsistent state.
5844: (4)               """
5845: (4)               return _mupdf.ll_fz_drop_document_writer(wri)
5846: (0)           def ll_fz_drop_font(font):
5847: (4)               r"""
5848: (4)               Low-level wrapper for `::fz_drop_font()`.
5849: (4)               Drop a reference to a fz_font, destroying the
5850: (4)               font when the last reference is dropped.
5851: (4)               font: The font to drop a reference to.
5852: (4)               """
5853: (4)               return _mupdf.ll_fz_drop_font(font)
5854: (0)           def ll_fz_drop_function(func):
5855: (4)               r""" Low-level wrapper for `::fz_drop_function()`."""
5856: (4)               return _mupdf.ll_fz_drop_function(func)
5857: (0)           def ll_fz_drop_glyph(pix):
5858: (4)               r"""
```

```
5859: (4)               Low-level wrapper for `::fz_drop_glyph()`.
5860: (4)               Drop a reference and free a glyph.
5861: (4)               Decrement the reference count for the glyph. When no
5862: (4)               references remain the glyph will be freed.
5863: (4)               """
5864: (4)               return _mupdf.ll_fz_drop_glyph(pix)
5865: (0)           def ll_fz_drop_halftone(ht):
5866: (4)               r"""
5867: (4)               Low-level wrapper for `::fz_drop_halftone()`.
5868: (4)               Drop a reference to the halftone. When the reference count
5869: (4)               reaches zero, the halftone is destroyed.
5870: (4)               Never throws exceptions.
5871: (4)               """
5872: (4)               return _mupdf.ll_fz_drop_halftone(ht)
5873: (0)           def ll_fz_drop_hash_table(table):
5874: (4)               r"""
5875: (4)               Low-level wrapper for `::fz_drop_hash_table()`.
5876: (4)               Destroy the hash table.
5877: (4)               Values are dropped using the drop function.
5878: (4)               """
5879: (4)               return _mupdf.ll_fz_drop_hash_table(table)
5880: (0)           def ll_fz_drop_image(image):
5881: (4)               r"""
5882: (4)               Low-level wrapper for `::fz_drop_image()`.
5883: (4)               Decrement the (normal) reference count for an image. When the
5884: (4)               total (normal + key) reference count reaches zero, the image and
5885: (4)               its resources are freed.
5886: (4)               Never throws exceptions.
5887: (4)               """
5888: (4)               return _mupdf.ll_fz_drop_image(image)
5889: (0)           def ll_fz_drop_image_base(image):
5890: (4)               r"""
5891: (4)               Low-level wrapper for `::fz_drop_image_base()`.
5892: (4)               Internal destructor for the base image class members.
5893: (4)               Exposed to allow derived image classes to be written.
5894: (4)               """
5895: (4)               return _mupdf.ll_fz_drop_image_base(image)
5896: (0)           def ll_fz_drop_image_imp(image):
5897: (4)               r"""
5898: (4)               Low-level wrapper for `::fz_drop_image_imp()`.
5899: (4)               Internal destructor exposed for fz_store integration.
5900: (4)               """
5901: (4)               return _mupdf.ll_fz_drop_image_imp(image)
5902: (0)           def ll_fz_drop_image_store_key(image):
5903: (4)               r"""
5904: (4)               Low-level wrapper for `::fz_drop_image_store_key()`.
5905: (4)               Decrement the store key reference count for an image. When the
5906: (4)               total (normal + key) reference count reaches zero, the image and
5907: (4)               its resources are freed.
5908: (4)               Never throws exceptions.
5909: (4)               """
5910: (4)               return _mupdf.ll_fz_drop_image_store_key(image)
5911: (0)           def ll_fz_drop_imp(p, refs):
5912: (4)               r"""
5913: (5)                Low-level wrapper for `::fz_drop_imp()`.
5914: (4)               This function has out-params. Python/C# wrappers look like:
5915: (8)                   `ll_fz_drop_imp(void *p)` => `(int, int refs)`
5916: (4)               """
5917: (4)               return _mupdf.ll_fz_drop_imp(p, refs)
5918: (0)           def ll_fz_drop_imp16(p, refs):
5919: (4)               r"""
5920: (5)                Low-level wrapper for `::fz_drop_imp16()`.
5921: (4)               This function has out-params. Python/C# wrappers look like:
5922: (8)                   `ll_fz_drop_imp16(void *p)` => `(int, int16_t refs)`
5923: (4)               """
5924: (4)               return _mupdf.ll_fz_drop_imp16(p, refs)
5925: (0)           def ll_fz_drop_imp8(p, refs):
5926: (4)               r""" Low-level wrapper for `::fz_drop_imp8()`."""
5927: (4)               return _mupdf.ll_fz_drop_imp8(p, refs)
```

```
(0)     def ll_fz_drop_jbig2_globals(globals):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_jbig2_globals()`.
(4)         Decrement the reference count for a jbig2 globals record.
(4)         When the reference count hits zero, the record is freed.
(4)         Never throws an exception.
(4)         """
(4)         return _mupdf.ll_fz_drop_jbig2_globals(globals)
(0)     def ll_fz_drop_jbig2_globals_imp(globals):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_jbig2_globals_imp()`.
(4)         Special jbig2 globals drop function for use in implementing
(4)         store support.
(4)         """
(4)         return _mupdf.ll_fz_drop_jbig2_globals_imp(globals)
(0)     def ll_fz_drop_key_storable(arg_1):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_key_storable()`.
(4)         Decrement the (normal) reference count for a storable object.
(4)         When the total reference count hits zero, the drop function for
(4)         that object is called to free the object.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_drop_key_storable(arg_1)
(0)     def ll_fz_drop_key_storable_key(arg_1):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_key_storable_key()`.
(4)         Decrement the (key) reference count for a storable object.
(4)         When the total reference count hits zero, the drop function for
(4)         that object is called to free the object.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_drop_key_storable_key(arg_1)
(0)     def ll_fz_drop_layout(block):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_layout()`.
(4)         Drop layout block. Free the pool, and linked blocks.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_drop_layout(block)
(0)     def ll_fz_drop_link(link):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_link()`.
(4)         Decrement the reference count for a link. When the reference
(4)         count reaches zero, the link is destroyed.
(4)         When a link is freed, the reference for any linked link (next)
(4)         is dropped too, thus an entire linked list of fz_link's can be
(4)         freed by just dropping the head.
(4)         """
(4)         return _mupdf.ll_fz_drop_link(link)
(0)     def ll_fz_drop_outline(outline):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_outline()`.
(4)         Decrements the reference count. When the reference point
(4)         reaches zero, the outline is freed.
(4)         When freed, it will drop linked    outline entries (next and down)
(4)         too, thus a whole outline structure can be dropped by dropping
(4)         the top entry.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_drop_outline(outline)
(0)     def ll_fz_drop_outline_iterator(iter):
(4)         r"""
(4)         Low-level wrapper for `::fz_drop_outline_iterator()`.
(4)         Drop the current iterator.
(4)         """
(4)         return _mupdf.ll_fz_drop_outline_iterator(iter)
(0)     def ll_fz_drop_output(arg_1):
(4)         r"""
```

```
5997: (4)               Low-level wrapper for `::fz_drop_output()`.
5998: (4)               Free an output stream. Don't forget to close it first!
5999: (4)               """
6000: (4)               return _mupdf.ll_fz_drop_output(arg_1)
6001: (0)           def ll_fz_drop_page(page):
6002: (4)               r"""
6003: (4)               Low-level wrapper for `::fz_drop_page()`.
6004: (4)               Decrements the reference count for the page. When the reference
6005: (4)               count hits 0, the page and its references are freed.
6006: (4)               Never throws exceptions.
6007: (4)               """
6008: (4)               return _mupdf.ll_fz_drop_page(page)
6009: (0)           def ll_fz_drop_path(path):
6010: (4)               r"""
6011: (4)               Low-level wrapper for `::fz_drop_path()`.
6012: (4)               Decrement the reference count. When the reference count hits
6013: (4)               zero, free the path.
6014: (4)               All paths can be dropped, regardless of their packing type.
6015: (4)               Packed paths do not own the blocks into which they are packed
6016: (4)               so dropping them does not free those blocks.
6017: (4)               Never throws exceptions.
6018: (4)               """
6019: (4)               return _mupdf.ll_fz_drop_path(path)
6020: (0)           def ll_fz_drop_pixmap(pix):
6021: (4)               r"""
6022: (4)               Low-level wrapper for `::fz_drop_pixmap()`.
6023: (4)               Decrement the reference count for the pixmap. When the
6024: (4)               reference count hits 0, the pixmap is freed.
6025: (4)               Never throws exceptions.
6026: (4)               """
6027: (4)               return _mupdf.ll_fz_drop_pixmap(pix)
6028: (0)           def ll_fz_drop_pool(pool):
6029: (4)               r"""
6030: (4)               Low-level wrapper for `::fz_drop_pool()`.
6031: (4)               Drop a pool, freeing and invalidating all storage returned from
6032: (4)               the pool.
6033: (4)               """
6034: (4)               return _mupdf.ll_fz_drop_pool(pool)
6035: (0)           def ll_fz_drop_separations(sep):
6036: (4)               r"""
6037: (4)               Low-level wrapper for `::fz_drop_separations()`.
6038: (4)               Decrement the reference count for a separations structure.
6039: (4)               When the reference count hits zero, the separations structure
6040: (4)               is freed.
6041: (4)               Never throws exceptions.
6042: (4)               """
6043: (4)               return _mupdf.ll_fz_drop_separations(sep)
6044: (0)           def ll_fz_drop_shade(shade):
6045: (4)               r"""
6046: (4)               Low-level wrapper for `::fz_drop_shade()`.
6047: (4)               Decrement the reference count for the shade structure. When
6048: (4)               the reference count hits zero, the structure is freed.
6049: (4)               Never throws exceptions.
6050: (4)               """
6051: (4)               return _mupdf.ll_fz_drop_shade(shade)
6052: (0)           def ll_fz_drop_shade_color_cache(cache):
6053: (4)               r""" Low-level wrapper for `::fz_drop_shade_color_cache()`."""
6054: (4)               return _mupdf.ll_fz_drop_shade_color_cache(cache)
6055: (0)           def ll_fz_drop_shade_imp(shade):
6056: (4)               r"""
6057: (4)               Low-level wrapper for `::fz_drop_shade_imp()`.
6058: (4)               Internal function to destroy a
6059: (4)               shade. Only exposed for use with the fz_store.
6060: (4)               shade: The reference to destroy.
6061: (4)               """
6062: (4)               return _mupdf.ll_fz_drop_shade_imp(shade)
6063: (0)           def ll_fz_drop_stext_page(page):
6064: (4)               r""" Low-level wrapper for `::fz_drop_stext_page()`."""
6065: (4)               return _mupdf.ll_fz_drop_stext_page(page)
```

```
(0)         def ll_fz_drop_storable(arg_1):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_storable()`.
(4)             Decrement the reference count for a storable object. When the
(4)             reference count hits zero, the drop function for that object
(4)             is called to free the object.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_drop_storable(arg_1)
(0)         def ll_fz_drop_store_context():
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_store_context()`.
(4)             Decrement the reference count for the store context. When the
(4)             reference count hits zero, the store context is freed.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_drop_store_context()
(0)         def ll_fz_drop_story(story):
(4)             r""" Low-level wrapper for `::fz_drop_story()`."""
(4)             return _mupdf.ll_fz_drop_story(story)
(0)         def ll_fz_drop_stream(stm):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_stream()`.
(4)             Decrements the reference count for a stream.
(4)             When the reference count for the stream hits zero, frees the
(4)             storage used for the fz_stream itself, and (usually)
(4)             releases the underlying resources that the stream is based upon
(4)             (depends on the method used to open the stream initially).
(4)             """
(4)             return _mupdf.ll_fz_drop_stream(stm)
(0)         def ll_fz_drop_string(str):
(4)             r""" Low-level wrapper for `::fz_drop_string()`."""
(4)             return _mupdf.ll_fz_drop_string(str)
(0)         def ll_fz_drop_stroke_state(stroke):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_stroke_state()`.
(4)             Drop a reference to a stroke state structure, destroying the
(4)             structure if it is the last reference.
(4)             """
(4)             return _mupdf.ll_fz_drop_stroke_state(stroke)
(0)         def ll_fz_drop_text(text):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_text()`.
(4)             Decrement the reference count for the text object. When the
(4)             reference count hits zero, the text object is freed.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_drop_text(text)
(0)         def ll_fz_drop_tree(node, dropfunc):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_tree()`.
(4)             Drop the tree.
(4)             The storage used by the tree is freed, and each value has
(4)             dropfunc called on it.
(4)             """
(4)             return _mupdf.ll_fz_drop_tree(node, dropfunc)
(0)         def ll_fz_drop_xml(xml):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_xml()`.
(4)             Drop a reference to the XML. When the last reference is
(4)             dropped, the node and all its children and siblings will
(4)             be freed.
(4)             """
(4)             return _mupdf.ll_fz_drop_xml(xml)
(0)         def ll_fz_drop_zip_writer(zip):
(4)             r"""
(4)             Low-level wrapper for `::fz_drop_zip_writer()`.
(4)             Drop the reference to the zipfile.
(4)             In common with other 'drop' methods, this will never throw an
```

```
6135: (4)               exception.
6136: (4)               """
6137: (4)               return _mupdf.ll_fz_drop_zip_writer(zip)
6138: (0)           def ll_fz_dump_glyph_cache_stats(out):
6139: (4)               r"""
6140: (4)               Low-level wrapper for `::fz_dump_glyph_cache_stats()`.
6141: (4)               Dump debug statistics for the glyph cache.
6142: (4)               """
6143: (4)               return _mupdf.ll_fz_dump_glyph_cache_stats(out)
6144: (0)           def ll_fz_duplicate_glyph_names_from_unicode(unicode):
6145: (4)               r""" Low-level wrapper for `::fz_duplicate_glyph_names_from_unicode()`."""
6146: (4)               return _mupdf.ll_fz_duplicate_glyph_names_from_unicode(unicode)
6147: (0)           def ll_fz_empty_store():
6148: (4)               r"""
6149: (4)               Low-level wrapper for `::fz_empty_store()`.
6150: (4)               Evict every item from the store.
6151: (4)               """
6152: (4)               return _mupdf.ll_fz_empty_store()
6153: (0)           def ll_fz_enable_device_hints(dev, hints):
6154: (4)               r"""
6155: (4)               Low-level wrapper for `::fz_enable_device_hints()`.
6156: (4)               Enable (set) hint bits within the hint bitfield for a device.
6157: (4)               """
6158: (4)               return _mupdf.ll_fz_enable_device_hints(dev, hints)
6159: (0)           def ll_fz_enable_icc():
6160: (4)               r"""
6161: (4)               Low-level wrapper for `::fz_enable_icc()`.
6162: (4)               Enable icc profile based operation.
6163: (4)               """
6164: (4)               return _mupdf.ll_fz_enable_icc()
6165: (0)           def ll_fz_encode_character(font, unicode):
6166: (4)               r"""
6167: (4)               Low-level wrapper for `::fz_encode_character()`.
6168: (4)               Find the glyph id for a given unicode
6169: (4)               character within a font.
6170: (4)               font: The font to look for the unicode character in.
6171: (4)               unicode: The unicode character to encode.
6172: (4)               Returns the glyph id for the given unicode value, or 0 if
6173: (4)               unknown.
6174: (4)               """
6175: (4)               return _mupdf.ll_fz_encode_character(font, unicode)
6176: (0)           def ll_fz_encode_character_by_glyph_name(font, glyphname):
6177: (4)               r"""
6178: (4)               Low-level wrapper for `::fz_encode_character_by_glyph_name()`.
6179: (4)               Encode character.
6180: (4)               Either by direct lookup of glyphname within a font, or, failing
6181: (4)               that, by mapping glyphname to unicode and thence to the glyph
6182: (4)               index within the given font.
6183: (4)               Returns zero for type3 fonts.
6184: (4)               """
6185: (4)               return _mupdf.ll_fz_encode_character_by_glyph_name(font, glyphname)
6186: (0)           def ll_fz_encode_character_sc(font, unicode):
6187: (4)               r"""
6188: (4)               Low-level wrapper for `::fz_encode_character_sc()`.
6189: (4)               Encode character, preferring small-caps variant if available.
6190: (4)               font: The font to look for the unicode character in.
6191: (4)               unicode: The unicode character to encode.
6192: (4)               Returns the glyph id for the given unicode value, or 0 if
6193: (4)               unknown.
6194: (4)               """
6195: (4)               return _mupdf.ll_fz_encode_character_sc(font, unicode)
6196: (0)           def ll_fz_encode_character_with_fallback(font, unicode, script, language,
out_font):
6197: (4)               r"""
6198: (5)                Low-level wrapper for `::fz_encode_character_with_fallback()`.
6199: (4)               This function has out-params. Python/C# wrappers look like:
6200: (8)                   `ll_fz_encode_character_with_fallback(::fz_font *font, int unicode,
int script, int language, ::fz_font **out_font)` => `(int)`
6201: (8)                   Find the glyph id for
```

```
6202: (8)                   a given unicode character within a font, falling back to
6203: (8)                   an alternative if not found.
6204: (8)                   font: The font to look for the unicode character in.
6205: (8)                   unicode: The unicode character to encode.
6206: (8)                   script: The script in use.
6207: (8)                   language: The language in use.
6208: (8)                   out_font: The font handle in which the given glyph represents
6209: (8)                   the requested unicode character. The caller does not own the
6210: (8)                   reference it is passed, so should call fz_keep_font if it is
6211: (8)                   not simply to be used immediately.
6212: (8)                   Returns the glyph id for the given unicode value in the supplied
6213: (8)                   font (and sets *out_font to font) if it is present. Otherwise
6214: (8)                   an alternative fallback font (based on script/language) is
6215: (8)                   searched for. If the glyph is found therein, *out_font is set
6216: (8)                   to this reference, and the glyph reference is returned. If it
6217: (8)                   cannot be found anywhere, the function returns 0.
6218: (4)               """
6219: (4)               return _mupdf.ll_fz_encode_character_with_fallback(font, unicode, script,
language, out_font)
6220: (0)           def ll_fz_encode_uri(s):
6221: (4)               r"""
6222: (5)                Low-level wrapper for `::fz_encode_uri()`.
6223: (4)               Return a new string representing the provided string encoded as a URI.
6224: (4)               """
6225: (4)               return _mupdf.ll_fz_encode_uri(s)
6226: (0)           def ll_fz_encode_uri_component(s):
6227: (4)               r"""
6228: (5)                Low-level wrapper for `::fz_encode_uri_component()`.
6229: (4)               Return a new string representing the provided string encoded as an URI
component.
6230: (4)               This also encodes the special reserved characters (; / ? : @ & = + $ , #).
6231: (4)               """
6232: (4)               return _mupdf.ll_fz_encode_uri_component(s)
6233: (0)           def ll_fz_encode_uri_pathname(s):
6234: (4)               r"""
6235: (5)                Low-level wrapper for `::fz_encode_uri_pathname()`.
6236: (4)               Return a new string representing the provided string encoded as an URI
path name.
6237: (4)               This also encodes the special reserved characters except /.
6238: (4)               """
6239: (4)               return _mupdf.ll_fz_encode_uri_pathname(s)
6240: (0)           def ll_fz_end_group(dev):
6241: (4)               r""" Low-level wrapper for `::fz_end_group()`."""
6242: (4)               return _mupdf.ll_fz_end_group(dev)
6243: (0)           def ll_fz_end_layer(dev):
6244: (4)               r""" Low-level wrapper for `::fz_end_layer()`."""
6245: (4)               return _mupdf.ll_fz_end_layer(dev)
6246: (0)           def ll_fz_end_mask(dev):
6247: (4)               r""" Low-level wrapper for `::fz_end_mask()`."""
6248: (4)               return _mupdf.ll_fz_end_mask(dev)
6249: (0)           def ll_fz_end_mask_tr(dev, fn):
6250: (4)               r""" Low-level wrapper for `::fz_end_mask_tr()`."""
6251: (4)               return _mupdf.ll_fz_end_mask_tr(dev, fn)
6252: (0)           def ll_fz_end_metatext(dev):
6253: (4)               r""" Low-level wrapper for `::fz_end_metatext()`."""
6254: (4)               return _mupdf.ll_fz_end_metatext(dev)
6255: (0)           def ll_fz_end_page(wri):
6256: (4)               r"""
6257: (4)               Low-level wrapper for `::fz_end_page()`.
6258: (4)               Called to end the process of writing a page to a
6259: (4)               document.
6260: (4)               """
6261: (4)               return _mupdf.ll_fz_end_page(wri)
6262: (0)           def ll_fz_end_structure(dev):
6263: (4)               r""" Low-level wrapper for `::fz_end_structure()`."""
6264: (4)               return _mupdf.ll_fz_end_structure(dev)
6265: (0)           def ll_fz_end_throw_on_repair():
6266: (4)               r""" Low-level wrapper for `::fz_end_throw_on_repair()`."""
6267: (4)               return _mupdf.ll_fz_end_throw_on_repair()
```

```
6268: (0)           def ll_fz_end_tile(dev):
6269: (4)               r""" Low-level wrapper for `::fz_end_tile()`."""
6270: (4)               return _mupdf.ll_fz_end_tile(dev)
6271: (0)           def ll_fz_error_callback(user):
6272: (4)               r"""
6273: (5)                Low-level wrapper for `::fz_error_callback()`.
6274: (4)               This function has out-params. Python/C# wrappers look like:
6275: (8)                   `ll_fz_error_callback()` => `(fz_error_cb *, void *user)`
6276: (8)                   Retrieve the currently set error callback, or NULL if none
6277: (8)                   has been set. Optionally, if user is non-NULL, the user pointer
6278: (8)                   given when the warning callback was set is also passed back to
6279: (8)                   the caller.
6280: (4)               """
6281: (4)               return _mupdf.ll_fz_error_callback(user)
6282: (0)           def ll_fz_eval_function(func, _in, inlen, out, outlen):
6283: (4)               r"""
6284: (5)                Low-level wrapper for `::fz_eval_function()`.
6285: (4)               This function has out-params. Python/C# wrappers look like:
6286: (8)                   `ll_fz_eval_function(::fz_function *func, const float *in, int inlen,
int outlen)` => float out
6287: (4)               """
6288: (4)               return _mupdf.ll_fz_eval_function(func, _in, inlen, out, outlen)
6289: (0)           def ll_fz_expand_irect(a, expand):
6290: (4)               r""" Low-level wrapper for `::fz_expand_irect()`."""
6291: (4)               return _mupdf.ll_fz_expand_irect(a, expand)
6292: (0)           def ll_fz_expand_rect(b, expand):
6293: (4)               r"""
6294: (4)               Low-level wrapper for `::fz_expand_rect()`.
6295: (4)               Expand a bbox by a given amount in all directions.
6296: (4)               """
6297: (4)               return _mupdf.ll_fz_expand_rect(b, expand)
6298: (0)           def ll_fz_extract_ttf_from_ttc(font):
6299: (4)               r""" Low-level wrapper for `::fz_extract_ttf_from_ttc()`."""
6300: (4)               return _mupdf.ll_fz_extract_ttf_from_ttc(font)
6301: (0)           def ll_fz_file_exists(path):
6302: (4)               r"""
6303: (4)               Low-level wrapper for `::fz_file_exists()`.
6304: (4)               Return true if the named file exists and is readable.
6305: (4)               """
6306: (4)               return _mupdf.ll_fz_file_exists(path)
6307: (0)           def ll_fz_fill_image(dev, image, ctm, alpha, color_params):
6308: (4)               r""" Low-level wrapper for `::fz_fill_image()`."""
6309: (4)               return _mupdf.ll_fz_fill_image(dev, image, ctm, alpha, color_params)
6310: (0)           def ll_fz_fill_image_mask(dev, image, ctm, colorspace, color, alpha,
color_params):
6311: (4)               r""" Low-level wrapper for `::fz_fill_image_mask()`."""
6312: (4)               return _mupdf.ll_fz_fill_image_mask(dev, image, ctm, colorspace, color,
alpha, color_params)
6313: (0)           def ll_fz_fill_path(dev, path, even_odd, ctm, colorspace, color, alpha,
color_params):
6314: (4)               r"""
6315: (4)               Low-level wrapper for `::fz_fill_path()`.
6316: (4)               Device calls; graphics primitives and containers.
6317: (4)               """
6318: (4)               return _mupdf.ll_fz_fill_path(dev, path, even_odd, ctm, colorspace, color,
alpha, color_params)
6319: (0)           def ll_fz_fill_pixmap_from_display_list(list, ctm, pix):
6320: (4)               r""" Low-level wrapper for `::fz_fill_pixmap_from_display_list()`."""
6321: (4)               return _mupdf.ll_fz_fill_pixmap_from_display_list(list, ctm, pix)
6322: (0)           def ll_fz_fill_pixmap_with_color(pix, colorspace, color, color_params):
6323: (4)               r"""
6324: (5)                Low-level wrapper for `::fz_fill_pixmap_with_color()`.
6325: (4)               This function has out-params. Python/C# wrappers look like:
6326: (8)                   `ll_fz_fill_pixmap_with_color(::fz_pixmap *pix, ::fz_colorspace
*colorspace, ::fz_color_params color_params)` => float color
6327: (8)                   Fill pixmap with solid color.
6328: (4)               """
6329: (4)               return _mupdf.ll_fz_fill_pixmap_with_color(pix, colorspace, color,
color_params)
```

```
6330: (0)           def ll_fz_fill_shade(dev, shade, ctm, alpha, color_params):
6331: (4)               r""" Low-level wrapper for `::fz_fill_shade()`."""
6332: (4)               return _mupdf.ll_fz_fill_shade(dev, shade, ctm, alpha, color_params)
6333: (0)           def ll_fz_fill_text(dev, text, ctm, colorspace, color, alpha, color_params):
6334: (4)               r""" Low-level wrapper for `::fz_fill_text()`."""
6335: (4)               return _mupdf.ll_fz_fill_text(dev, text, ctm, colorspace, color, alpha,
color_params)
6336: (0)           def ll_fz_filter_store(fn, arg, type):
6337: (4)               r"""
6338: (4)               Low-level wrapper for `::fz_filter_store()`.
6339: (4)               Filter every element in the store with a matching type with the
6340: (4)               given function.
6341: (4)               If the function returns 1 for an element, drop the element.
6342: (4)               """
6343: (4)               return _mupdf.ll_fz_filter_store(fn, arg, type)
6344: (0)           def ll_fz_find_item(drop, key, type):
6345: (4)               r"""
6346: (4)               Low-level wrapper for `::fz_find_item()`.
6347: (4)               Find an item within the store.
6348: (4)               drop: The function used to free the value (to ensure we get a
6349: (4)               value of the correct type).
6350: (4)               key: The key used to index the item.
6351: (4)               type: Functions used to manipulate the key.
6352: (4)               Returns NULL for not found, otherwise returns a pointer to the
6353: (4)               value indexed by key to which a reference has been taken.
6354: (4)               """
6355: (4)               return _mupdf.ll_fz_find_item(drop, key, type)
6356: (0)           def ll_fz_flush_output(out):
6357: (4)               r"""
6358: (4)               Low-level wrapper for `::fz_flush_output()`.
6359: (4)               Flush unwritten data.
6360: (4)               """
6361: (4)               return _mupdf.ll_fz_flush_output(out)
6362: (0)           def ll_fz_flush_warnings():
6363: (4)               r"""
6364: (4)               Low-level wrapper for `::fz_flush_warnings()`.
6365: (4)               Flush any repeated warnings.
6366: (4)               Repeated warnings are buffered, counted and eventually printed
6367: (4)               along with the number of repetitions. Call fz_flush_warnings
6368: (4)               to force printing of the latest buffered warning and the
6369: (4)               number of repetitions, for example to make sure that all
6370: (4)               warnings are printed before exiting an application.
6371: (4)               """
6372: (4)               return _mupdf.ll_fz_flush_warnings()
6373: (0)           def ll_fz_font_ascender(font):
6374: (4)               r"""
6375: (4)               Low-level wrapper for `::fz_font_ascender()`.
6376: (4)               Retrieve font ascender in ems.
6377: (4)               """
6378: (4)               return _mupdf.ll_fz_font_ascender(font)
6379: (0)           def ll_fz_font_bbox(font):
6380: (4)               r"""
6381: (4)               Low-level wrapper for `::fz_font_bbox()`.
6382: (4)               Retrieve the font bbox.
6383: (4)               font: The font to query.
6384: (4)               Returns the font bbox by value; it is valid only if
6385: (4)               fz_font_flags(font)->invalid_bbox is zero.
6386: (4)               """
6387: (4)               return _mupdf.ll_fz_font_bbox(font)
6388: (0)           def ll_fz_font_descender(font):
6389: (4)               r"""
6390: (4)               Low-level wrapper for `::fz_font_descender()`.
6391: (4)               Retrieve font descender in ems.
6392: (4)               """
6393: (4)               return _mupdf.ll_fz_font_descender(font)
6394: (0)           def ll_fz_font_digest(font, digest):
6395: (4)               r"""
6396: (4)               Low-level wrapper for `::fz_font_digest()`.
6397: (4)               Retrieve the MD5 digest for the font's data.
```

```
6398: (4)                """
6399: (4)                return _mupdf.ll_fz_font_digest(font, digest)
6400: (0)            def ll_fz_font_flags(font):
6401: (4)                r"""
6402: (4)                Low-level wrapper for `::fz_font_flags()`.
6403: (4)                Retrieve a pointer to the font flags
6404: (4)                for a given font. These can then be updated as required.
6405: (4)                font: The font to query
6406: (4)                Returns a pointer to the flags structure (or NULL, if
6407: (4)                the font is NULL).
6408: (4)                """
6409: (4)                return _mupdf.ll_fz_font_flags(font)
6410: (0)            def ll_fz_font_ft_face(font):
6411: (4)                r"""
6412: (4)                Low-level wrapper for `::fz_font_ft_face()`.
6413: (4)                Retrieve the FT_Face handle
6414: (4)                for the font.
6415: (4)                font: The font to query
6416: (4)                Returns the FT_Face handle for the font, or NULL
6417: (4)                if not a freetype handled font. (Cast to void *
6418: (4)                to avoid nasty header exposure).
6419: (4)                """
6420: (4)                return _mupdf.ll_fz_font_ft_face(font)
6421: (0)            def ll_fz_font_is_bold(font):
6422: (4)                r"""
6423: (4)                Low-level wrapper for `::fz_font_is_bold()`.
6424: (4)                Query whether the font flags say that this font is bold.
6425: (4)                """
6426: (4)                return _mupdf.ll_fz_font_is_bold(font)
6427: (0)            def ll_fz_font_is_italic(font):
6428: (4)                r"""
6429: (4)                Low-level wrapper for `::fz_font_is_italic()`.
6430: (4)                Query whether the font flags say that this font is italic.
6431: (4)                """
6432: (4)                return _mupdf.ll_fz_font_is_italic(font)
6433: (0)            def ll_fz_font_is_monospaced(font):
6434: (4)                r"""
6435: (4)                Low-level wrapper for `::fz_font_is_monospaced()`.
6436: (4)                Query whether the font flags say that this font is monospaced.
6437: (4)                """
6438: (4)                return _mupdf.ll_fz_font_is_monospaced(font)
6439: (0)            def ll_fz_font_is_serif(font):
6440: (4)                r"""
6441: (4)                Low-level wrapper for `::fz_font_is_serif()`.
6442: (4)                Query whether the font flags say that this font is serif.
6443: (4)                """
6444: (4)                return _mupdf.ll_fz_font_is_serif(font)
6445: (0)            def ll_fz_font_name(font):
6446: (4)                r"""
6447: (4)                Low-level wrapper for `::fz_font_name()`.
6448: (4)                Retrieve a pointer to the name of the font.
6449: (4)                font: The font to query.
6450: (4)                Returns a pointer to an internal copy of the font name.
6451: (4)                Will never be NULL, but may be the empty string.
6452: (4)                """
6453: (4)                return _mupdf.ll_fz_font_name(font)
6454: (0)            def ll_fz_font_shaper_data(font):
6455: (4)                r"""
6456: (4)                Low-level wrapper for `::fz_font_shaper_data()`.
6457: (4)                Retrieve a pointer to the shaper data
6458: (4)                structure for the given font.
6459: (4)                font: The font to query.
6460: (4)                Returns a pointer to the shaper data structure (or NULL if
6461: (4)                font is NULL).
6462: (4)                """
6463: (4)                return _mupdf.ll_fz_font_shaper_data(font)
6464: (0)            def ll_fz_font_t3_procs(font):
6465: (4)                r"""
6466: (4)                Low-level wrapper for `::fz_font_t3_procs()`.
```

```
6467: (4)               Retrieve the Type3 procs
6468: (4)               for a font.
6469: (4)               font: The font to query
6470: (4)               Returns the t3_procs pointer. Will be NULL for a
6471: (4)               non type-3 font.
6472: (4)               """
6473: (4)               return _mupdf.ll_fz_font_t3_procs(font)
6474: (0)           def ll_fz_format_double(fmt, value):
6475: (4)               r"""
6476: (5)                Low-level wrapper for `::fz_format_double()`.  Swig-friendly and typesafe
way to do fz_snprintf(fmt, value). `fmt`
6477: (4)               must end with one of 'efg' otherwise we throw an exception.
6478: (4)               """
6479: (4)               return _mupdf.ll_fz_format_double(fmt, value)
6480: (0)           def ll_fz_format_link_uri(doc, dest):
6481: (4)               r"""
6482: (4)               Low-level wrapper for `::fz_format_link_uri()`.
6483: (4)               Format an internal link to a page number, location, and possible viewing
parameters,
6484: (4)               suitable for use with fz_create_link.
6485: (4)               Returns a newly allocated string that the caller must free.
6486: (4)               """
6487: (4)               return _mupdf.ll_fz_format_link_uri(doc, dest)
6488: (0)           def ll_fz_format_output_path(path, size, fmt, page):
6489: (4)               r"""
6490: (4)               Low-level wrapper for `::fz_format_output_path()`.
6491: (4)               create output file name using a template.
6492: (4)               If the path contains %[0-9]*d, the first such pattern will be
6493: (4)               replaced with the page number. If the template does not contain
6494: (4)               such a pattern, the page number will be inserted before the
6495: (4)               filename extension. If the template does not have a filename
6496: (4)               extension, the page number will be added to the end.
6497: (4)               """
6498: (4)               return _mupdf.ll_fz_format_output_path(path, size, fmt, page)
6499: (0)           def ll_fz_free(p):
6500: (4)               r"""
6501: (4)               Low-level wrapper for `::fz_free()`.
6502: (4)               Free a previously allocated block of memory.
6503: (4)               fz_free(ctx, NULL) does nothing.
6504: (4)               Never throws exceptions.
6505: (4)               """
6506: (4)               return _mupdf.ll_fz_free(p)
6507: (0)           def ll_fz_ft_lock():
6508: (4)               r""" Low-level wrapper for `::fz_ft_lock()`."""
6509: (4)               return _mupdf.ll_fz_ft_lock()
6510: (0)           def ll_fz_ft_lock_held():
6511: (4)               r""" Low-level wrapper for `::fz_ft_lock_held()`."""
6512: (4)               return _mupdf.ll_fz_ft_lock_held()
6513: (0)           def ll_fz_ft_unlock():
6514: (4)               r""" Low-level wrapper for `::fz_ft_unlock()`."""
6515: (4)               return _mupdf.ll_fz_ft_unlock()
6516: (0)           def ll_fz_function_size(func):
6517: (4)               r""" Low-level wrapper for `::fz_function_size()`."""
6518: (4)               return _mupdf.ll_fz_function_size(func)
6519: (0)           def ll_fz_gamma_pixmap(pix, gamma):
6520: (4)               r"""
6521: (4)               Low-level wrapper for `::fz_gamma_pixmap()`.
6522: (4)               Apply gamma correction to a pixmap. All components
6523: (4)               of all pixels are modified (except alpha, which is unchanged).
6524: (4)               gamma: The gamma value to apply; 1.0 for no change.
6525: (4)               """
6526: (4)               return _mupdf.ll_fz_gamma_pixmap(pix, gamma)
6527: (0)           def ll_fz_generate_transition(tpix, opix, npix, time, trans):
6528: (4)               r"""
6529: (4)               Low-level wrapper for `::fz_generate_transition()`.
6530: (4)               Generate a frame of a transition.
6531: (4)               tpix: Target pixmap
6532: (4)               opix: Old pixmap
6533: (4)               npix: New pixmap
```

```
6534: (4)               time: Position within the transition (0 to 256)
6535: (4)               trans: Transition details
6536: (4)               Returns 1 if successfully generated a frame.
6537: (4)               Note: Pixmaps must include alpha.
6538: (4)               """
6539: (4)               return _mupdf.ll_fz_generate_transition(tpix, opix, npix, time, trans)
6540: (0)           def ll_fz_get_glyph_name(font, glyph, buf, size):
6541: (4)               r"""
6542: (4)               Low-level wrapper for `::fz_get_glyph_name()`.
6543: (4)               Find the name of a glyph
6544: (4)               font: The font to look for the glyph in.
6545: (4)               glyph: The glyph id to look for.
6546: (4)               buf: Pointer to a buffer for the name to be inserted into.
6547: (4)               size: The size of the buffer.
6548: (4)               If a font contains a name table, then the name of the glyph
6549: (4)               will be returned in the supplied buffer. Otherwise a name
6550: (4)               is synthesised. The name will be truncated to fit in
6551: (4)               the buffer.
6552: (4)               """
6553: (4)               return _mupdf.ll_fz_get_glyph_name(font, glyph, buf, size)
6554: (0)           def ll_fz_get_glyph_name2(font, glyph):
6555: (4)               r"""
6556: (5)                Low-level wrapper for `::fz_get_glyph_name2()`.
6557: (4)               C++ alternative to fz_get_glyph_name() that returns information in a
std::string.
6558: (4)               """
6559: (4)               return _mupdf.ll_fz_get_glyph_name2(font, glyph)
6560: (0)           def ll_fz_get_pixmap_from_image(image, subarea, ctm, w, h):
6561: (4)               r"""
6562: (5)                Low-level wrapper for `::fz_get_pixmap_from_image()`.
6563: (4)               This function has out-params. Python/C# wrappers look like:
6564: (8)                   `ll_fz_get_pixmap_from_image(::fz_image *image, const ::fz_irect
*subarea, ::fz_matrix *ctm)` => `(fz_pixmap *, int w, int h)`
6565: (8)                   Called to get a handle to a pixmap from an image.
6566: (8)                   image: The image to retrieve a pixmap from.
6567: (8)                   subarea: The subarea of the image that we actually care about
6568: (8)                   (or NULL to indicate the whole image).
6569: (8)                   ctm: Optional, unless subarea is given. If given, then on
6570: (8)                   entry this is the transform that will be applied to the complete
6571: (8)                   image. It should be updated on exit to the transform to apply to
6572: (8)                   the given subarea of the image. This is used to calculate the
6573: (8)                   desired width/height for subsampling.
6574: (8)                   w: If non-NULL, a pointer to an int to be updated on exit to the
6575: (8)                   width (in pixels) that the scaled output will cover.
6576: (8)                   h: If non-NULL, a pointer to an int to be updated on exit to the
6577: (8)                   height (in pixels) that the scaled output will cover.
6578: (8)                   Returns a non NULL kept pixmap pointer. May throw exceptions.
6579: (4)               """
6580: (4)               return _mupdf.ll_fz_get_pixmap_from_image(image, subarea, ctm, w, h)
6581: (0)           def ll_fz_get_unscaled_pixmap_from_image(image):
6582: (4)               r"""
6583: (4)               Low-level wrapper for `::fz_get_unscaled_pixmap_from_image()`.
6584: (4)               Calls fz_get_pixmap_from_image() with ctm, subarea, w and h all set to
NULL.
6585: (4)               """
6586: (4)               return _mupdf.ll_fz_get_unscaled_pixmap_from_image(image)
6587: (0)           def ll_fz_getopt(nargc, nargv, ostr):
6588: (4)               r"""
6589: (5)                Low-level wrapper for `::fz_getopt()`.
6590: (4)               This function has out-params. Python/C# wrappers look like:
6591: (8)                   `ll_fz_getopt(int nargc, const char *ostr)` => `(int, char *nargv)`
6592: (8)                   Identical to fz_getopt_long, but with a NULL longopts field,
signifying no long
6593: (8)                   options.
6594: (4)               """
6595: (4)               return _mupdf.ll_fz_getopt(nargc, nargv, ostr)
6596: (0)           def ll_fz_getopt_long(nargc, nargv, ostr, longopts):
6597: (4)               r"""
6598: (5)                Low-level wrapper for `::fz_getopt_long()`.
```

```
6599: (4)               This function has out-params. Python/C# wrappers look like:
6600: (8)                   `ll_fz_getopt_long(int nargc, const char *ostr, const
::fz_getopt_long_options *longopts)` => `(int, char *nargv)`
6601: (8)                   Simple functions/variables for use in tools.
6602: (8)                   ostr = option string. Comprises single letter options, followed by :
if there
6603: (8)                   is an argument to the option.
6604: (8)                   longopts: NULL (indicating no long options), or a pointer to an array
of
6605: (8)                   longoptions, terminated by an entry with option == NULL.
6606: (8)                   In the event of matching a single char option, this function will
normally
6607: (8)                   return the char. The exception to this is when the option requires an
6608: (8)                   argument and none is supplied; in this case we return ':'.
6609: (8)                   In the event of matching a long option, this function returns 0, with
fz_optlong
6610: (8)                   set to point to the matching option.
6611: (8)                   A long option entry may be followed with : to indicate there is an
argument to the
6612: (8)                   option. If the need for an argument is specified in this way, and no
argument is
6613: (8)                   given, an error will be displayed and argument processing will stop.
If an argument
6614: (8)                   is given, and the long option record contains a non-null flag pointer,
then the code
6615: (8)                   will decode the argument and fill in that flag pointer. Specifically,
6616: (8)                   case-insensitive matches to 'yes', 'no', 'true' and 'false' will cause
a value of 0
6617: (8)                   or 1 as appropriate to be written; failing this the arg will be
interpreted as a
6618: (8)                   decimal integer using atoi.
6619: (8)                   A long option entry may be followed by an list of options (e.g.
myoption=foo|bar|baz)
6620: (8)                   and the option will be passed to fz_opt_from_list. The return value of
that will be
6621: (8)                   placed in fz_optitem. If the return value of that function is -1, then
an error will
6622: (8)                   be displayed and argument processing will stop.
6623: (8)                   In the event of reaching the end of the arg list or '--', this
function returns EOF.
6624: (8)                   In the event of failing to match anything, an error is printed, and we
return '?'.
6625: (8)                   If an argument is expected for the option, then fz_optarg will be
returned pointing
6626: (8)                   at the start of the argument. Examples of supported argument formats:
'-r500', '-r 500',
6627: (8)                   '--resolution 500', '--resolution=500'.
6628: (4)               """
6629: (4)               return _mupdf.ll_fz_getopt_long(nargc, nargv, ostr, longopts)
6630: (0)           def ll_fz_glyph_bbox(glyph):
6631: (4)               r"""
6632: (4)               Low-level wrapper for `::fz_glyph_bbox()`.
6633: (4)               Return the bounding box of the glyph in pixels.
6634: (4)               """
6635: (4)               return _mupdf.ll_fz_glyph_bbox(glyph)
6636: (0)           def ll_fz_glyph_bbox_no_ctx(src):
6637: (4)               r""" Low-level wrapper for `::fz_glyph_bbox_no_ctx()`."""
6638: (4)               return _mupdf.ll_fz_glyph_bbox_no_ctx(src)
6639: (0)           def ll_fz_glyph_cacheable(font, gid):
6640: (4)               r"""
6641: (4)               Low-level wrapper for `::fz_glyph_cacheable()`.
6642: (4)               Determine if a given glyph in a font
6643: (4)               is cacheable. Certain glyphs in a type 3 font cannot safely
6644: (4)               be cached, as their appearance depends on the enclosing
6645: (4)               graphic state.
6646: (4)               font: The font to look for the glyph in.
6647: (4)               gif: The glyph to query.
6648: (4)               Returns non-zero if cacheable, 0 if not.
6649: (4)               """
```

```
6650: (4)               return _mupdf.ll_fz_glyph_cacheable(font, gid)
6651: (0)           def ll_fz_glyph_height(glyph):
6652: (4)               r"""
6653: (4)               Low-level wrapper for `::fz_glyph_height()`.
6654: (4)               Return the height of the glyph in pixels.
6655: (4)               """
6656: (4)               return _mupdf.ll_fz_glyph_height(glyph)
6657: (0)           def ll_fz_glyph_name_from_unicode_sc(unicode):
6658: (4)               r""" Low-level wrapper for `::fz_glyph_name_from_unicode_sc()`."""
6659: (4)               return _mupdf.ll_fz_glyph_name_from_unicode_sc(unicode)
6660: (0)           def ll_fz_glyph_width(glyph):
6661: (4)               r"""
6662: (4)               Low-level wrapper for `::fz_glyph_width()`.
6663: (4)               Return the width of the glyph in pixels.
6664: (4)               """
6665: (4)               return _mupdf.ll_fz_glyph_width(glyph)
6666: (0)           def ll_fz_graphics_aa_level():
6667: (4)               r"""
6668: (4)               Low-level wrapper for `::fz_graphics_aa_level()`.
6669: (4)               Get the number of bits of antialiasing we are
6670: (4)               using for graphics. Between 0 and 8.
6671: (4)               """
6672: (4)               return _mupdf.ll_fz_graphics_aa_level()
6673: (0)           def ll_fz_graphics_min_line_width():
6674: (4)               r"""
6675: (4)               Low-level wrapper for `::fz_graphics_min_line_width()`.
6676: (4)               Get the minimum line width to be
6677: (4)               used for stroked lines.
6678: (4)               min_line_width: The minimum line width to use (in pixels).
6679: (4)               """
6680: (4)               return _mupdf.ll_fz_graphics_min_line_width()
6681: (0)           def ll_fz_gridfit_matrix(as_tiled, m):
6682: (4)               r"""
6683: (4)               Low-level wrapper for `::fz_gridfit_matrix()`.
6684: (4)               Grid fit a matrix.
6685: (4)               as_tiled = 0 => adjust the matrix so that the image of the unit
6686: (4)               square completely covers any pixel that was touched by the
6687: (4)               image of the unit square under the original matrix.
6688: (4)               as_tiled = 1 => adjust the matrix so that the corners of the
6689: (4)               image of the unit square align with the closest integer corner
6690: (4)               of the image of the unit square under the original matrix.
6691: (4)               """
6692: (4)               return _mupdf.ll_fz_gridfit_matrix(as_tiled, m)
6693: (0)           def ll_fz_grisu(f, s, exp):
6694: (4)               r"""
6695: (5)                Low-level wrapper for `::fz_grisu()`.
6696: (4)               This function has out-params. Python/C# wrappers look like:
6697: (8)                   `ll_fz_grisu(float f, char *s)` => `(int, int exp)`
6698: (4)               """
6699: (4)               return _mupdf.ll_fz_grisu(f, s, exp)
6700: (0)           def ll_fz_grow_buffer(buf):
6701: (4)               r"""
6702: (4)               Low-level wrapper for `::fz_grow_buffer()`.
6703: (4)               Make some space within a buffer (i.e. ensure that
6704: (4)               capacity > size).
6705: (4)               """
6706: (4)               return _mupdf.ll_fz_grow_buffer(buf)
6707: (0)           def ll_fz_has_archive_entry(arch, name):
6708: (4)               r"""
6709: (4)               Low-level wrapper for `::fz_has_archive_entry()`.
6710: (4)               Check if entry by given name exists.
6711: (4)               If named entry does not exist 0 will be returned, if it does
6712: (4)               exist 1 is returned.
6713: (4)               name: Entry name to look for, this must be an exact match to
6714: (4)               the entry name in the archive.
6715: (4)               """
6716: (4)               return _mupdf.ll_fz_has_archive_entry(arch, name)
6717: (0)           def ll_fz_has_option(opts, key, val):
6718: (4)               r"""
```

```
(5)                 Low-level wrapper for `::fz_has_option()`.
(4)                This function has out-params. Python/C# wrappers look like:
(8)                    `ll_fz_has_option(const char *opts, const char *key)` => `(int, const char *val)`
(8)                    Look for a given option (key) in the opts string. Return 1 if
(8)                    it has it, and update *val to point to the value within opts.
(4)                """
(4)                return _mupdf.ll_fz_has_option(opts, key, val)
(0)            def ll_fz_has_permission(doc, p):
(4)                r"""
(4)                Low-level wrapper for `::fz_has_permission()`.
(4)                Check permission flags on document.
(4)                """
(4)                return _mupdf.ll_fz_has_permission(doc, p)
(0)            def ll_fz_hash_filter(table, state, callback):
(4)                r"""
(4)                Low-level wrapper for `::fz_hash_filter()`.
(4)                Iterate over the entries in a hash table, removing all the ones where callback returns true.
(4)                Does NOT free the value of the entry, so the caller is expected to take care of this.
(4)                """
(4)                return _mupdf.ll_fz_hash_filter(table, state, callback)
(0)            def ll_fz_hash_find(table, key):
(4)                r"""
(4)                Low-level wrapper for `::fz_hash_find()`.
(4)                Search for a matching hash within the table, and return the
(4)                associated value.
(4)                """
(4)                return _mupdf.ll_fz_hash_find(table, key)
(0)            def ll_fz_hash_for_each(table, state, callback):
(4)                r"""
(4)                Low-level wrapper for `::fz_hash_for_each()`.
(4)                Iterate over the entries in a hash table.
(4)                """
(4)                return _mupdf.ll_fz_hash_for_each(table, state, callback)
(0)            def ll_fz_hash_insert(table, key, val):
(4)                r"""
(4)                Low-level wrapper for `::fz_hash_insert()`.
(4)                Insert a new key/value pair into the hash table.
(4)                If an existing entry with the same key is found, no change is
(4)                made to the hash table, and a pointer to the existing value is
(4)                returned.
(4)                If no existing entry with the same key is found, ownership of
(4)                val passes in, key is copied, and NULL is returned.
(4)                """
(4)                return _mupdf.ll_fz_hash_insert(table, key, val)
(0)            def ll_fz_hash_remove(table, key):
(4)                r"""
(4)                Low-level wrapper for `::fz_hash_remove()`.
(4)                Remove the entry for a given key.
(4)                The value is NOT freed, so the caller is expected to take care
(4)                of this.
(4)                """
(4)                return _mupdf.ll_fz_hash_remove(table, key)
(0)            def ll_fz_hb_lock():
(4)                r"""
(4)                Low-level wrapper for `::fz_hb_lock()`.
(4)                Lock against Harfbuzz being called
(4)                simultaneously in several threads. This reuses
(4)                FZ_LOCK_FREETYPE.
(4)                """
(4)                return _mupdf.ll_fz_hb_lock()
(0)            def ll_fz_hb_unlock():
(4)                r"""
(4)                Low-level wrapper for `::fz_hb_unlock()`.
(4)                Unlock after a Harfbuzz call. This reuses
(4)                FZ_LOCK_FREETYPE.
(4)                """
```

```
6785: (4)                return _mupdf.ll_fz_hb_unlock()
6786: (0)            def ll_fz_highlight_selection(page, a, b, quads, max_quads):
6787: (4)                r"""
6788: (4)                Low-level wrapper for `::fz_highlight_selection()`.
6789: (4)                Return a list of quads to highlight lines inside the selection
6790: (4)                points.
6791: (4)                """
6792: (4)                return _mupdf.ll_fz_highlight_selection(page, a, b, quads, max_quads)
6793: (0)            def ll_fz_highlight_selection2(page, a, b, max_quads):
6794: (4)                r"""
6795: (5)                 Low-level wrapper for `::fz_highlight_selection2()`.
6796: (4)                C++ alternative to fz_highlight_selection() that returns quads in a
6797: (4)                std::vector.
6798: (4)                """
6799: (4)                return _mupdf.ll_fz_highlight_selection2(page, a, b, max_quads)
6800: (0)            def ll_fz_ignore_error():
6801: (4)                r""" Low-level wrapper for `::fz_ignore_error()`."""
6802: (4)                return _mupdf.ll_fz_ignore_error()
6803: (0)            def ll_fz_ignore_text(dev, text, ctm):
6804: (4)                r""" Low-level wrapper for `::fz_ignore_text()`."""
6805: (4)                return _mupdf.ll_fz_ignore_text(dev, text, ctm)
6806: (0)            def ll_fz_image_orientation(image):
6807: (4)                r"""
6808: (4)                Low-level wrapper for `::fz_image_orientation()`.
6809: (4)                Request the natural orientation of an image.
6810: (4)                This is for images (such as JPEG) that can contain internal
6811: (4)                specifications of rotation/flips. This is ignored by all the
6812: (4)                internal decode/rendering routines, but can be used by callers
6813: (4)                (such as the image document handler) to respect such
6814: (4)                specifications.
6815: (4)                The values used by MuPDF are as follows, with the equivalent
6816: (4)                Exif specifications given for information:
6817: (4)                0: Undefined
6818: (4)                1: 0 degree ccw rotation. (Exif = 1)
6819: (4)                2: 90 degree ccw rotation. (Exif = 8)
6820: (4)                3: 180 degree ccw rotation. (Exif = 3)
6821: (4)                4: 270 degree ccw rotation. (Exif = 6)
6822: (4)                5: flip on X. (Exif = 2)
6823: (4)                6: flip on X, then rotate ccw by 90 degrees. (Exif = 5)
6824: (4)                7: flip on X, then rotate ccw by 180 degrees. (Exif = 4)
6825: (4)                8: flip on X, then rotate ccw by 270 degrees. (Exif = 7)
6826: (4)                """
6827: (4)                return _mupdf.ll_fz_image_orientation(image)
6828: (0)            def ll_fz_image_orientation_matrix(image):
6829: (4)                r""" Low-level wrapper for `::fz_image_orientation_matrix()`."""
6830: (4)                return _mupdf.ll_fz_image_orientation_matrix(image)
6831: (0)            def ll_fz_image_resolution(image, xres, yres):
6832: (4)                r"""
6833: (5)                 Low-level wrapper for `::fz_image_resolution()`.
6834: (4)                This function has out-params. Python/C# wrappers look like:
6835: (8)                    `ll_fz_image_resolution(::fz_image *image)` => `(int xres, int yres)`
6836: (8)                    Request the natural resolution
6837: (8)                    of an image.
6838: (8)                    xres, yres: Pointers to ints to be updated with the
6839: (8)                    natural resolution of an image (or a sensible default
6840: (8)                    if not encoded).
6841: (4)                """
6842: (4)                return _mupdf.ll_fz_image_resolution(image, xres, yres)
6843: (0)            def ll_fz_image_size(im):
6844: (4)                r"""
6845: (4)                Low-level wrapper for `::fz_image_size()`.
6846: (4)                Return the size of the storage used by an image.
6847: (4)                """
6848: (4)                return _mupdf.ll_fz_image_size(im)
6849: (0)            def ll_fz_image_type_name(type):
6850: (4)                r"""
6851: (4)                Low-level wrapper for `::fz_image_type_name()`.
6852: (4)                Map from FZ_IMAGE_* value to string.
6853: (4)                The returned string is static and therefore must not be freed.
```

```
6854: (4)                """
6855: (4)                return _mupdf.ll_fz_image_type_name(type)
6856: (0)            def ll_fz_include_point_in_rect(r, p):
6857: (4)                r"""
6858: (4)                Low-level wrapper for `::fz_include_point_in_rect()`.
6859: (4)                Expand a bbox to include a given point.
6860: (4)                To create a rectangle that encompasses a sequence of points, the
6861: (4)                rectangle must first be set to be the empty rectangle at one of
6862: (4)                the points before including the others.
6863: (4)                """
6864: (4)                return _mupdf.ll_fz_include_point_in_rect(r, p)
6865: (0)            def ll_fz_init_text_decoder(dec, encoding):
6866: (4)                r""" Low-level wrapper for `::fz_init_text_decoder()`."""
6867: (4)                return _mupdf.ll_fz_init_text_decoder(dec, encoding)
6868: (0)            def ll_fz_install_load_system_font_funcs(f, f_cjk, f_fallback):
6869: (4)                r"""
6870: (4)                Low-level wrapper for `::fz_install_load_system_font_funcs()`.
6871: (4)                Install functions to allow MuPDF to request fonts from the
6872: (4)                system.
6873: (4)                Only one set of hooks can be in use at a time.
6874: (4)                """
6875: (4)                return _mupdf.ll_fz_install_load_system_font_funcs(f, f_cjk, f_fallback)
6876: (0)            def ll_fz_install_load_system_font_funcs2(args):
6877: (4)                r"""
6878: (5)                 Low-level wrapper for `::fz_install_load_system_font_funcs2()`.
6879: (4)                Alternative to fz_install_load_system_font_funcs() that takes args in a
6880: (4)                struct, to allow use from Python/C# via Swig Directors.
6881: (4)                """
6882: (4)                return _mupdf.ll_fz_install_load_system_font_funcs2(args)
6883: (0)            def ll_fz_int2_heap_insert(heap, v):
6884: (4)                r""" Low-level wrapper for `::fz_int2_heap_insert()`."""
6885: (4)                return _mupdf.ll_fz_int2_heap_insert(heap, v)
6886: (0)            def ll_fz_int2_heap_sort(heap):
6887: (4)                r""" Low-level wrapper for `::fz_int2_heap_sort()`."""
6888: (4)                return _mupdf.ll_fz_int2_heap_sort(heap)
6889: (0)            def ll_fz_int2_heap_uniq(heap):
6890: (4)                r""" Low-level wrapper for `::fz_int2_heap_uniq()`."""
6891: (4)                return _mupdf.ll_fz_int2_heap_uniq(heap)
6892: (0)            def ll_fz_int_heap_insert(heap, v):
6893: (4)                r""" Low-level wrapper for `::fz_int_heap_insert()`."""
6894: (4)                return _mupdf.ll_fz_int_heap_insert(heap, v)
6895: (0)            def ll_fz_int_heap_sort(heap):
6896: (4)                r""" Low-level wrapper for `::fz_int_heap_sort()`."""
6897: (4)                return _mupdf.ll_fz_int_heap_sort(heap)
6898: (0)            def ll_fz_int_heap_uniq(heap):
6899: (4)                r""" Low-level wrapper for `::fz_int_heap_uniq()`."""
6900: (4)                return _mupdf.ll_fz_int_heap_uniq(heap)
6901: (0)            def ll_fz_intersect_irect(a, b):
6902: (4)                r"""
6903: (4)                Low-level wrapper for `::fz_intersect_irect()`.
6904: (4)                Compute intersection of two bounding boxes.
6905: (4)                Similar to fz_intersect_rect but operates on two bounding
6906: (4)                boxes instead of two rectangles.
6907: (4)                """
6908: (4)                return _mupdf.ll_fz_intersect_irect(a, b)
6909: (0)            def ll_fz_intersect_rect(a, b):
6910: (4)                r"""
6911: (4)                Low-level wrapper for `::fz_intersect_rect()`.
6912: (4)                Compute intersection of two rectangles.
6913: (4)                Given two rectangles, update the first to be the smallest
6914: (4)                axis-aligned rectangle that covers the area covered by both
6915: (4)                given rectangles. If either rectangle is empty then the
6916: (4)                intersection is also empty. If either rectangle is infinite
6917: (4)                then the intersection is simply the non-infinite rectangle.
6918: (4)                Should both rectangles be infinite, then the intersection is
6919: (4)                also infinite.
6920: (4)                """
6921: (4)                return _mupdf.ll_fz_intersect_rect(a, b)
6922: (0)            def ll_fz_intptr_heap_insert(heap, v):
```

```
6923: (4)               r""" Low-level wrapper for `::fz_intptr_heap_insert()`."""
6924: (4)               return _mupdf.ll_fz_intptr_heap_insert(heap, v)
6925: (0)           def ll_fz_intptr_heap_sort(heap):
6926: (4)               r""" Low-level wrapper for `::fz_intptr_heap_sort()`."""
6927: (4)               return _mupdf.ll_fz_intptr_heap_sort(heap)
6928: (0)           def ll_fz_intptr_heap_uniq(heap):
6929: (4)               r""" Low-level wrapper for `::fz_intptr_heap_uniq()`."""
6930: (4)               return _mupdf.ll_fz_intptr_heap_uniq(heap)
6931: (0)           def ll_fz_invert_bitmap(bmp):
6932: (4)               r"""
6933: (4)               Low-level wrapper for `::fz_invert_bitmap()`.
6934: (4)               Invert bitmap.
6935: (4)               Never throws exceptions.
6936: (4)               """
6937: (4)               return _mupdf.ll_fz_invert_bitmap(bmp)
6938: (0)           def ll_fz_invert_matrix(matrix):
6939: (4)               r"""
6940: (4)               Low-level wrapper for `::fz_invert_matrix()`.
6941: (4)               Create an inverse matrix.
6942: (4)               inverse: Place to store inverse matrix.
6943: (4)               matrix: Matrix to invert. A degenerate matrix, where the
6944: (4)               determinant is equal to zero, can not be inverted and the
6945: (4)               original matrix is returned instead.
6946: (4)               Returns inverse.
6947: (4)               """
6948: (4)               return _mupdf.ll_fz_invert_matrix(matrix)
6949: (0)           def ll_fz_invert_pixmap(pix):
6950: (4)               r"""
6951: (4)               Low-level wrapper for `::fz_invert_pixmap()`.
6952: (4)               Invert all the pixels in a pixmap. All components (process and
6953: (4)               spots) of all pixels are inverted (except alpha, which is
6954: (4)               unchanged).
6955: (4)               """
6956: (4)               return _mupdf.ll_fz_invert_pixmap(pix)
6957: (0)           def ll_fz_invert_pixmap_alpha(pix):
6958: (4)               r"""
6959: (4)               Low-level wrapper for `::fz_invert_pixmap_alpha()`.
6960: (4)               Invert the alpha fo all the pixels in a pixmap.
6961: (4)               """
6962: (4)               return _mupdf.ll_fz_invert_pixmap_alpha(pix)
6963: (0)           def ll_fz_invert_pixmap_luminance(pix):
6964: (4)               r"""
6965: (4)               Low-level wrapper for `::fz_invert_pixmap_luminance()`.
6966: (4)               Transform the pixels in a pixmap so that luminance of each
6967: (4)               pixel is inverted, and the chrominance remains unchanged (as
6968: (4)               much as accuracy allows).
6969: (4)               All components of all pixels are inverted (except alpha, which
6970: (4)               is unchanged). Only supports Grey and RGB bitmaps.
6971: (4)               """
6972: (4)               return _mupdf.ll_fz_invert_pixmap_luminance(pix)
6973: (0)           def ll_fz_invert_pixmap_raw(pix):
6974: (4)               r"""
6975: (4)               Low-level wrapper for `::fz_invert_pixmap_raw()`.
6976: (4)               Invert all the pixels in a non-premultiplied pixmap in a
6977: (4)               very naive manner.
6978: (4)               """
6979: (4)               return _mupdf.ll_fz_invert_pixmap_raw(pix)
6980: (0)           def ll_fz_invert_pixmap_rect(image, rect):
6981: (4)               r"""
6982: (4)               Low-level wrapper for `::fz_invert_pixmap_rect()`.
6983: (4)               Invert all the pixels in a given rectangle of a (premultiplied)
6984: (4)               pixmap. All components of all pixels in the rectangle are
6985: (4)               inverted (except alpha, which is unchanged).
6986: (4)               """
6987: (4)               return _mupdf.ll_fz_invert_pixmap_rect(image, rect)
6988: (0)           def ll_fz_irect_from_rect(rect):
6989: (4)               r"""
6990: (4)               Low-level wrapper for `::fz_irect_from_rect()`.
6991: (4)               Convert a rect into the minimal bounding box
```

```
6992: (4)               that covers the rectangle.
6993: (4)               Coordinates in a bounding box are integers, so rounding of the
6994: (4)               rects coordinates takes place. The top left corner is rounded
6995: (4)               upwards and left while the bottom right corner is rounded
6996: (4)               downwards and to the right.
6997: (4)               """
6998: (4)               return _mupdf.ll_fz_irect_from_rect(rect)
6999: (0)           def ll_fz_irect_height(r):
7000: (4)               r"""
7001: (4)               Low-level wrapper for `::fz_irect_height()`.
7002: (4)               Return the height of an irect. Invalid irects return 0.
7003: (4)               """
7004: (4)               return _mupdf.ll_fz_irect_height(r)
7005: (0)           def ll_fz_irect_width(r):
7006: (4)               r"""
7007: (4)               Low-level wrapper for `::fz_irect_width()`.
7008: (4)               Return the width of an irect. Invalid irects return 0.
7009: (4)               """
7010: (4)               return _mupdf.ll_fz_irect_width(r)
7011: (0)           def ll_fz_is_cfb_archive(file):
7012: (4)               r"""
7013: (4)               Low-level wrapper for `::fz_is_cfb_archive()`.
7014: (4)               Detect if stream object is a cfb archive.
7015: (4)               Assumes that the stream object is seekable.
7016: (4)               """
7017: (4)               return _mupdf.ll_fz_is_cfb_archive(file)
7018: (0)           def ll_fz_is_directory(path):
7019: (4)               r"""
7020: (4)               Low-level wrapper for `::fz_is_directory()`.
7021: (4)               Determine if a given path is a directory.
7022: (4)               In the case of the path not existing, or having no access
7023: (4)               we will return 0.
7024: (4)               """
7025: (4)               return _mupdf.ll_fz_is_directory(path)
7026: (0)           def ll_fz_is_document_reflowable(doc):
7027: (4)               r"""
7028: (4)               Low-level wrapper for `::fz_is_document_reflowable()`.
7029: (4)               Is the document reflowable.
7030: (4)               Returns 1 to indicate reflowable documents, otherwise 0.
7031: (4)               """
7032: (4)               return _mupdf.ll_fz_is_document_reflowable(doc)
7033: (0)           def ll_fz_is_empty_irect(r):
7034: (4)               r""" Low-level wrapper for `::fz_is_empty_irect()`."""
7035: (4)               return _mupdf.ll_fz_is_empty_irect(r)
7036: (0)           def ll_fz_is_empty_rect(r):
7037: (4)               r"""
7038: (4)               Low-level wrapper for `::fz_is_empty_rect()`.
7039: (4)               Check if rectangle is empty.
7040: (4)               An empty rectangle is defined as one whose area is zero.
7041: (4)               All invalid rectangles are empty.
7042: (4)               """
7043: (4)               return _mupdf.ll_fz_is_empty_rect(r)
7044: (0)           def ll_fz_is_eof(stm):
7045: (4)               r"""
7046: (4)               Low-level wrapper for `::fz_is_eof()`.
7047: (4)               Query if the stream has reached EOF (during normal bytewise
7048: (4)               reading).
7049: (4)               See fz_is_eof_bits for the equivalent function for bitwise
7050: (4)               reading.
7051: (4)               """
7052: (4)               return _mupdf.ll_fz_is_eof(stm)
7053: (0)           def ll_fz_is_eof_bits(stm):
7054: (4)               r"""
7055: (4)               Low-level wrapper for `::fz_is_eof_bits()`.
7056: (4)               Query if the stream has reached EOF (during bitwise
7057: (4)               reading).
7058: (4)               See fz_is_eof for the equivalent function for bytewise
7059: (4)               reading.
7060: (4)               """
```

```
7061: (4)               return _mupdf.ll_fz_is_eof_bits(stm)
7062: (0)           def ll_fz_is_external_link(uri):
7063: (4)               r"""
7064: (4)               Low-level wrapper for `::fz_is_external_link()`.
7065: (4)               Query whether a link is external to a document (determined by
7066: (4)               uri containing a ':', intended to match with '://' which
7067: (4)               separates the scheme from the scheme specific parts in URIs).
7068: (4)               """
7069: (4)               return _mupdf.ll_fz_is_external_link(uri)
7070: (0)           def ll_fz_is_identity(m):
7071: (4)               r""" Low-level wrapper for `::fz_is_identity()`."""
7072: (4)               return _mupdf.ll_fz_is_identity(m)
7073: (0)           def ll_fz_is_infinite_irect(r):
7074: (4)               r"""
7075: (4)               Low-level wrapper for `::fz_is_infinite_irect()`.
7076: (4)               Check if an integer rectangle
7077: (4)               is infinite.
7078: (4)               """
7079: (4)               return _mupdf.ll_fz_is_infinite_irect(r)
7080: (0)           def ll_fz_is_infinite_rect(r):
7081: (4)               r"""
7082: (4)               Low-level wrapper for `::fz_is_infinite_rect()`.
7083: (4)               Check if rectangle is infinite.
7084: (4)               """
7085: (4)               return _mupdf.ll_fz_is_infinite_rect(r)
7086: (0)           def ll_fz_is_libarchive_archive(file):
7087: (4)               r"""
7088: (4)               Low-level wrapper for `::fz_is_libarchive_archive()`.
7089: (4)               Detect if stream object is an archive supported by libarchive.
7090: (4)               Assumes that the stream object is seekable.
7091: (4)               """
7092: (4)               return _mupdf.ll_fz_is_libarchive_archive(file)
7093: (0)           def ll_fz_is_page_range(s):
7094: (4)               r"""
7095: (5)                Low-level wrapper for `::fz_is_page_range()`.
7096: (8)                   Check and parse string into page ranges:
7097: (4)               ,?(-?+|N)(-(-?+|N))?/
7098: (4)               """
7099: (4)               return _mupdf.ll_fz_is_page_range(s)
7100: (0)           def ll_fz_is_pixmap_monochrome(pixmap):
7101: (4)               r"""
7102: (4)               Low-level wrapper for `::fz_is_pixmap_monochrome()`.
7103: (4)               Check if the pixmap is a 1-channel image containing samples with
7104: (4)               only values 0 and 255
7105: (4)               """
7106: (4)               return _mupdf.ll_fz_is_pixmap_monochrome(pixmap)
7107: (0)           def ll_fz_is_point_inside_irect(x, y, r):
7108: (4)               r"""
7109: (4)               Low-level wrapper for `::fz_is_point_inside_irect()`.
7110: (4)               Inclusion test for irects. (Rect is assumed to be open, i.e.
7111: (4)               top right corner is not included).
7112: (4)               """
7113: (4)               return _mupdf.ll_fz_is_point_inside_irect(x, y, r)
7114: (0)           def ll_fz_is_point_inside_quad(p, q):
7115: (4)               r"""
7116: (4)               Low-level wrapper for `::fz_is_point_inside_quad()`.
7117: (4)               Inclusion test for quads.
7118: (4)               """
7119: (4)               return _mupdf.ll_fz_is_point_inside_quad(p, q)
7120: (0)           def ll_fz_is_point_inside_rect(p, r):
7121: (4)               r"""
7122: (4)               Low-level wrapper for `::fz_is_point_inside_rect()`.
7123: (4)               Inclusion test for rects. (Rect is assumed to be open, i.e.
7124: (4)               top right corner is not included).
7125: (4)               """
7126: (4)               return _mupdf.ll_fz_is_point_inside_rect(p, r)
7127: (0)           def ll_fz_is_pow2(a):
7128: (4)               r""" Low-level wrapper for `::fz_is_pow2()`."""
7129: (4)               return _mupdf.ll_fz_is_pow2(a)
```

```
(0)         def ll_fz_is_quad_inside_quad(needle, haystack):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_quad_inside_quad()`.
(4)             Inclusion test for quad in quad.
(4)             This may break down if quads are not 'well formed'.
(4)             """
(4)             return _mupdf.ll_fz_is_quad_inside_quad(needle, haystack)
(0)         def ll_fz_is_quad_intersecting_quad(a, b):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_quad_intersecting_quad()`.
(4)             Intersection test for quads.
(4)             This may break down if quads are not 'well formed'.
(4)             """
(4)             return _mupdf.ll_fz_is_quad_intersecting_quad(a, b)
(0)         def ll_fz_is_rectilinear(m):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_rectilinear()`.
(4)             Check if a transformation is rectilinear.
(4)             Rectilinear means that no shearing is present and that any
(4)             rotations present are a multiple of 90 degrees. Usually this
(4)             is used to make sure that axis-aligned rectangles before the
(4)             transformation are still axis-aligned rectangles afterwards.
(4)             """
(4)             return _mupdf.ll_fz_is_rectilinear(m)
(0)         def ll_fz_is_tar_archive(file):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_tar_archive()`.
(4)             Detect if stream object is a tar archive.
(4)             Assumes that the stream object is seekable.
(4)             """
(4)             return _mupdf.ll_fz_is_tar_archive(file)
(0)         def ll_fz_is_valid_blend_colorspace(cs):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_valid_blend_colorspace()`.
(4)             Check to see that a colorspace is appropriate to be used as
(4)             a blending space (i.e. only grey, rgb or cmyk).
(4)             """
(4)             return _mupdf.ll_fz_is_valid_blend_colorspace(cs)
(0)         def ll_fz_is_valid_irect(r):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_valid_irect()`.
(4)             Check if an integer rectangle is valid.
(4)             """
(4)             return _mupdf.ll_fz_is_valid_irect(r)
(0)         def ll_fz_is_valid_rect(r):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_valid_rect()`.
(4)             Check if rectangle is valid.
(4)             """
(4)             return _mupdf.ll_fz_is_valid_rect(r)
(0)         def ll_fz_is_zip_archive(file):
(4)             r"""
(4)             Low-level wrapper for `::fz_is_zip_archive()`.
(4)             Detect if stream object is a zip archive.
(4)             Assumes that the stream object is seekable.
(4)             """
(4)             return _mupdf.ll_fz_is_zip_archive(file)
(0)         def ll_fz_iso8859_1_from_unicode(u):
(4)             r""" Low-level wrapper for `::fz_iso8859_1_from_unicode()`."""
(4)             return _mupdf.ll_fz_iso8859_1_from_unicode(u)
(0)         def ll_fz_iso8859_7_from_unicode(u):
(4)             r""" Low-level wrapper for `::fz_iso8859_7_from_unicode()`."""
(4)             return _mupdf.ll_fz_iso8859_7_from_unicode(u)
(0)         def ll_fz_jbig2_globals_data(globals):
(4)             r"""
(4)             Low-level wrapper for `::fz_jbig2_globals_data()`.
(4)             Return buffer containing jbig2 globals data stream.
(4)             """
(4)             return _mupdf.ll_fz_jbig2_globals_data(globals)
```

```
def ll_fz_keep_archive(arch):
    r"""
    Low-level wrapper for `::fz_keep_archive()`.
    Keep a reference to an archive.
    """
    return _mupdf.ll_fz_keep_archive(arch)
def ll_fz_keep_bitmap(bit):
    r"""
    Low-level wrapper for `::fz_keep_bitmap()`.
    Take an additional reference to the bitmap. The same pointer
    is returned.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_bitmap(bit)
def ll_fz_keep_buffer(buf):
    r"""
    Low-level wrapper for `::fz_keep_buffer()`.
    Take an additional reference to the buffer. The same pointer
    is returned.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_buffer(buf)
def ll_fz_keep_colorspace(colorspace):
    r"""
    Low-level wrapper for `::fz_keep_colorspace()`.
    Increment the reference count for the colorspace.
    Returns the same pointer. Never throws an exception.
    """
    return _mupdf.ll_fz_keep_colorspace(colorspace)
def ll_fz_keep_compressed_buffer(cbuf):
    r"""
    Low-level wrapper for `::fz_keep_compressed_buffer()`.
    Take a reference to an fz_compressed_buffer.
    """
    return _mupdf.ll_fz_keep_compressed_buffer(cbuf)
def ll_fz_keep_default_colorspaces(default_cs):
    r"""
    Low-level wrapper for `::fz_keep_default_colorspaces()`.
    Keep an additional reference to the default colorspaces
    structure.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_default_colorspaces(default_cs)
def ll_fz_keep_device(dev):
    r"""
    Low-level wrapper for `::fz_keep_device()`.
    Increment the reference count for a device. Returns the same
    pointer.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_device(dev)
def ll_fz_keep_display_list(list):
    r"""
    Low-level wrapper for `::fz_keep_display_list()`.
    Increment the reference count for a display list. Returns the
    same pointer.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_display_list(list)
def ll_fz_keep_document(doc):
    r"""
    Low-level wrapper for `::fz_keep_document()`.
    Increment the document reference count. The same pointer is
    returned.
    Never throws exceptions.
    """
    return _mupdf.ll_fz_keep_document(doc)
def ll_fz_keep_font(font):
    r"""
```

```
(4)             Low-level wrapper for `::fz_keep_font()`.
(4)             Add a reference to an existing fz_font.
(4)             font: The font to add a reference to.
(4)             Returns the same font.
(4)             """
(4)             return _mupdf.ll_fz_keep_font(font)
(0)         def ll_fz_keep_function(func):
(4)             r""" Low-level wrapper for `::fz_keep_function()`."""
(4)             return _mupdf.ll_fz_keep_function(func)
(0)         def ll_fz_keep_glyph(pix):
(4)             r"""
(4)             Low-level wrapper for `::fz_keep_glyph()`.
(4)             Take a reference to a glyph.
(4)             pix: The glyph to increment the reference for.
(4)             Returns pix.
(4)             """
(4)             return _mupdf.ll_fz_keep_glyph(pix)
(0)         def ll_fz_keep_halftone(half):
(4)             r"""
(4)             Low-level wrapper for `::fz_keep_halftone()`.
(4)             Take an additional reference to the halftone. The same pointer
(4)             is returned.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_keep_halftone(half)
(0)         def ll_fz_keep_image(image):
(4)             r"""
(4)             Low-level wrapper for `::fz_keep_image()`.
(4)             Increment the (normal) reference count for an image. Returns the
(4)             same pointer.
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_keep_image(image)
(0)         def ll_fz_keep_image_store_key(image):
(4)             r"""
(4)             Low-level wrapper for `::fz_keep_image_store_key()`.
(4)             Increment the store key reference for an image. Returns the same
(4)             pointer. (This is the count of references for an image held by
(4)             keys in the image store).
(4)             Never throws exceptions.
(4)             """
(4)             return _mupdf.ll_fz_keep_image_store_key(image)
(0)         def ll_fz_keep_imp(p, refs):
(4)             r"""
(5)              Low-level wrapper for `::fz_keep_imp()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_fz_keep_imp(void *p)` => `(void *, int refs)`
(4)             """
(4)             return _mupdf.ll_fz_keep_imp(p, refs)
(0)         def ll_fz_keep_imp16(p, refs):
(4)             r"""
(5)              Low-level wrapper for `::fz_keep_imp16()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_fz_keep_imp16(void *p)` => `(void *, int16_t refs)`
(4)             """
(4)             return _mupdf.ll_fz_keep_imp16(p, refs)
(0)         def ll_fz_keep_imp8(p, refs):
(4)             r""" Low-level wrapper for `::fz_keep_imp8()`."""
(4)             return _mupdf.ll_fz_keep_imp8(p, refs)
(0)         def ll_fz_keep_imp8_locked(p, refs):
(4)             r""" Low-level wrapper for `::fz_keep_imp8_locked()`."""
(4)             return _mupdf.ll_fz_keep_imp8_locked(p, refs)
(0)         def ll_fz_keep_imp_locked(p, refs):
(4)             r"""
(5)              Low-level wrapper for `::fz_keep_imp_locked()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_fz_keep_imp_locked(void *p)` => `(void *, int refs)`
(4)             """
(4)             return _mupdf.ll_fz_keep_imp_locked(p, refs)
```

```
(0)     def ll_fz_keep_jbig2_globals(globals):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_jbig2_globals()`.
(4)         Increment the reference count for a jbig2 globals record.
(4)         Never throws an exception.
(4)         """
(4)         return _mupdf.ll_fz_keep_jbig2_globals(globals)
(0)     def ll_fz_keep_key_storable(arg_1):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_key_storable()`.
(4)         Increment the (normal) reference count for a key storable
(4)         object. Returns the same pointer.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_key_storable(arg_1)
(0)     def ll_fz_keep_key_storable_key(arg_1):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_key_storable_key()`.
(4)         Increment the (key) reference count for a key storable
(4)         object. Returns the same pointer.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_key_storable_key(arg_1)
(0)     def ll_fz_keep_link(link):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_link()`.
(4)         Increment the reference count for a link. The same pointer is
(4)         returned.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_link(link)
(0)     def ll_fz_keep_outline(outline):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_outline()`.
(4)         Increment the reference count. Returns the same pointer.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_outline(outline)
(0)     def ll_fz_keep_page(page):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_page()`.
(4)         Increment the reference count for the page. Returns the same
(4)         pointer.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_page(page)
(0)     def ll_fz_keep_page_locked(page):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_page_locked()`.
(4)         Increment the reference count for the page. Returns the same
(4)         pointer. Must only be used when the alloc lock is already taken.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_page_locked(page)
(0)     def ll_fz_keep_path(path):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_path()`.
(4)         Increment the reference count. Returns the same pointer.
(4)         All paths can be kept, regardless of their packing type.
(4)         Never throws exceptions.
(4)         """
(4)         return _mupdf.ll_fz_keep_path(path)
(0)     def ll_fz_keep_pixmap(pix):
(4)         r"""
(4)         Low-level wrapper for `::fz_keep_pixmap()`.
(4)         Increment the reference count for the pixmap. The same pointer
(4)         is returned.
(4)         Never throws exceptions.
(4)         """
```

```
7406: (4)               return _mupdf.ll_fz_keep_pixmap(pix)
7407: (0)           def ll_fz_keep_separations(sep):
7408: (4)               r"""
7409: (4)               Low-level wrapper for `::fz_keep_separations()`.
7410: (4)               Increment the reference count for a separations structure.
7411: (4)               Returns the same pointer.
7412: (4)               Never throws exceptions.
7413: (4)               """
7414: (4)               return _mupdf.ll_fz_keep_separations(sep)
7415: (0)           def ll_fz_keep_shade(shade):
7416: (4)               r"""
7417: (4)               Low-level wrapper for `::fz_keep_shade()`.
7418: (4)               Increment the reference count for the shade structure. The
7419: (4)               same pointer is returned.
7420: (4)               Never throws exceptions.
7421: (4)               """
7422: (4)               return _mupdf.ll_fz_keep_shade(shade)
7423: (0)           def ll_fz_keep_storable(arg_1):
7424: (4)               r"""
7425: (4)               Low-level wrapper for `::fz_keep_storable()`.
7426: (4)               Increment the reference count for a storable object.
7427: (4)               Returns the same pointer.
7428: (4)               Never throws exceptions.
7429: (4)               """
7430: (4)               return _mupdf.ll_fz_keep_storable(arg_1)
7431: (0)           def ll_fz_keep_store_context():
7432: (4)               r"""
7433: (4)               Low-level wrapper for `::fz_keep_store_context()`.
7434: (4)               Increment the reference count for the store context. Returns
7435: (4)               the same pointer.
7436: (4)               Never throws exceptions.
7437: (4)               """
7438: (4)               return _mupdf.ll_fz_keep_store_context()
7439: (0)           def ll_fz_keep_stream(stm):
7440: (4)               r"""
7441: (4)               Low-level wrapper for `::fz_keep_stream()`.
7442: (4)               Increments the reference count for a stream. Returns the same
7443: (4)               pointer.
7444: (4)               Never throws exceptions.
7445: (4)               """
7446: (4)               return _mupdf.ll_fz_keep_stream(stm)
7447: (0)           def ll_fz_keep_string(str):
7448: (4)               r""" Low-level wrapper for `::fz_keep_string()`."""
7449: (4)               return _mupdf.ll_fz_keep_string(str)
7450: (0)           def ll_fz_keep_stroke_state(stroke):
7451: (4)               r"""
7452: (4)               Low-level wrapper for `::fz_keep_stroke_state()`.
7453: (4)               Take an additional reference to a stroke state structure.
7454: (4)               No modifications should be carried out on a stroke
7455: (4)               state to which more than one reference is held, as
7456: (4)               this can cause race conditions.
7457: (4)               """
7458: (4)               return _mupdf.ll_fz_keep_stroke_state(stroke)
7459: (0)           def ll_fz_keep_text(text):
7460: (4)               r"""
7461: (4)               Low-level wrapper for `::fz_keep_text()`.
7462: (4)               Increment the reference count for the text object. The same
7463: (4)               pointer is returned.
7464: (4)               Never throws exceptions.
7465: (4)               """
7466: (4)               return _mupdf.ll_fz_keep_text(text)
7467: (0)           def ll_fz_keep_xml(xml):
7468: (4)               r"""
7469: (4)               Low-level wrapper for `::fz_keep_xml()`.
7470: (4)               Add a reference to the XML.
7471: (4)               """
7472: (4)               return _mupdf.ll_fz_keep_xml(xml)
7473: (0)           def ll_fz_koi8u_from_unicode(u):
7474: (4)               r""" Low-level wrapper for `::fz_koi8u_from_unicode()`."""
```

```
7475: (4)                return _mupdf.ll_fz_koi8u_from_unicode(u)
7476: (0)            def ll_fz_last_page(doc):
7477: (4)                r"""
7478: (4)                Low-level wrapper for `::fz_last_page()`.
7479: (4)                Function to get the location for the last page in the document.
7480: (4)                Using this can be far more efficient in some cases than calling
7481: (4)                fz_count_pages and using the page number.
7482: (4)                """
7483: (4)                return _mupdf.ll_fz_last_page(doc)
7484: (0)            def ll_fz_layout_document(doc, w, h, em):
7485: (4)                r"""
7486: (4)                Low-level wrapper for `::fz_layout_document()`.
7487: (4)                Layout reflowable document types.
7488: (4)                w, h: Page size in points.
7489: (4)                em: Default font size in points.
7490: (4)                """
7491: (4)                return _mupdf.ll_fz_layout_document(doc, w, h, em)
7492: (0)            def ll_fz_lineto(path, x, y):
7493: (4)                r"""
7494: (4)                Low-level wrapper for `::fz_lineto()`.
7495: (4)                Append a 'lineto' command to an open path.
7496: (4)                path: The path to modify.
7497: (4)                x, y: The coordinate to line to.
7498: (4)                Throws exceptions on failure to allocate, or attempting to
7499: (4)                modify a packed path.
7500: (4)                """
7501: (4)                return _mupdf.ll_fz_lineto(path, x, y)
7502: (0)            def ll_fz_list_archive_entry(arch, idx):
7503: (4)                r"""
7504: (4)                Low-level wrapper for `::fz_list_archive_entry()`.
7505: (4)                Get listed name of entry position idx.
7506: (4)                idx: Must be a value >= 0 < return value from
7507: (4)                fz_count_archive_entries. If not in range NULL will be
7508: (4)                returned.
7509: (4)                May throw an exception if this type of archive cannot list the
7510: (4)                entries (such as a directory).
7511: (4)                """
7512: (4)                return _mupdf.ll_fz_list_archive_entry(arch, idx)
7513: (0)            def ll_fz_load_bmp_subimage(buf, len, subimage):
7514: (4)                r""" Low-level wrapper for `::fz_load_bmp_subimage()`."""
7515: (4)                return _mupdf.ll_fz_load_bmp_subimage(buf, len, subimage)
7516: (0)            def ll_fz_load_bmp_subimage_count(buf, len):
7517: (4)                r""" Low-level wrapper for `::fz_load_bmp_subimage_count()`."""
7518: (4)                return _mupdf.ll_fz_load_bmp_subimage_count(buf, len)
7519: (0)            def ll_fz_load_chapter_page(doc, chapter, page):
7520: (4)                r"""
7521: (4)                Low-level wrapper for `::fz_load_chapter_page()`.
7522: (4)                Load a page.
7523: (4)                After fz_load_page is it possible to retrieve the size of the
7524: (4)                page using fz_bound_page, or to render the page using
7525: (4)                fz_run_page_*. Free the page by calling fz_drop_page.
7526: (4)                chapter: chapter number, 0 is the first chapter of the document.
7527: (4)                number: page number, 0 is the first page of the chapter.
7528: (4)                """
7529: (4)                return _mupdf.ll_fz_load_chapter_page(doc, chapter, page)
7530: (0)            def ll_fz_load_fallback_font(script, language, serif, bold, italic):
7531: (4)                r"""
7532: (4)                Low-level wrapper for `::fz_load_fallback_font()`.
7533: (4)                Try to load a fallback font for the
7534: (4)                given combination of font attributes. Whether a font is
7535: (4)                present or not will depend on the configuration in which
7536: (4)                MuPDF is built.
7537: (4)                script: The script desired (e.g. UCDN_SCRIPT_KATAKANA).
7538: (4)                language: The language desired (e.g. FZ_LANG_ja).
7539: (4)                serif: 1 if serif desired, 0 otherwise.
7540: (4)                bold: 1 if bold desired, 0 otherwise.
7541: (4)                italic: 1 if italic desired, 0 otherwise.
7542: (4)                Returns a new font handle, or NULL if not available.
7543: (4)                """
```

```
7544: (4)                return _mupdf.ll_fz_load_fallback_font(script, language, serif, bold,
italic)
7545: (0)            def ll_fz_load_jbig2_globals(buf):
7546: (4)                r"""
7547: (4)                Low-level wrapper for `::fz_load_jbig2_globals()`.
7548: (4)                Create a jbig2 globals record from a buffer.
7549: (4)                Immutable once created.
7550: (4)                """
7551: (4)                return _mupdf.ll_fz_load_jbig2_globals(buf)
7552: (0)            def ll_fz_load_jbig2_subimage(buf, len, subimage):
7553: (4)                r""" Low-level wrapper for `::fz_load_jbig2_subimage()`."""
7554: (4)                return _mupdf.ll_fz_load_jbig2_subimage(buf, len, subimage)
7555: (0)            def ll_fz_load_jbig2_subimage_count(buf, len):
7556: (4)                r""" Low-level wrapper for `::fz_load_jbig2_subimage_count()`."""
7557: (4)                return _mupdf.ll_fz_load_jbig2_subimage_count(buf, len)
7558: (0)            def ll_fz_load_jpx(data, size, cs):
7559: (4)                r"""
7560: (4)                Low-level wrapper for `::fz_load_jpx()`.
7561: (4)                Exposed for PDF.
7562: (4)                """
7563: (4)                return _mupdf.ll_fz_load_jpx(data, size, cs)
7564: (0)            def ll_fz_load_links(page):
7565: (4)                r"""
7566: (4)                Low-level wrapper for `::fz_load_links()`.
7567: (4)                Load the list of links for a page.
7568: (4)                Returns a linked list of all the links on the page, each with
7569: (4)                its clickable region and link destination. Each link is
7570: (4)                reference counted so drop and free the list of links by
7571: (4)                calling fz_drop_link on the pointer return from fz_load_links.
7572: (4)                page: Page obtained from fz_load_page.
7573: (4)                """
7574: (4)                return _mupdf.ll_fz_load_links(page)
7575: (0)            def ll_fz_load_outline(doc):
7576: (4)                r"""
7577: (4)                Low-level wrapper for `::fz_load_outline()`.
7578: (4)                Load the hierarchical document outline.
7579: (4)                Should be freed by fz_drop_outline.
7580: (4)                """
7581: (4)                return _mupdf.ll_fz_load_outline(doc)
7582: (0)            def ll_fz_load_outline_from_iterator(iter):
7583: (4)                r"""
7584: (4)                Low-level wrapper for `::fz_load_outline_from_iterator()`.
7585: (4)                Routine to implement the old Structure based API from an iterator.
7586: (4)                """
7587: (4)                return _mupdf.ll_fz_load_outline_from_iterator(iter)
7588: (0)            def ll_fz_load_page(doc, number):
7589: (4)                r"""
7590: (4)                Low-level wrapper for `::fz_load_page()`.
7591: (4)                Load a given page number from a document. This may be much less
7592: (4)                efficient than loading by location (chapter+page) for some
7593: (4)                document types.
7594: (4)                """
7595: (4)                return _mupdf.ll_fz_load_page(doc, number)
7596: (0)            def ll_fz_load_pnm_subimage(buf, len, subimage):
7597: (4)                r""" Low-level wrapper for `::fz_load_pnm_subimage()`."""
7598: (4)                return _mupdf.ll_fz_load_pnm_subimage(buf, len, subimage)
7599: (0)            def ll_fz_load_pnm_subimage_count(buf, len):
7600: (4)                r""" Low-level wrapper for `::fz_load_pnm_subimage_count()`."""
7601: (4)                return _mupdf.ll_fz_load_pnm_subimage_count(buf, len)
7602: (0)            def ll_fz_load_system_cjk_font(name, ordering, serif):
7603: (4)                r"""
7604: (4)                Low-level wrapper for `::fz_load_system_cjk_font()`.
7605: (4)                Attempt to load a given font from
7606: (4)                the system.
7607: (4)                name: The name of the desired font.
7608: (4)                ordering: The ordering to load the font from (e.g. FZ_ADOBE_KOREA)
7609: (4)                serif: 1 if serif desired, 0 otherwise.
7610: (4)                Returns a new font handle, or NULL if no matching font was found
7611: (4)                (or on error).
```

```
7612: (4)                """
7613: (4)                return _mupdf.ll_fz_load_system_cjk_font(name, ordering, serif)
7614: (0)            def ll_fz_load_system_font(name, bold, italic, needs_exact_metrics):
7615: (4)                r"""
7616: (4)                Low-level wrapper for `::fz_load_system_font()`.
7617: (4)                Attempt to load a given font from the system.
7618: (4)                name: The name of the desired font.
7619: (4)                bold: 1 if bold desired, 0 otherwise.
7620: (4)                italic: 1 if italic desired, 0 otherwise.
7621: (4)                needs_exact_metrics: 1 if an exact metrical match is required,
7622: (4)                0 otherwise.
7623: (4)                Returns a new font handle, or NULL if no matching font was found
7624: (4)                (or on error).
7625: (4)                """
7626: (4)                return _mupdf.ll_fz_load_system_font(name, bold, italic,
needs_exact_metrics)
7627: (0)            def ll_fz_load_tiff_subimage(buf, len, subimage):
7628: (4)                r""" Low-level wrapper for `::fz_load_tiff_subimage()`."""
7629: (4)                return _mupdf.ll_fz_load_tiff_subimage(buf, len, subimage)
7630: (0)            def ll_fz_load_tiff_subimage_count(buf, len):
7631: (4)                r"""
7632: (4)                Low-level wrapper for `::fz_load_tiff_subimage_count()`.
7633: (4)                Exposed for CBZ.
7634: (4)                """
7635: (4)                return _mupdf.ll_fz_load_tiff_subimage_count(buf, len)
7636: (0)            def ll_fz_location_from_page_number(doc, number):
7637: (4)                r"""
7638: (4)                Low-level wrapper for `::fz_location_from_page_number()`.
7639: (4)                Converts from page number to chapter+page. This may cause many
7640: (4)                chapters to be laid out in order to calculate the number of
7641: (4)                pages within those chapters.
7642: (4)                """
7643: (4)                return _mupdf.ll_fz_location_from_page_number(doc, number)
7644: (0)            def ll_fz_lock(lock):
7645: (4)                r"""
7646: (4)                Low-level wrapper for `::fz_lock()`.
7647: (4)                Lock one of the user supplied mutexes.
7648: (4)                """
7649: (4)                return _mupdf.ll_fz_lock(lock)
7650: (0)            def ll_fz_log_error(str):
7651: (4)                r"""
7652: (4)                Low-level wrapper for `::fz_log_error()`.
7653: (4)                Log a (preformatted) string to the registered
7654: (4)                error stream (stderr by default).
7655: (4)                """
7656: (4)                return _mupdf.ll_fz_log_error(str)
7657: (0)            def ll_fz_lookup_base14_font(name, len):
7658: (4)                r"""
7659: (5)                 Low-level wrapper for `::fz_lookup_base14_font()`.
7660: (4)                This function has out-params. Python/C# wrappers look like:
7661: (8)                    `ll_fz_lookup_base14_font(const char *name)` => `(const unsigned char
*, int len)`
7662: (8)                    Search the builtin base14 fonts for a match.
7663: (8)                    Whether a given font is present or not will depend on the
7664: (8)                    configuration in which MuPDF is built.
7665: (8)                    name: The name of the font desired.
7666: (8)                    len: Pointer to a place to receive the length of the discovered
7667: (8)                    font buffer.
7668: (8)                    Returns a pointer to the font file data, or NULL if not present.
7669: (4)                """
7670: (4)                return _mupdf.ll_fz_lookup_base14_font(name, len)
7671: (0)            def ll_fz_lookup_blendmode(name):
7672: (4)                r"""
7673: (4)                Low-level wrapper for `::fz_lookup_blendmode()`.
7674: (4)                Map from (case sensitive) blend mode string to enumeration.
7675: (4)                """
7676: (4)                return _mupdf.ll_fz_lookup_blendmode(name)
7677: (0)            def ll_fz_lookup_bookmark(doc, mark):
7678: (4)                r"""
```

```
7679: (4)               Low-level wrapper for `::fz_lookup_bookmark()`.
7680: (4)               Find a bookmark and return its page number.
7681: (4)               """
7682: (4)               return _mupdf.ll_fz_lookup_bookmark(doc, mark)
7683: (0)           def ll_fz_lookup_builtin_font(name, bold, italic, len):
7684: (4)               r"""
7685: (5)                Low-level wrapper for `::fz_lookup_builtin_font()`.
7686: (4)               This function has out-params. Python/C# wrappers look like:
7687: (8)                   `ll_fz_lookup_builtin_font(const char *name, int bold, int italic)` =>
`(const unsigned char *, int len)`
7688: (8)                   Search the builtin fonts for a match.
7689: (8)                   Whether a given font is present or not will depend on the
7690: (8)                   configuration in which MuPDF is built.
7691: (8)                   name: The name of the font desired.
7692: (8)                   bold: 1 if bold desired, 0 otherwise.
7693: (8)                   italic: 1 if italic desired, 0 otherwise.
7694: (8)                   len: Pointer to a place to receive the length of the discovered
7695: (8)                   font buffer.
7696: (8)                   Returns a pointer to the font file data, or NULL if not present.
7697: (4)               """
7698: (4)               return _mupdf.ll_fz_lookup_builtin_font(name, bold, italic, len)
7699: (0)           def ll_fz_lookup_cjk_font(ordering, len, index):
7700: (4)               r"""
7701: (5)                Low-level wrapper for `::fz_lookup_cjk_font()`.
7702: (4)               This function has out-params. Python/C# wrappers look like:
7703: (8)                   `ll_fz_lookup_cjk_font(int ordering)` => `(const unsigned char *, int
len, int index)`
7704: (8)                   Search the builtin cjk fonts for a match.
7705: (8)                   Whether a font is present or not will depend on the
7706: (8)                   configuration in which MuPDF is built.
7707: (8)                   ordering: The desired ordering of the font (e.g. FZ_ADOBE_KOREA).
7708: (8)                   len: Pointer to a place to receive the length of the discovered
7709: (8)                   font buffer.
7710: (8)                   Returns a pointer to the font file data, or NULL if not present.
7711: (4)               """
7712: (4)               return _mupdf.ll_fz_lookup_cjk_font(ordering, len, index)
7713: (0)           def ll_fz_lookup_cjk_font_by_language(lang, len, subfont):
7714: (4)               r"""
7715: (5)                Low-level wrapper for `::fz_lookup_cjk_font_by_language()`.
7716: (4)               This function has out-params. Python/C# wrappers look like:
7717: (8)                   `ll_fz_lookup_cjk_font_by_language(const char *lang)` => `(const
unsigned char *, int len, int subfont)`
7718: (8)                   Search the builtin cjk fonts for a match for a given language.
7719: (8)                   Whether a font is present or not will depend on the
7720: (8)                   configuration in which MuPDF is built.
7721: (8)                   lang: Pointer to a (case sensitive) language string (e.g.
7722: (8)                   "ja", "ko", "zh-Hant" etc).
7723: (8)                   len: Pointer to a place to receive the length of the discovered
7724: (8)                   font buffer.
7725: (8)                   subfont: Pointer to a place to store the subfont index of the
7726: (8)                   discovered font.
7727: (8)                   Returns a pointer to the font file data, or NULL if not present.
7728: (4)               """
7729: (4)               return _mupdf.ll_fz_lookup_cjk_font_by_language(lang, len, subfont)
7730: (0)           def ll_fz_lookup_cjk_ordering_by_language(name):
7731: (4)               r"""
7732: (4)               Low-level wrapper for `::fz_lookup_cjk_ordering_by_language()`.
7733: (4)               Return the matching FZ_ADOBE_* ordering
7734: (4)               for the given language tag, such as "zh-Hant", "zh-Hans", "ja", or "ko".
7735: (4)               """
7736: (4)               return _mupdf.ll_fz_lookup_cjk_ordering_by_language(name)
7737: (0)           def ll_fz_lookup_image_type(type):
7738: (4)               r"""
7739: (4)               Low-level wrapper for `::fz_lookup_image_type()`.
7740: (4)               Map from (case sensitive) image type string to FZ_IMAGE_*
7741: (4)               type value.
7742: (4)               """
7743: (4)               return _mupdf.ll_fz_lookup_image_type(type)
7744: (0)           def ll_fz_lookup_metadata(doc, key, buf, size):
```

```
7745: (4)               r"""
7746: (4)               Low-level wrapper for `::fz_lookup_metadata()`.
7747: (4)               Retrieve document meta data strings.
7748: (4)               doc: The document to query.
7749: (4)               key: Which meta data key to retrieve...
7750: (4)               Basic information:
7751: (8)                   'format'    -- Document format and version.
7752: (8)                   'encryption'    -- Description of the encryption used.
7753: (4)               From the document information dictionary:
7754: (8)                   'info:Title'
7755: (8)                   'info:Author'
7756: (8)                   'info:Subject'
7757: (8)                   'info:Keywords'
7758: (8)                   'info:Creator'
7759: (8)                   'info:Producer'
7760: (8)                   'info:CreationDate'
7761: (8)                   'info:ModDate'
7762: (4)               buf: The buffer to hold the results (a nul-terminated UTF-8
7763: (4)               string).
7764: (4)               size: Size of 'buf'.
7765: (4)               Returns the number of bytes need to store the string plus terminator
7766: (4)               (will be larger than 'size' if the output was truncated), or -1 if the
7767: (4)               key is not recognized or found.
7768: (4)               """
7769: (4)               return _mupdf.ll_fz_lookup_metadata(doc, key, buf, size)
7770: (0)           def ll_fz_lookup_metadata2(doc, key):
7771: (4)               r"""
7772: (5)                Low-level wrapper for `::fz_lookup_metadata2()`.
7773: (4)               C++ alternative to `fz_lookup_metadata()` that returns a `std::string`
7774: (4)               or calls `fz_throw()` if not found.
7775: (4)               """
7776: (4)               return _mupdf.ll_fz_lookup_metadata2(doc, key)
7777: (0)           def ll_fz_lookup_noto_boxes_font(len):
7778: (4)               r"""
7779: (5)                Low-level wrapper for `::fz_lookup_noto_boxes_font()`.
7780: (4)               This function has out-params. Python/C# wrappers look like:
7781: (8)                   `ll_fz_lookup_noto_boxes_font()` => `(const unsigned char *, int len)`
7782: (4)               """
7783: (4)               return _mupdf.ll_fz_lookup_noto_boxes_font(len)
7784: (0)           def ll_fz_lookup_noto_emoji_font(len):
7785: (4)               r"""
7786: (5)                Low-level wrapper for `::fz_lookup_noto_emoji_font()`.
7787: (4)               This function has out-params. Python/C# wrappers look like:
7788: (8)                   `ll_fz_lookup_noto_emoji_font()` => `(const unsigned char *, int len)`
7789: (4)               """
7790: (4)               return _mupdf.ll_fz_lookup_noto_emoji_font(len)
7791: (0)           def ll_fz_lookup_noto_font(script, lang, len, subfont):
7792: (4)               r"""
7793: (5)                Low-level wrapper for `::fz_lookup_noto_font()`.
7794: (4)               This function has out-params. Python/C# wrappers look like:
7795: (8)                   `ll_fz_lookup_noto_font(int script, int lang)` => `(const unsigned
char *, int len, int subfont)`
7796: (8)                   Search the builtin noto fonts for a match.
7797: (8)                   Whether a font is present or not will depend on the
7798: (8)                   configuration in which MuPDF is built.
7799: (8)                   script: The script desired (e.g. UCDN_SCRIPT_KATAKANA).
7800: (8)                   lang: The language desired (e.g. FZ_LANG_ja).
7801: (8)                   len: Pointer to a place to receive the length of the discovered
7802: (8)                   font buffer.
7803: (8)                   Returns a pointer to the font file data, or NULL if not present.
7804: (4)               """
7805: (4)               return _mupdf.ll_fz_lookup_noto_font(script, lang, len, subfont)
7806: (0)           def ll_fz_lookup_noto_math_font(len):
7807: (4)               r"""
7808: (5)                Low-level wrapper for `::fz_lookup_noto_math_font()`.
7809: (4)               This function has out-params. Python/C# wrappers look like:
7810: (8)                   `ll_fz_lookup_noto_math_font()` => `(const unsigned char *, int len)`
7811: (8)                   Search the builtin noto fonts specific symbol fonts.
7812: (8)                   Whether a font is present or not will depend on the
```

```
7813: (8)                   configuration in which MuPDF is built.
7814: (4)               """
7815: (4)               return _mupdf.ll_fz_lookup_noto_math_font(len)
7816: (0)           def ll_fz_lookup_noto_music_font(len):
7817: (4)               r"""
7818: (5)                Low-level wrapper for `::fz_lookup_noto_music_font()`.
7819: (4)               This function has out-params. Python/C# wrappers look like:
7820: (8)                   `ll_fz_lookup_noto_music_font()` => `(const unsigned char *, int len)`
7821: (4)               """
7822: (4)               return _mupdf.ll_fz_lookup_noto_music_font(len)
7823: (0)           def ll_fz_lookup_noto_symbol1_font(len):
7824: (4)               r"""
7825: (5)                Low-level wrapper for `::fz_lookup_noto_symbol1_font()`.
7826: (4)               This function has out-params. Python/C# wrappers look like:
7827: (8)                   `ll_fz_lookup_noto_symbol1_font()` => `(const unsigned char *, int
len)`
7828: (4)               """
7829: (4)               return _mupdf.ll_fz_lookup_noto_symbol1_font(len)
7830: (0)           def ll_fz_lookup_noto_symbol2_font(len):
7831: (4)               r"""
7832: (5)                Low-level wrapper for `::fz_lookup_noto_symbol2_font()`.
7833: (4)               This function has out-params. Python/C# wrappers look like:
7834: (8)                   `ll_fz_lookup_noto_symbol2_font()` => `(const unsigned char *, int
len)`
7835: (4)               """
7836: (4)               return _mupdf.ll_fz_lookup_noto_symbol2_font(len)
7837: (0)           def ll_fz_lookup_rendering_intent(name):
7838: (4)               r"""
7839: (4)               Low-level wrapper for `::fz_lookup_rendering_intent()`.
7840: (4)               Map from (case sensitive) rendering intent string to enumeration
7841: (4)               value.
7842: (4)               """
7843: (4)               return _mupdf.ll_fz_lookup_rendering_intent(name)
7844: (0)           def ll_fz_make_bookmark(doc, loc):
7845: (4)               r"""
7846: (4)               Low-level wrapper for `::fz_make_bookmark()`.
7847: (4)               Create a bookmark for the given page, which can be used to find
7848: (4)               the same location after the document has been laid out with
7849: (4)               different parameters.
7850: (4)               """
7851: (4)               return _mupdf.ll_fz_make_bookmark(doc, loc)
7852: (0)           def ll_fz_make_irect(x0, y0, x1, y1):
7853: (4)               r""" Low-level wrapper for `::fz_make_irect()`."""
7854: (4)               return _mupdf.ll_fz_make_irect(x0, y0, x1, y1)
7855: (0)           def ll_fz_make_link_dest_none():
7856: (4)               r""" Low-level wrapper for `::fz_make_link_dest_none()`."""
7857: (4)               return _mupdf.ll_fz_make_link_dest_none()
7858: (0)           def ll_fz_make_link_dest_xyz(chapter, page, x, y, z):
7859: (4)               r""" Low-level wrapper for `::fz_make_link_dest_xyz()`."""
7860: (4)               return _mupdf.ll_fz_make_link_dest_xyz(chapter, page, x, y, z)
7861: (0)           def ll_fz_make_location(chapter, page):
7862: (4)               r"""
7863: (4)               Low-level wrapper for `::fz_make_location()`.
7864: (4)               Simple constructor for fz_locations.
7865: (4)               """
7866: (4)               return _mupdf.ll_fz_make_location(chapter, page)
7867: (0)           def ll_fz_make_matrix(a, b, c, d, e, f):
7868: (4)               r""" Low-level wrapper for `::fz_make_matrix()`."""
7869: (4)               return _mupdf.ll_fz_make_matrix(a, b, c, d, e, f)
7870: (0)           def ll_fz_make_point(x, y):
7871: (4)               r""" Low-level wrapper for `::fz_make_point()`."""
7872: (4)               return _mupdf.ll_fz_make_point(x, y)
7873: (0)           def ll_fz_make_quad(ul_x, ul_y, ur_x, ur_y, ll_x, ll_y, lr_x, lr_y):
7874: (4)               r"""
7875: (4)               Low-level wrapper for `::fz_make_quad()`.
7876: (4)               Inline convenience construction function.
7877: (4)               """
7878: (4)               return _mupdf.ll_fz_make_quad(ul_x, ul_y, ur_x, ur_y, ll_x, ll_y, lr_x,
lr_y)
```

```
7879: (0)           def ll_fz_make_rect(x0, y0, x1, y1):
7880: (4)               r""" Low-level wrapper for `::fz_make_rect()`."""
7881: (4)               return _mupdf.ll_fz_make_rect(x0, y0, x1, y1)
7882: (0)           def ll_fz_malloc(size):
7883: (4)               r"""
7884: (4)               Low-level wrapper for `::fz_malloc()`.
7885: (4)               Allocate uninitialized memory of a given size.
7886: (4)               Does NOT clear the memory!
7887: (4)               May return NULL for size = 0.
7888: (4)               Throws exception in the event of failure to allocate.
7889: (4)               """
7890: (4)               return _mupdf.ll_fz_malloc(size)
7891: (0)           def ll_fz_malloc_no_throw(size):
7892: (4)               r"""
7893: (4)               Low-level wrapper for `::fz_malloc_no_throw()`.
7894: (4)               fz_malloc equivalent that returns NULL rather than throwing
7895: (4)               exceptions.
7896: (4)               """
7897: (4)               return _mupdf.ll_fz_malloc_no_throw(size)
7898: (0)           def ll_fz_matrix_expansion(m):
7899: (4)               r"""
7900: (4)               Low-level wrapper for `::fz_matrix_expansion()`.
7901: (4)               Calculate average scaling factor of matrix.
7902: (4)               """
7903: (4)               return _mupdf.ll_fz_matrix_expansion(m)
7904: (0)           def ll_fz_matrix_max_expansion(m):
7905: (4)               r"""
7906: (4)               Low-level wrapper for `::fz_matrix_max_expansion()`.
7907: (4)               Find the largest expansion performed by this matrix.
7908: (4)               (i.e. max(abs(m.a),abs(m.b),abs(m.c),abs(m.d))
7909: (4)               """
7910: (4)               return _mupdf.ll_fz_matrix_max_expansion(m)
7911: (0)           def ll_fz_max(a, b):
7912: (4)               r""" Low-level wrapper for `::fz_max()`."""
7913: (4)               return _mupdf.ll_fz_max(a, b)
7914: (0)           def ll_fz_maxi(a, b):
7915: (4)               r""" Low-level wrapper for `::fz_maxi()`."""
7916: (4)               return _mupdf.ll_fz_maxi(a, b)
7917: (0)           def ll_fz_maxi64(a, b):
7918: (4)               r""" Low-level wrapper for `::fz_maxi64()`."""
7919: (4)               return _mupdf.ll_fz_maxi64(a, b)
7920: (0)           def ll_fz_maxz(a, b):
7921: (4)               r""" Low-level wrapper for `::fz_maxz()`."""
7922: (4)               return _mupdf.ll_fz_maxz(a, b)
7923: (0)           def ll_fz_md5_buffer(buffer, digest):
7924: (4)               r"""
7925: (4)               Low-level wrapper for `::fz_md5_buffer()`.
7926: (4)               Create an MD5 digest from buffer contents.
7927: (4)               Never throws exceptions.
7928: (4)               """
7929: (4)               return _mupdf.ll_fz_md5_buffer(buffer, digest)
7930: (0)           def ll_fz_md5_final(state, digest):
7931: (4)               r"""
7932: (4)               Low-level wrapper for `::fz_md5_final()`.
7933: (4)               MD5 finalization. Ends an MD5 message-digest operation, writing
7934: (4)               the message digest and zeroizing the context.
7935: (4)               Never throws an exception.
7936: (4)               """
7937: (4)               return _mupdf.ll_fz_md5_final(state, digest)
7938: (0)           def ll_fz_md5_final2(md5):
7939: (4)               r"""
7940: (5)                Low-level wrapper for `::fz_md5_final2()`.
7941: (4)               C++ alternative to fz_md5_final() that returns the digest by value.
7942: (4)               """
7943: (4)               return _mupdf.ll_fz_md5_final2(md5)
7944: (0)           def ll_fz_md5_init(state):
7945: (4)               r"""
7946: (4)               Low-level wrapper for `::fz_md5_init()`.
7947: (4)               MD5 initialization. Begins an MD5 operation, writing a new
```

```
7948: (4)                context.
7949: (4)                Never throws an exception.
7950: (4)                """
7951: (4)                return _mupdf.ll_fz_md5_init(state)
7952: (0)            def ll_fz_md5_pixmap(pixmap, digest):
7953: (4)                r""" Low-level wrapper for `::fz_md5_pixmap()`."""
7954: (4)                return _mupdf.ll_fz_md5_pixmap(pixmap, digest)
7955: (0)            def ll_fz_md5_pixmap2(pixmap):
7956: (4)                r"""
7957: (5)                 Low-level wrapper for `::fz_md5_pixmap2()`.
7958: (4)                C++ alternative to `fz_md5_pixmap()` that returns the digest by value.
7959: (4)                """
7960: (4)                return _mupdf.ll_fz_md5_pixmap2(pixmap)
7961: (0)            def ll_fz_md5_update(state, input, inlen):
7962: (4)                r"""
7963: (4)                Low-level wrapper for `::fz_md5_update()`.
7964: (4)                MD5 block update operation. Continues an MD5 message-digest
7965: (4)                operation, processing another message block, and updating the
7966: (4)                context.
7967: (4)                Never throws an exception.
7968: (4)                """
7969: (4)                return _mupdf.ll_fz_md5_update(state, input, inlen)
7970: (0)            def ll_fz_md5_update_int64(state, i):
7971: (4)                r"""
7972: (4)                Low-level wrapper for `::fz_md5_update_int64()`.
7973: (4)                MD5 block update operation. Continues an MD5 message-digest
7974: (4)                operation, processing an int64, and updating the context.
7975: (4)                Never throws an exception.
7976: (4)                """
7977: (4)                return _mupdf.ll_fz_md5_update_int64(state, i)
7978: (0)            def ll_fz_measure_string(user_font, trm, s, wmode, bidi_level, markup_dir,
language):
7979: (4)                r"""
7980: (4)                Low-level wrapper for `::fz_measure_string()`.
7981: (4)                Measure the advance width of a UTF8 string should it be added to a text
object.
7982: (4)                This uses the same layout algorithms as fz_show_string, and can be used
7983: (4)                to calculate text alignment adjustments.
7984: (4)                """
7985: (4)                return _mupdf.ll_fz_measure_string(user_font, trm, s, wmode, bidi_level,
markup_dir, language)
7986: (0)            def ll_fz_memmem(haystack, haystacklen, needle, needlelen):
7987: (4)                r"""
7988: (4)                Low-level wrapper for `::fz_memmem()`.
7989: (4)                Find the start of the first occurrence of the substring needle in
haystack.
7990: (4)                """
7991: (4)                return _mupdf.ll_fz_memmem(haystack, haystacklen, needle, needlelen)
7992: (0)            def ll_fz_memrnd(block, len):
7993: (4)                r"""
7994: (4)                Low-level wrapper for `::fz_memrnd()`.
7995: (4)                Fill block with len bytes of pseudo-randomness.
7996: (4)                """
7997: (4)                return _mupdf.ll_fz_memrnd(block, len)
7998: (0)            def ll_fz_min(a, b):
7999: (4)                r""" Low-level wrapper for `::fz_min()`."""
8000: (4)                return _mupdf.ll_fz_min(a, b)
8001: (0)            def ll_fz_mini(a, b):
8002: (4)                r""" Low-level wrapper for `::fz_mini()`."""
8003: (4)                return _mupdf.ll_fz_mini(a, b)
8004: (0)            def ll_fz_mini64(a, b):
8005: (4)                r""" Low-level wrapper for `::fz_mini64()`."""
8006: (4)                return _mupdf.ll_fz_mini64(a, b)
8007: (0)            def ll_fz_minz(a, b):
8008: (4)                r""" Low-level wrapper for `::fz_minz()`."""
8009: (4)                return _mupdf.ll_fz_minz(a, b)
8010: (0)            def ll_fz_mkdir(path):
8011: (4)                r""" Low-level wrapper for `::fz_mkdir()`."""
8012: (4)                return _mupdf.ll_fz_mkdir(path)
```

```
8013: (0)           def ll_fz_morph_error(fromcode, tocode):
8014: (4)               r"""
8015: (4)               Low-level wrapper for `::fz_morph_error()`.
8016: (4)               Called within a catch block this modifies the current
8017: (4)               exception's code. If it's of type 'fromcode' it is
8018: (4)               modified to 'tocode'. Typically used for 'downgrading'
8019: (4)               exception severity.
8020: (4)               """
8021: (4)               return _mupdf.ll_fz_morph_error(fromcode, tocode)
8022: (0)           def ll_fz_mount_multi_archive(arch_, sub, path):
8023: (4)               r"""
8024: (4)               Low-level wrapper for `::fz_mount_multi_archive()`.
8025: (4)               Add an archive to the set of archives handled by a multi
8026: (4)               archive.
8027: (4)               If path is NULL, then the archive contents will appear at the
8028: (4)               top level, otherwise, the archives contents will appear prefixed
8029: (4)               by path.
8030: (4)               """
8031: (4)               return _mupdf.ll_fz_mount_multi_archive(arch_, sub, path)
8032: (0)           def ll_fz_moveto(path, x, y):
8033: (4)               r"""
8034: (4)               Low-level wrapper for `::fz_moveto()`.
8035: (4)               Append a 'moveto' command to a path.
8036: (4)               This 'opens' a path.
8037: (4)               path: The path to modify.
8038: (4)               x, y: The coordinate to move to.
8039: (4)               Throws exceptions on failure to allocate, or attempting to
8040: (4)               modify a packed path.
8041: (4)               """
8042: (4)               return _mupdf.ll_fz_moveto(path, x, y)
8043: (0)           def ll_fz_mul255(a, b):
8044: (4)               r"""
8045: (4)               Low-level wrapper for `::fz_mul255()`.
8046: (4)               Multiply scaled two integers in the 0..255 range
8047: (4)               """
8048: (4)               return _mupdf.ll_fz_mul255(a, b)
8049: (0)           def ll_fz_needs_password(doc):
8050: (4)               r"""
8051: (4)               Low-level wrapper for `::fz_needs_password()`.
8052: (4)               Check if a document is encrypted with a
8053: (4)               non-blank password.
8054: (4)               """
8055: (4)               return _mupdf.ll_fz_needs_password(doc)
8056: (0)           def ll_fz_new_arc4_output(chain, key, keylen):
8057: (4)               r""" Low-level wrapper for `::fz_new_arc4_output()`."""
8058: (4)               return _mupdf.ll_fz_new_arc4_output(chain, key, keylen)
8059: (0)           def ll_fz_new_archive_of_size(file, size):
8060: (4)               r""" Low-level wrapper for `::fz_new_archive_of_size()`."""
8061: (4)               return _mupdf.ll_fz_new_archive_of_size(file, size)
8062: (0)           def ll_fz_new_ascii85_output(chain):
8063: (4)               r""" Low-level wrapper for `::fz_new_ascii85_output()`."""
8064: (4)               return _mupdf.ll_fz_new_ascii85_output(chain)
8065: (0)           def ll_fz_new_asciihex_output(chain):
8066: (4)               r"""
8067: (4)               Low-level wrapper for `::fz_new_asciihex_output()`.
8068: (4)               Compression and other filtering outputs.
8069: (4)               These outputs write encoded data to another output. Create a
8070: (4)               filter output with the destination, write to the filter, then
8071: (4)               close and drop it when you're done. These can also be chained
8072: (4)               together, for example to write ASCII Hex encoded, Deflate
8073: (4)               compressed, and RC4 encrypted data to a buffer output.
8074: (4)               Output streams don't use reference counting, so make sure to
8075: (4)               close all of the filters in the reverse order of creation so
8076: (4)               that data is flushed properly.
8077: (4)               Accordingly, ownership of 'chain' is never passed into the
8078: (4)               following functions, but remains with the caller, whose
8079: (4)               responsibility it is to ensure they exist at least until
8080: (4)               the returned fz_output is dropped.
8081: (4)               """
```

```
(4)             return _mupdf.ll_fz_new_asciihex_output(chain)
(0)         def ll_fz_new_band_writer_of_size(size, out):
(4)             r""" Low-level wrapper for `::fz_new_band_writer_of_size()`."""
(4)             return _mupdf.ll_fz_new_band_writer_of_size(size, out)
(0)         def ll_fz_new_base14_font(name):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_base14_font()`.
(4)             Create a new font from one of the built-in fonts.
(4)             """
(4)             return _mupdf.ll_fz_new_base14_font(name)
(0)         def ll_fz_new_bbox_device(rectp):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_bbox_device()`.
(4)             Create a device to compute the bounding
(4)             box of all marks on a page.
(4)             The returned bounding box will be the union of all bounding
(4)             boxes of all objects on a page.
(4)             """
(4)             return _mupdf.ll_fz_new_bbox_device(rectp)
(0)         def ll_fz_new_bitmap(w, h, n, xres, yres):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_bitmap()`.
(4)             Create a new bitmap.
(4)             w, h: Width and Height for the bitmap
(4)             n: Number of color components (assumed to be a divisor of 8)
(4)             xres, yres: X and Y resolutions (in pixels per inch).
(4)             Returns pointer to created bitmap structure. The bitmap
(4)             data is uninitialised.
(4)             """
(4)             return _mupdf.ll_fz_new_bitmap(w, h, n, xres, yres)
(0)         def ll_fz_new_bitmap_from_pixmap(pix, ht):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_bitmap_from_pixmap()`.
(4)             Make a bitmap from a pixmap and a halftone.
(4)             pix: The pixmap to generate from. Currently must be a single
(4)             color component with no alpha.
(4)             ht: The halftone to use. NULL implies the default halftone.
(4)             Returns the resultant bitmap. Throws exceptions in the case of
(4)             failure to allocate.
(4)             """
(4)             return _mupdf.ll_fz_new_bitmap_from_pixmap(pix, ht)
(0)         def ll_fz_new_bitmap_from_pixmap_band(pix, ht, band_start):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_bitmap_from_pixmap_band()`.
(4)             Make a bitmap from a pixmap and a
(4)             halftone, allowing for the position of the pixmap within an
(4)             overall banded rendering.
(4)             pix: The pixmap to generate from. Currently must be a single
(4)             color component with no alpha.
(4)             ht: The halftone to use. NULL implies the default halftone.
(4)             band_start: Vertical offset within the overall banded rendering
(4)             (in pixels)
(4)             Returns the resultant bitmap. Throws exceptions in the case of
(4)             failure to allocate.
(4)             """
(4)             return _mupdf.ll_fz_new_bitmap_from_pixmap_band(pix, ht, band_start)
(0)         def ll_fz_new_buffer(capacity):
(4)             r""" Low-level wrapper for `::fz_new_buffer()`."""
(4)             return _mupdf.ll_fz_new_buffer(capacity)
(0)         def ll_fz_new_buffer_from_base64(data, size):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_buffer_from_base64()`.
(4)             Create a new buffer with data decoded from a base64 input string.
(4)             """
(4)             return _mupdf.ll_fz_new_buffer_from_base64(data, size)
(0)         def ll_fz_new_buffer_from_copied_data(data, size):
(4)             r"""
(4)             Low-level wrapper for `::fz_new_buffer_from_copied_data()`.
(4)             Create a new buffer containing a copy of the passed data.
```

```
8151: (4)                """
8152: (4)                return _mupdf.ll_fz_new_buffer_from_copied_data(data, size)
8153: (0)            def ll_fz_new_buffer_from_data(data, size):
8154: (4)                r"""
8155: (4)                Low-level wrapper for `::fz_new_buffer_from_data()`.
8156: (4)                Create a new buffer with existing data.
8157: (4)                data: Pointer to existing data.
8158: (4)                size: Size of existing data.
8159: (4)                Takes ownership of data. Does not make a copy. Calls fz_free on
8160: (4)                the data when the buffer is deallocated. Do not use 'data' after
8161: (4)                passing to this function.
8162: (4)                Returns pointer to new buffer. Throws exception on allocation
8163: (4)                failure.
8164: (4)                """
8165: (4)                return _mupdf.ll_fz_new_buffer_from_data(data, size)
8166: (0)            def ll_fz_new_buffer_from_display_list(list, options):
8167: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_display_list()`."""
8168: (4)                return _mupdf.ll_fz_new_buffer_from_display_list(list, options)
8169: (0)            def ll_fz_new_buffer_from_image_as_jpeg(image, color_params, quality,
invert_cmyk):
8170: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_image_as_jpeg()`."""
8171: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_jpeg(image, color_params,
quality, invert_cmyk)
8172: (0)            def ll_fz_new_buffer_from_image_as_jpx(image, color_params, quality):
8173: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_image_as_jpx()`."""
8174: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_jpx(image, color_params,
quality)
8175: (0)            def ll_fz_new_buffer_from_image_as_pam(image, color_params):
8176: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_image_as_pam()`."""
8177: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_pam(image, color_params)
8178: (0)            def ll_fz_new_buffer_from_image_as_png(image, color_params):
8179: (4)                r"""
8180: (4)                Low-level wrapper for `::fz_new_buffer_from_image_as_png()`.
8181: (4)                Reencode a given image as a PNG into a buffer.
8182: (4)                Ownership of the buffer is returned.
8183: (4)                """
8184: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_png(image, color_params)
8185: (0)            def ll_fz_new_buffer_from_image_as_pnm(image, color_params):
8186: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_image_as_pnm()`."""
8187: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_pnm(image, color_params)
8188: (0)            def ll_fz_new_buffer_from_image_as_psd(image, color_params):
8189: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_image_as_psd()`."""
8190: (4)                return _mupdf.ll_fz_new_buffer_from_image_as_psd(image, color_params)
8191: (0)            def ll_fz_new_buffer_from_page(page, options):
8192: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_page()`."""
8193: (4)                return _mupdf.ll_fz_new_buffer_from_page(page, options)
8194: (0)            def ll_fz_new_buffer_from_page_number(doc, number, options):
8195: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_page_number()`."""
8196: (4)                return _mupdf.ll_fz_new_buffer_from_page_number(doc, number, options)
8197: (0)            def ll_fz_new_buffer_from_page_with_format(page, format, options, transform,
cookie):
8198: (4)                r"""
8199: (4)                Low-level wrapper for `::fz_new_buffer_from_page_with_format()`.
8200: (4)                Returns an fz_buffer containing a page after conversion to specified
format.
8201: (4)                page: The page to convert.
8202: (4)                format, options: Passed to fz_new_document_writer_with_output()
internally.
8203: (4)                transform, cookie: Passed to fz_run_page() internally.
8204: (4)                """
8205: (4)                return _mupdf.ll_fz_new_buffer_from_page_with_format(page, format,
options, transform, cookie)
8206: (0)            def ll_fz_new_buffer_from_pixmap_as_jpeg(pixmap, color_params, quality,
invert_cmyk):
8207: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_pixmap_as_jpeg()`."""
8208: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_jpeg(pixmap, color_params,
quality, invert_cmyk)
8209: (0)            def ll_fz_new_buffer_from_pixmap_as_jpx(pix, color_params, quality):
8210: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_pixmap_as_jpx()`."""
```

```
8211: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_jpx(pix, color_params,
quality)
8212: (0)            def ll_fz_new_buffer_from_pixmap_as_pam(pixmap, color_params):
8213: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_pixmap_as_pam()`."""
8214: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_pam(pixmap, color_params)
8215: (0)            def ll_fz_new_buffer_from_pixmap_as_png(pixmap, color_params):
8216: (4)                r"""
8217: (4)                Low-level wrapper for `::fz_new_buffer_from_pixmap_as_png()`.
8218: (4)                Reencode a given pixmap as a PNG into a buffer.
8219: (4)                Ownership of the buffer is returned.
8220: (4)                """
8221: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_png(pixmap, color_params)
8222: (0)            def ll_fz_new_buffer_from_pixmap_as_pnm(pixmap, color_params):
8223: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_pixmap_as_pnm()`."""
8224: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_pnm(pixmap, color_params)
8225: (0)            def ll_fz_new_buffer_from_pixmap_as_psd(pix, color_params):
8226: (4)                r""" Low-level wrapper for `::fz_new_buffer_from_pixmap_as_psd()`."""
8227: (4)                return _mupdf.ll_fz_new_buffer_from_pixmap_as_psd(pix, color_params)
8228: (0)            def ll_fz_new_buffer_from_shared_data(data, size):
8229: (4)                r"""
8230: (4)                Low-level wrapper for `::fz_new_buffer_from_shared_data()`.
8231: (4)                Like fz_new_buffer, but does not take ownership.
8232: (4)                """
8233: (4)                return _mupdf.ll_fz_new_buffer_from_shared_data(data, size)
8234: (0)            def ll_fz_new_buffer_from_stext_page(text):
8235: (4)                r"""
8236: (4)                Low-level wrapper for `::fz_new_buffer_from_stext_page()`.
8237: (4)                Convert structured text into plain text.
8238: (4)                """
8239: (4)                return _mupdf.ll_fz_new_buffer_from_stext_page(text)
8240: (0)            def ll_fz_new_builtin_font(name, is_bold, is_italic):
8241: (4)                r""" Low-level wrapper for `::fz_new_builtin_font()`."""
8242: (4)                return _mupdf.ll_fz_new_builtin_font(name, is_bold, is_italic)
8243: (0)            def ll_fz_new_cal_gray_colorspace(wp, bp, gamma):
8244: (4)                r"""
8245: (4)                Low-level wrapper for `::fz_new_cal_gray_colorspace()`.
8246: (4)                Create a calibrated gray colorspace.
8247: (4)                The returned reference should be dropped when it is finished
8248: (4)                with.
8249: (4)                Colorspaces are immutable once created.
8250: (4)                """
8251: (4)                return _mupdf.ll_fz_new_cal_gray_colorspace(wp, bp, gamma)
8252: (0)            def ll_fz_new_cal_rgb_colorspace(wp, bp, gamma, matrix):
8253: (4)                r"""
8254: (4)                Low-level wrapper for `::fz_new_cal_rgb_colorspace()`.
8255: (4)                Create a calibrated rgb colorspace.
8256: (4)                The returned reference should be dropped when it is finished
8257: (4)                with.
8258: (4)                Colorspaces are immutable once created.
8259: (4)                """
8260: (4)                return _mupdf.ll_fz_new_cal_rgb_colorspace(wp, bp, gamma, matrix)
8261: (0)            def ll_fz_new_cbz_writer(path, options):
8262: (4)                r""" Low-level wrapper for `::fz_new_cbz_writer()`."""
8263: (4)                return _mupdf.ll_fz_new_cbz_writer(path, options)
8264: (0)            def ll_fz_new_cbz_writer_with_output(out, options):
8265: (4)                r""" Low-level wrapper for `::fz_new_cbz_writer_with_output()`."""
8266: (4)                return _mupdf.ll_fz_new_cbz_writer_with_output(out, options)
8267: (0)            def ll_fz_new_cjk_font(ordering):
8268: (4)                r""" Low-level wrapper for `::fz_new_cjk_font()`."""
8269: (4)                return _mupdf.ll_fz_new_cjk_font(ordering)
8270: (0)            def ll_fz_new_color_pcl_band_writer(out, options):
8271: (4)                r"""
8272: (4)                Low-level wrapper for `::fz_new_color_pcl_band_writer()`.
8273: (4)                Create a new band writer, outputing color pcl.
8274: (4)                """
8275: (4)                return _mupdf.ll_fz_new_color_pcl_band_writer(out, options)
8276: (0)            def ll_fz_new_colorspace(type, flags, n, name):
8277: (4)                r"""
8278: (4)                Low-level wrapper for `::fz_new_colorspace()`.
```

```
8279: (4)                Creates a new colorspace instance and returns a reference.
8280: (4)                No internal checking is done that the colorspace type (e.g.
8281: (4)                CMYK) matches with the flags (e.g. FZ_COLORSPACE_HAS_CMYK) or
8282: (4)                colorant count (n) or name.
8283: (4)                The reference should be dropped when it is finished with.
8284: (4)                Colorspaces are immutable once created (with the exception of
8285: (4)                setting up colorant names for separation spaces).
8286: (4)                """
8287: (4)                return _mupdf.ll_fz_new_colorspace(type, flags, n, name)
8288: (0)            def ll_fz_new_compressed_buffer():
8289: (4)                r"""
8290: (4)                Low-level wrapper for `::fz_new_compressed_buffer()`.
8291: (4)                Create a new, UNKNOWN format, compressed_buffer.
8292: (4)                """
8293: (4)                return _mupdf.ll_fz_new_compressed_buffer()
8294: (0)            def ll_fz_new_context_imp(alloc, locks, max_store, version):
8295: (4)                r""" Low-level wrapper for `::fz_new_context_imp()`."""
8296: (4)                return _mupdf.ll_fz_new_context_imp(alloc, locks, max_store, version)
8297: (0)            def ll_fz_new_default_colorspaces():
8298: (4)                r"""
8299: (4)                Low-level wrapper for `::fz_new_default_colorspaces()`.
8300: (4)                Create a new default colorspace structure with values inherited
8301: (4)                from the context, and return a reference to it.
8302: (4)                These can be overridden using fz_set_default_xxxx.
8303: (4)                These should not be overridden while more than one caller has
8304: (4)                the reference for fear of race conditions.
8305: (4)                The caller should drop this reference once finished with it.
8306: (4)                """
8307: (4)                return _mupdf.ll_fz_new_default_colorspaces()
8308: (0)            def ll_fz_new_deflate_output(chain, effort, raw):
8309: (4)                r""" Low-level wrapper for `::fz_new_deflate_output()`."""
8310: (4)                return _mupdf.ll_fz_new_deflate_output(chain, effort, raw)
8311: (0)            def ll_fz_new_deflated_data(compressed_length, source, source_length, level):
8312: (4)                r"""
8313: (5)                 Low-level wrapper for `::fz_new_deflated_data()`.
8314: (4)                This function has out-params. Python/C# wrappers look like:
8315: (8)                    `ll_fz_new_deflated_data(const unsigned char *source, size_t
source_length, ::fz_deflate_level level)` => `(unsigned char *, size_t compressed_length)`
8316: (8)                    Compress source_length bytes of data starting
8317: (8)                    at source, into a new memory block malloced for that purpose.
8318: (4)                compressed_length is updated on exit to contain the size used.
8319: (8)                    Ownership of the block is returned from this function, and the
8320: (8)                    caller is therefore responsible for freeing it. The block may be
8321: (8)                    considerably larger than is actually required. The caller is
8322: (8)                    free to fz_realloc it down if it wants to.
8323: (4)                """
8324: (4)                return _mupdf.ll_fz_new_deflated_data(compressed_length, source,
source_length, level)
8325: (0)            def ll_fz_new_deflated_data_from_buffer(compressed_length, buffer, level):
8326: (4)                r"""
8327: (5)                 Low-level wrapper for `::fz_new_deflated_data_from_buffer()`.
8328: (4)                This function has out-params. Python/C# wrappers look like:
8329: (8)                    `ll_fz_new_deflated_data_from_buffer(::fz_buffer *buffer,
::fz_deflate_level level)` => `(unsigned char *, size_t compressed_length)`
8330: (8)                    Compress the contents of a fz_buffer into a
8331: (8)                    new block malloced for that purpose. *compressed_length is
8332: (8)                    updated on exit to contain the size used. Ownership of the block
8333: (8)                    is returned from this function, and the caller is therefore
8334: (8)                    responsible for freeing it. The block may be considerably larger
8335: (8)                    than is actually required. The caller is free to fz_realloc it
8336: (8)                    down if it wants to.
8337: (4)                """
8338: (4)                return _mupdf.ll_fz_new_deflated_data_from_buffer(compressed_length,
buffer, level)
8339: (0)            def ll_fz_new_device_of_size(size):
8340: (4)                r"""
8341: (4)                Low-level wrapper for `::fz_new_device_of_size()`.
8342: (4)                Devices are created by calls to device implementations, for
8343: (4)                instance: foo_new_device(). These will be implemented by calling
```

```
8344: (4)               fz_new_derived_device(ctx, foo_device) where foo_device is a
8345: (4)               structure "derived from" fz_device, for instance
8346: (4)               typedef struct { fz_device base;  ...extras...} foo_device;
8347: (4)               """
8348: (4)               return _mupdf.ll_fz_new_device_of_size(size)
8349: (0)           def ll_fz_new_display_list(mediabox):
8350: (4)               r"""
8351: (4)               Low-level wrapper for `::fz_new_display_list()`.
8352: (4)               Create an empty display list.
8353: (4)               A display list contains drawing commands (text, images, etc.).
8354: (4)               Use fz_new_list_device for populating the list.
8355: (4)               mediabox: Bounds of the page (in points) represented by the
8356: (4)               display list.
8357: (4)               """
8358: (4)               return _mupdf.ll_fz_new_display_list(mediabox)
8359: (0)           def ll_fz_new_display_list_from_page(page):
8360: (4)               r"""
8361: (4)               Low-level wrapper for `::fz_new_display_list_from_page()`.
8362: (4)               Create a display list.
8363: (4)               Ownership of the display list is returned to the caller.
8364: (4)               """
8365: (4)               return _mupdf.ll_fz_new_display_list_from_page(page)
8366: (0)           def ll_fz_new_display_list_from_page_contents(page):
8367: (4)               r"""
8368: (4)               Low-level wrapper for `::fz_new_display_list_from_page_contents()`.
8369: (4)               Create a display list from page contents (no annotations).
8370: (4)               Ownership of the display list is returned to the caller.
8371: (4)               """
8372: (4)               return _mupdf.ll_fz_new_display_list_from_page_contents(page)
8373: (0)           def ll_fz_new_display_list_from_page_number(doc, number):
8374: (4)               r""" Low-level wrapper for `::fz_new_display_list_from_page_number()`."""
8375: (4)               return _mupdf.ll_fz_new_display_list_from_page_number(doc, number)
8376: (0)           def ll_fz_new_display_list_from_svg(buf, base_uri, dir, w, h):
8377: (4)               r"""
8378: (5)                Low-level wrapper for `::fz_new_display_list_from_svg()`.
8379: (4)               This function has out-params. Python/C# wrappers look like:
8380: (8)                   `ll_fz_new_display_list_from_svg(::fz_buffer *buf, const char
*base_uri, ::fz_archive *dir)` => `(fz_display_list *, float w, float h)`
8381: (8)                   Parse an SVG document into a display-list.
8382: (4)               """
8383: (4)               return _mupdf.ll_fz_new_display_list_from_svg(buf, base_uri, dir, w, h)
8384: (0)           def ll_fz_new_display_list_from_svg_xml(xmldoc, xml, base_uri, dir, w, h):
8385: (4)               r"""
8386: (5)                Low-level wrapper for `::fz_new_display_list_from_svg_xml()`.
8387: (4)               This function has out-params. Python/C# wrappers look like:
8388: (8)                   `ll_fz_new_display_list_from_svg_xml(::fz_xml_doc *xmldoc, ::fz_xml
*xml, const char *base_uri, ::fz_archive *dir)` => `(fz_display_list *, float w, float h)`
8389: (8)                   Parse an SVG document into a display-list.
8390: (4)               """
8391: (4)               return _mupdf.ll_fz_new_display_list_from_svg_xml(xmldoc, xml, base_uri,
dir, w, h)
8392: (0)           def ll_fz_new_document_of_size(size):
8393: (4)               r"""
8394: (4)               Low-level wrapper for `::fz_new_document_of_size()`.
8395: (4)               New documents are typically created by calls like
8396: (4)               foo_new_document(fz_context *ctx, ...). These work by
8397: (4)               deriving a new document type from fz_document, for instance:
8398: (4)               typedef struct { fz_document base; ...extras... } foo_document;
8399: (4)               These are allocated by calling
8400: (4)               fz_new_derived_document(ctx, foo_document)
8401: (4)               """
8402: (4)               return _mupdf.ll_fz_new_document_of_size(size)
8403: (0)           def ll_fz_new_document_writer(path, format, options):
8404: (4)               r"""
8405: (4)               Low-level wrapper for `::fz_new_document_writer()`.
8406: (4)               Create a new fz_document_writer, for a
8407: (4)               file of the given type.
8408: (4)               path: The document name to write (or NULL for default)
8409: (4)               format: Which format to write (currently cbz, html, pdf, pam,
```

```
8410: (4)                pbm, pgm, pkm, png, ppm, pnm, svg, text, xhtml, docx, odt)
8411: (4)                options: NULL, or pointer to comma separated string to control
8412: (4)                file generation.
8413: (4)                """
8414: (4)                return _mupdf.ll_fz_new_document_writer(path, format, options)
8415: (0)            def ll_fz_new_document_writer_of_size(size, begin_page, end_page, close,
drop):
8416: (4)                r"""
8417: (4)                Low-level wrapper for `::fz_new_document_writer_of_size()`.
8418: (4)                Internal function to allocate a
8419: (4)                block for a derived document_writer structure, with the base
8420: (4)                structure's function pointers populated correctly, and the extra
8421: (4)                space zero initialised.
8422: (4)                """
8423: (4)                return _mupdf.ll_fz_new_document_writer_of_size(size, begin_page,
end_page, close, drop)
8424: (0)            def ll_fz_new_document_writer_with_buffer(buf, format, options):
8425: (4)                r""" Low-level wrapper for `::fz_new_document_writer_with_buffer()`."""
8426: (4)                return _mupdf.ll_fz_new_document_writer_with_buffer(buf, format, options)
8427: (0)            def ll_fz_new_document_writer_with_output(out, format, options):
8428: (4)                r"""
8429: (4)                Low-level wrapper for `::fz_new_document_writer_with_output()`.
8430: (4)                Like fz_new_document_writer but takes a fz_output for writing
8431: (4)                the result. Only works for multi-page formats.
8432: (4)                """
8433: (4)                return _mupdf.ll_fz_new_document_writer_with_output(out, format, options)
8434: (0)            def ll_fz_new_docx_writer(path, options):
8435: (4)                r""" Low-level wrapper for `::fz_new_docx_writer()`."""
8436: (4)                return _mupdf.ll_fz_new_docx_writer(path, options)
8437: (0)            def ll_fz_new_docx_writer_with_output(out, options):
8438: (4)                r""" Low-level wrapper for `::fz_new_docx_writer_with_output()`."""
8439: (4)                return _mupdf.ll_fz_new_docx_writer_with_output(out, options)
8440: (0)            def ll_fz_new_draw_device(transform, dest):
8441: (4)                r"""
8442: (4)                Low-level wrapper for `::fz_new_draw_device()`.
8443: (4)                Create a device to draw on a pixmap.
8444: (4)                dest: Target pixmap for the draw device. See fz_new_pixmap*
8445: (4)                for how to obtain a pixmap. The pixmap is not cleared by the
8446: (4)                draw device, see fz_clear_pixmap* for how to clear it prior to
8447: (4)                calling fz_new_draw_device. Free the device by calling
8448: (4)                fz_drop_device.
8449: (4)                transform: Transform from user space in points to device space
8450: (4)                in pixels.
8451: (4)                """
8452: (4)                return _mupdf.ll_fz_new_draw_device(transform, dest)
8453: (0)            def ll_fz_new_draw_device_type3(transform, dest):
8454: (4)                r""" Low-level wrapper for `::fz_new_draw_device_type3()`."""
8455: (4)                return _mupdf.ll_fz_new_draw_device_type3(transform, dest)
8456: (0)            def ll_fz_new_draw_device_with_bbox(transform, dest, clip):
8457: (4)                r"""
8458: (4)                Low-level wrapper for `::fz_new_draw_device_with_bbox()`.
8459: (4)                Create a device to draw on a pixmap.
8460: (4)                dest: Target pixmap for the draw device. See fz_new_pixmap*
8461: (4)                for how to obtain a pixmap. The pixmap is not cleared by the
8462: (4)                draw device, see fz_clear_pixmap* for how to clear it prior to
8463: (4)                calling fz_new_draw_device. Free the device by calling
8464: (4)                fz_drop_device.
8465: (4)                transform: Transform from user space in points to device space
8466: (4)                in pixels.
8467: (4)                clip: Bounding box to restrict any marking operations of the
8468: (4)                draw device.
8469: (4)                """
8470: (4)                return _mupdf.ll_fz_new_draw_device_with_bbox(transform, dest, clip)
8471: (0)            def ll_fz_new_draw_device_with_bbox_proof(transform, dest, clip, cs):
8472: (4)                r"""
8473: (4)                Low-level wrapper for `::fz_new_draw_device_with_bbox_proof()`.
8474: (4)                Create a device to draw on a pixmap.
8475: (4)                dest: Target pixmap for the draw device. See fz_new_pixmap*
8476: (4)                for how to obtain a pixmap. The pixmap is not cleared by the
```

```
8477: (4)               draw device, see fz_clear_pixmap* for how to clear it prior to
8478: (4)               calling fz_new_draw_device. Free the device by calling
8479: (4)               fz_drop_device.
8480: (4)               transform: Transform from user space in points to device space
8481: (4)               in pixels.
8482: (4)               clip: Bounding box to restrict any marking operations of the
8483: (4)               draw device.
8484: (4)               proof_cs: Color space to render to prior to mapping to color
8485: (4)               space defined by pixmap.
8486: (4)               """
8487: (4)               return _mupdf.ll_fz_new_draw_device_with_bbox_proof(transform, dest, clip,
cs)
8488: (0)           def ll_fz_new_draw_device_with_options(options, mediabox, pixmap):
8489: (4)               r"""
8490: (5)                Low-level wrapper for `::fz_new_draw_device_with_options()`.
8491: (4)               This function has out-params. Python/C# wrappers look like:
8492: (8)                   `ll_fz_new_draw_device_with_options(const ::fz_draw_options *options,
::fz_rect mediabox, ::fz_pixmap **pixmap)` => `(fz_device *)`
8493: (8)                   Create a new pixmap and draw device, using the specified options.
8494: (8)                   options: Options to configure the draw device, and choose the
8495: (8)                   resolution and colorspace.
8496: (8)                   mediabox: The bounds of the page in points.
8497: (8)                   pixmap: An out parameter containing the newly created pixmap.
8498: (4)               """
8499: (4)               return _mupdf.ll_fz_new_draw_device_with_options(options, mediabox,
pixmap)
8500: (0)           def ll_fz_new_draw_device_with_proof(transform, dest, proof_cs):
8501: (4)               r"""
8502: (4)               Low-level wrapper for `::fz_new_draw_device_with_proof()`.
8503: (4)               Create a device to draw on a pixmap.
8504: (4)               dest: Target pixmap for the draw device. See fz_new_pixmap*
8505: (4)               for how to obtain a pixmap. The pixmap is not cleared by the
8506: (4)               draw device, see fz_clear_pixmap* for how to clear it prior to
8507: (4)               calling fz_new_draw_device. Free the device by calling
8508: (4)               fz_drop_device.
8509: (4)               transform: Transform from user space in points to device space
8510: (4)               in pixels.
8511: (4)               proof_cs: Intermediate color space to map though when mapping to
8512: (4)               color space defined by pixmap.
8513: (4)               """
8514: (4)               return _mupdf.ll_fz_new_draw_device_with_proof(transform, dest, proof_cs)
8515: (0)           def ll_fz_new_font_from_buffer(name, buffer, index, use_glyph_bbox):
8516: (4)               r"""
8517: (4)               Low-level wrapper for `::fz_new_font_from_buffer()`.
8518: (4)               Create a new font from a font file in a fz_buffer.
8519: (4)               Fonts created in this way, will be eligible for embedding by default.
8520: (4)               name: Name of font (leave NULL to use name from font).
8521: (4)               buffer: Buffer to load from.
8522: (4)               index: Which font from the file to load (0 for default).
8523: (4)               use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
8524: (4)               Returns new font handle, or throws exception on error.
8525: (4)               """
8526: (4)               return _mupdf.ll_fz_new_font_from_buffer(name, buffer, index,
use_glyph_bbox)
8527: (0)           def ll_fz_new_font_from_file(name, path, index, use_glyph_bbox):
8528: (4)               r"""
8529: (4)               Low-level wrapper for `::fz_new_font_from_file()`.
8530: (4)               Create a new font from a font file.
8531: (4)               Fonts created in this way, will be eligible for embedding by default.
8532: (4)               name: Name of font (leave NULL to use name from font).
8533: (4)               path: File path to load from.
8534: (4)               index: Which font from the file to load (0 for default).
8535: (4)               use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
8536: (4)               Returns new font handle, or throws exception on error.
8537: (4)               """
8538: (4)               return _mupdf.ll_fz_new_font_from_file(name, path, index, use_glyph_bbox)
8539: (0)           def ll_fz_new_font_from_memory(name, data, len, index, use_glyph_bbox):
8540: (4)               r"""
8541: (4)               Low-level wrapper for `::fz_new_font_from_memory()`.
```

```
8542: (4)                Create a new font from a font file in memory.
8543: (4)                Fonts created in this way, will be eligible for embedding by default.
8544: (4)                name: Name of font (leave NULL to use name from font).
8545: (4)                data: Pointer to the font file data.
8546: (4)                len: Length of the font file data.
8547: (4)                index: Which font from the file to load (0 for default).
8548: (4)                use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
8549: (4)                Returns new font handle, or throws exception on error.
8550: (4)                """
8551: (4)                return _mupdf.ll_fz_new_font_from_memory(name, data, len, index,
use_glyph_bbox)
8552: (0)            def ll_fz_new_function_of_size(size, size2, m, n, eval, drop):
8553: (4)                r""" Low-level wrapper for `::fz_new_function_of_size()`."""
8554: (4)                return _mupdf.ll_fz_new_function_of_size(size, size2, m, n, eval, drop)
8555: (0)            def ll_fz_new_hash_table(initialsize, keylen, lock, drop_val):
8556: (4)                r"""
8557: (4)                Low-level wrapper for `::fz_new_hash_table()`.
8558: (4)                Create a new hash table.
8559: (4)                initialsize: The initial size of the hashtable. The hashtable
8560: (4)                may grow (double in size) if it starts to get crowded (80%
8561: (4)                full).
8562: (4)                keylen: byte length for each key.
8563: (4)                lock: -1 for no lock, otherwise the FZ_LOCK to use to protect
8564: (4)                this table.
8565: (4)                drop_val: Function to use to destroy values on table drop.
8566: (4)                """
8567: (4)                return _mupdf.ll_fz_new_hash_table(initialsize, keylen, lock, drop_val)
8568: (0)            def ll_fz_new_icc_colorspace(type, flags, name, buf):
8569: (4)                r"""
8570: (4)                Low-level wrapper for `::fz_new_icc_colorspace()`.
8571: (4)                Create a colorspace from an ICC profile supplied in buf.
8572: (4)                Limited checking is done to ensure that the colorspace type is
8573: (4)                appropriate for the supplied ICC profile.
8574: (4)                An additional reference is taken to buf, which will be dropped
8575: (4)                on destruction. Ownership is NOT passed in.
8576: (4)                The returned reference should be dropped when it is finished
8577: (4)                with.
8578: (4)                Colorspaces are immutable once created.
8579: (4)                """
8580: (4)                return _mupdf.ll_fz_new_icc_colorspace(type, flags, name, buf)
8581: (0)            def ll_fz_new_image_from_buffer(buffer):
8582: (4)                r"""
8583: (4)                Low-level wrapper for `::fz_new_image_from_buffer()`.
8584: (4)                Create a new image from a
8585: (4)                buffer of data, inferring its type from the format
8586: (4)                of the data.
8587: (4)                """
8588: (4)                return _mupdf.ll_fz_new_image_from_buffer(buffer)
8589: (0)            def ll_fz_new_image_from_compressed_buffer(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, buffer, mask):
8590: (4)                r"""
8591: (4)                Low-level wrapper for `::fz_new_image_from_compressed_buffer()`.
8592: (4)                Create an image based on
8593: (4)                the data in the supplied compressed buffer.
8594: (4)                w,h: Width and height of the created image.
8595: (4)                bpc: Bits per component.
8596: (4)                colorspace: The colorspace (determines the number of components,
8597: (4)                and any color conversions required while decoding).
8598: (4)                xres, yres: The X and Y resolutions respectively.
8599: (4)                interpolate: 1 if interpolation should be used when decoding
8600: (4)                this image, 0 otherwise.
8601: (4)                imagemask: 1 if this is an imagemask (i.e. transparency bitmap
8602: (4)                mask), 0 otherwise.
8603: (4)                decode: NULL, or a pointer to to a decode array. The default
8604: (4)                decode array is [0 1] (repeated n times, for n color components).
8605: (4)                colorkey: NULL, or a pointer to a colorkey array. The default
8606: (4)                colorkey array is [0 255] (repeated n times, for n color
8607: (4)                components).
8608: (4)                buffer: Buffer of compressed data and compression parameters.
```

```
8609: (4)               Ownership of this reference is passed in.
8610: (4)               mask: NULL, or another image to use as a mask for this one.
8611: (4)               A new reference is taken to this image. Supplying a masked
8612: (4)               image as a mask to another image is illegal!
8613: (4)               """
8614: (4)               return _mupdf.ll_fz_new_image_from_compressed_buffer(w, h, bpc,
colorspace, xres, yres, interpolate, imagemask, decode, colorkey, buffer, mask)
8615: (0)           def ll_fz_new_image_from_compressed_buffer2(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, buffer, mask):
8616: (4)               r"""
8617: (5)                Low-level wrapper for `::fz_new_image_from_compressed_buffer2()`.  Swig-
friendly wrapper for fz_new_image_from_compressed_buffer(),
8618: (4)               uses specified `decode` and `colorkey` if they are not null (in which
8619: (4)               case we assert that they have size `2*fz_colorspace_n(colorspace)`).
8620: (4)               """
8621: (4)               return _mupdf.ll_fz_new_image_from_compressed_buffer2(w, h, bpc,
colorspace, xres, yres, interpolate, imagemask, decode, colorkey, buffer, mask)
8622: (0)           def ll_fz_new_image_from_display_list(w, h, list):
8623: (4)               r"""
8624: (4)               Low-level wrapper for `::fz_new_image_from_display_list()`.
8625: (4)               Create a new image from a display list.
8626: (4)               w, h: The conceptual width/height of the image.
8627: (4)               transform: The matrix that needs to be applied to the given
8628: (4)               list to make it render to the unit square.
8629: (4)               list: The display list.
8630: (4)               """
8631: (4)               return _mupdf.ll_fz_new_image_from_display_list(w, h, list)
8632: (0)           def ll_fz_new_image_from_file(path):
8633: (4)               r"""
8634: (4)               Low-level wrapper for `::fz_new_image_from_file()`.
8635: (4)               Create a new image from the contents
8636: (4)               of a file, inferring its type from the format of the
8637: (4)               data.
8638: (4)               """
8639: (4)               return _mupdf.ll_fz_new_image_from_file(path)
8640: (0)           def ll_fz_new_image_from_pixmap(pixmap, mask):
8641: (4)               r"""
8642: (4)               Low-level wrapper for `::fz_new_image_from_pixmap()`.
8643: (4)               Create an image from the given
8644: (4)               pixmap.
8645: (4)               pixmap: The pixmap to base the image upon. A new reference
8646: (4)               to this is taken.
8647: (4)               mask: NULL, or another image to use as a mask for this one.
8648: (4)               A new reference is taken to this image. Supplying a masked
8649: (4)               image as a mask to another image is illegal!
8650: (4)               """
8651: (4)               return _mupdf.ll_fz_new_image_from_pixmap(pixmap, mask)
8652: (0)           def ll_fz_new_image_from_svg(buf, base_uri, dir):
8653: (4)               r"""
8654: (4)               Low-level wrapper for `::fz_new_image_from_svg()`.
8655: (4)               Create a scalable image from an SVG document.
8656: (4)               """
8657: (4)               return _mupdf.ll_fz_new_image_from_svg(buf, base_uri, dir)
8658: (0)           def ll_fz_new_image_from_svg_xml(xmldoc, xml, base_uri, dir):
8659: (4)               r"""
8660: (4)               Low-level wrapper for `::fz_new_image_from_svg_xml()`.
8661: (4)               Create a scalable image from an SVG document.
8662: (4)               """
8663: (4)               return _mupdf.ll_fz_new_image_from_svg_xml(xmldoc, xml, base_uri, dir)
8664: (0)           def ll_fz_new_image_of_size(w, h, bpc, colorspace, xres, yres, interpolate,
imagemask, decode, colorkey, mask, size, get_pixmap, get_size, drop):
8665: (4)               r"""
8666: (4)               Low-level wrapper for `::fz_new_image_of_size()`.
8667: (4)               Internal function to make a new fz_image structure
8668: (4)               for a derived class.
8669: (4)               w,h: Width and height of the created image.
8670: (4)               bpc: Bits per component.
8671: (4)               colorspace: The colorspace (determines the number of components,
8672: (4)               and any color conversions required while decoding).
```

```
8673: (4)                xres, yres: The X and Y resolutions respectively.
8674: (4)                interpolate: 1 if interpolation should be used when decoding
8675: (4)                this image, 0 otherwise.
8676: (4)                imagemask: 1 if this is an imagemask (i.e. transparent), 0
8677: (4)                otherwise.
8678: (4)                decode: NULL, or a pointer to to a decode array. The default
8679: (4)                decode array is [0 1] (repeated n times, for n color components).
8680: (4)                colorkey: NULL, or a pointer to a colorkey array. The default
8681: (4)                colorkey array is [0 255] (repeated n times, for n color
8682: (4)                components).
8683: (4)                mask: NULL, or another image to use as a mask for this one.
8684: (4)                A new reference is taken to this image. Supplying a masked
8685: (4)                image as a mask to another image is illegal!
8686: (4)                size: The size of the required allocated structure (the size of
8687: (4)                the derived structure).
8688: (4)                get: The function to be called to obtain a decoded pixmap.
8689: (4)                get_size: The function to be called to return the storage size
8690: (4)                used by this image.
8691: (4)                drop: The function to be called to dispose of this image once
8692: (4)                the last reference is dropped.
8693: (4)                Returns a pointer to an allocated structure of the required size,
8694: (4)                with the first sizeof(fz_image) bytes initialised as appropriate
8695: (4)                given the supplied parameters, and the other bytes set to zero.
8696: (4)                """
8697: (4)                return _mupdf.ll_fz_new_image_of_size(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, mask, size, get_pixmap, get_size, drop)
8698: (0)            def ll_fz_new_indexed_colorspace(base, high, lookup):
8699: (4)                r"""
8700: (4)                Low-level wrapper for `::fz_new_indexed_colorspace()`.
8701: (4)                Create an indexed colorspace.
8702: (4)                The supplied lookup table is high palette entries long. Each
8703: (4)                entry is n bytes long, where n is given by the number of
8704: (4)                colorants in the base colorspace, one byte per colorant.
8705: (4)                Ownership of lookup is passed it; it will be freed on
8706: (4)                destruction, so must be heap allocated.
8707: (4)                The colorspace will keep an additional reference to the base
8708: (4)                colorspace that will be dropped on destruction.
8709: (4)                The returned reference should be dropped when it is finished
8710: (4)                with.
8711: (4)                Colorspaces are immutable once created.
8712: (4)                """
8713: (4)                return _mupdf.ll_fz_new_indexed_colorspace(base, high, lookup)
8714: (0)            def ll_fz_new_jpeg_pixmap_writer(path, options):
8715: (4)                r""" Low-level wrapper for `::fz_new_jpeg_pixmap_writer()`."""
8716: (4)                return _mupdf.ll_fz_new_jpeg_pixmap_writer(path, options)
8717: (0)            def ll_fz_new_layout():
8718: (4)                r"""
8719: (4)                Low-level wrapper for `::fz_new_layout()`.
8720: (4)                Create a new layout block, with new allocation pool, zero
8721: (4)                matrices, and initialise linked pointers.
8722: (4)                """
8723: (4)                return _mupdf.ll_fz_new_layout()
8724: (0)            def ll_fz_new_link_of_size(size, rect, uri):
8725: (4)                r"""
8726: (4)                Low-level wrapper for `::fz_new_link_of_size()`.
8727: (4)                Create a new link record.
8728: (4)                next is set to NULL with the expectation that the caller will
8729: (4)                handle the linked list setup. Internal function.
8730: (4)                Different document types will be implemented by deriving from
8731: (4)                fz_link. This macro allocates such derived structures, and
8732: (4)                initialises the base sections.
8733: (4)                """
8734: (4)                return _mupdf.ll_fz_new_link_of_size(size, rect, uri)
8735: (0)            def ll_fz_new_list_device(list):
8736: (4)                r"""
8737: (4)                Low-level wrapper for `::fz_new_list_device()`.
8738: (4)                Create a rendering device for a display list.
8739: (4)                When the device is rendering a page it will populate the
8740: (4)                display list with drawing commands (text, images, etc.). The
```

```
8741: (4)               display list can later be reused to render a page many times
8742: (4)               without having to re-interpret the page from the document file
8743: (4)               for each rendering. Once the device is no longer needed, free
8744: (4)               it with fz_drop_device.
8745: (4)               list: A display list that the list device takes a reference to.
8746: (4)               """
8747: (4)               return _mupdf.ll_fz_new_list_device(list)
8748: (0)           def ll_fz_new_log_for_module(module):
8749: (4)               r"""
8750: (4)               Low-level wrapper for `::fz_new_log_for_module()`.
8751: (4)               Internal function to actually do the opening of the logfile.
8752: (4)               Caller should close/drop the output when finished with it.
8753: (4)               """
8754: (4)               return _mupdf.ll_fz_new_log_for_module(module)
8755: (0)           def ll_fz_new_mono_pcl_band_writer(out, options):
8756: (4)               r"""
8757: (4)               Low-level wrapper for `::fz_new_mono_pcl_band_writer()`.
8758: (4)               Create a new band writer, outputing monochrome pcl.
8759: (4)               """
8760: (4)               return _mupdf.ll_fz_new_mono_pcl_band_writer(out, options)
8761: (0)           def ll_fz_new_mono_pwg_band_writer(out, pwg):
8762: (4)               r"""
8763: (4)               Low-level wrapper for `::fz_new_mono_pwg_band_writer()`.
8764: (4)               Create a new monochrome pwg band writer.
8765: (4)               """
8766: (4)               return _mupdf.ll_fz_new_mono_pwg_band_writer(out, pwg)
8767: (0)           def ll_fz_new_multi_archive():
8768: (4)               r"""
8769: (4)               Low-level wrapper for `::fz_new_multi_archive()`.
8770: (4)               Create a new multi archive (initially empty).
8771: (4)               """
8772: (4)               return _mupdf.ll_fz_new_multi_archive()
8773: (0)           def ll_fz_new_ocr_device(target, ctm, mediabox, with_list, language, datadir,
progress, progress_arg):
8774: (4)               r"""
8775: (4)               Low-level wrapper for `::fz_new_ocr_device()`.
8776: (4)               Create a device to OCR the text on the page.
8777: (4)               Renders the page internally to a bitmap that is then OCRd. Text
8778: (4)               is then forwarded onto the target device.
8779: (4)               target: The target device to receive the OCRd text.
8780: (4)               ctm: The transform to apply to the mediabox to get the size for
8781: (4)               the rendered page image. Also used to calculate the resolution
8782: (4)               for the page image. In general, this will be the same as the CTM
8783: (4)               that you pass to fz_run_page (or fz_run_display_list) to feed
8784: (4)               this device.
8785: (4)               mediabox: The mediabox (in points). Combined with the CTM to get
8786: (4)               the bounds of the pixmap used internally for the rendered page
8787: (4)               image.
8788: (4)               with_list: If with_list is false, then all non-text operations
8789: (4)               are forwarded instantly to the target device. This results in
8790: (4)               the target device seeing all NON-text operations, followed by
8791: (4)               all the text operations (derived from OCR).
8792: (4)               If with_list is true, then all the marking operations are
8793: (4)               collated into a display list which is then replayed to the
8794: (4)               target device at the end.
8795: (4)               language: NULL (for "eng"), or a pointer to a string to describe
8796: (4)               the languages/scripts that should be used for OCR (e.g.
8797: (4)               "eng,ara").
8798: (4)               datadir: NULL (for ""), or a pointer to a path string otherwise
8799: (4)               provided to Tesseract in the TESSDATA_PREFIX environment variable.
8800: (4)               progress: NULL, or function to be called periodically to indicate
8801: (4)               progress. Return 0 to continue, or 1 to cancel. progress_arg is
8802: (4)               returned as the void *. The int is a value between 0 and 100 to
8803: (4)               indicate progress.
8804: (4)               progress_arg: A void * value to be parrotted back to the progress
8805: (4)               function.
8806: (4)               """
8807: (4)               return _mupdf.ll_fz_new_ocr_device(target, ctm, mediabox, with_list,
language, datadir, progress, progress_arg)
```

```
8808: (0)           def ll_fz_new_odt_writer(path, options):
8809: (4)               r""" Low-level wrapper for `::fz_new_odt_writer()`."""
8810: (4)               return _mupdf.ll_fz_new_odt_writer(path, options)
8811: (0)           def ll_fz_new_odt_writer_with_output(out, options):
8812: (4)               r""" Low-level wrapper for `::fz_new_odt_writer_with_output()`."""
8813: (4)               return _mupdf.ll_fz_new_odt_writer_with_output(out, options)
8814: (0)           def ll_fz_new_outline():
8815: (4)               r"""
8816: (4)               Low-level wrapper for `::fz_new_outline()`.
8817: (4)               Create a new outline entry with zeroed fields for the caller
8818: (4)               to fill in.
8819: (4)               """
8820: (4)               return _mupdf.ll_fz_new_outline()
8821: (0)           def ll_fz_new_outline_iterator(doc):
8822: (4)               r"""
8823: (4)               Low-level wrapper for `::fz_new_outline_iterator()`.
8824: (4)               Get an iterator for the document outline.
8825: (4)               Should be freed by fz_drop_outline_iterator.
8826: (4)               """
8827: (4)               return _mupdf.ll_fz_new_outline_iterator(doc)
8828: (0)           def ll_fz_new_outline_iterator_of_size(size, doc):
8829: (4)               r""" Low-level wrapper for `::fz_new_outline_iterator_of_size()`."""
8830: (4)               return _mupdf.ll_fz_new_outline_iterator_of_size(size, doc)
8831: (0)           def ll_fz_new_output(bufsiz, state, write, close, drop):
8832: (4)               r"""
8833: (4)               Low-level wrapper for `::fz_new_output()`.
8834: (4)               Create a new output object with the given
8835: (4)               internal state and function pointers.
8836: (4)               state: Internal state (opaque to everything but implementation).
8837: (4)               write: Function to output a given buffer.
8838: (4)               close: Cleanup function to destroy state when output closed.
8839: (4)               May permissibly be null.
8840: (4)               """
8841: (4)               return _mupdf.ll_fz_new_output(bufsiz, state, write, close, drop)
8842: (0)           def ll_fz_new_output_with_buffer(buf):
8843: (4)               r"""
8844: (4)               Low-level wrapper for `::fz_new_output_with_buffer()`.
8845: (4)               Open an output stream that appends
8846: (4)               to a buffer.
8847: (4)               buf: The buffer to append to.
8848: (4)               """
8849: (4)               return _mupdf.ll_fz_new_output_with_buffer(buf)
8850: (0)           def ll_fz_new_output_with_file_ptr(file):
8851: (4)               r"""
8852: (4)               Low-level wrapper for `::fz_new_output_with_file_ptr()`.
8853: (4)               Open an output stream that writes to a
8854: (4)               given FILE *.
8855: (4)               file: The file pointers to write to. NULL is interpreted as effectively
8856: (4)               meaning /dev/null or similar.
8857: (4)               """
8858: (4)               return _mupdf.ll_fz_new_output_with_file_ptr(file)
8859: (0)           def ll_fz_new_output_with_path(filename, append):
8860: (4)               r"""
8861: (4)               Low-level wrapper for `::fz_new_output_with_path()`.
8862: (4)               Open an output stream that writes to a
8863: (4)               given path.
8864: (4)               filename: The filename to write to (specified in UTF-8).
8865: (4)               append: non-zero if we should append to the file, rather than
8866: (4)               overwriting it.
8867: (4)               """
8868: (4)               return _mupdf.ll_fz_new_output_with_path(filename, append)
8869: (0)           def ll_fz_new_page_of_size(size, doc):
8870: (4)               r"""
8871: (4)               Low-level wrapper for `::fz_new_page_of_size()`.
8872: (4)               Different document types will be implemented by deriving from
8873: (4)               fz_page. This macro allocates such derived structures, and
8874: (4)               initialises the base sections.
8875: (4)               """
8876: (4)               return _mupdf.ll_fz_new_page_of_size(size, doc)
```

```
8877: (0)           def ll_fz_new_pam_band_writer(out):
8878: (4)               r"""
8879: (4)               Low-level wrapper for `::fz_new_pam_band_writer()`.
8880: (4)               Create a band writer targetting pnm (greyscale, rgb or cmyk,
8881: (4)               with or without alpha).
8882: (4)               """
8883: (4)               return _mupdf.ll_fz_new_pam_band_writer(out)
8884: (0)           def ll_fz_new_pam_pixmap_writer(path, options):
8885: (4)               r""" Low-level wrapper for `::fz_new_pam_pixmap_writer()`."""
8886: (4)               return _mupdf.ll_fz_new_pam_pixmap_writer(path, options)
8887: (0)           def ll_fz_new_path():
8888: (4)               r"""
8889: (4)               Low-level wrapper for `::fz_new_path()`.
8890: (4)               Create a new (empty) path structure.
8891: (4)               """
8892: (4)               return _mupdf.ll_fz_new_path()
8893: (0)           def ll_fz_new_pbm_band_writer(out):
8894: (4)               r"""
8895: (4)               Low-level wrapper for `::fz_new_pbm_band_writer()`.
8896: (4)               Create a new band writer, targetting pbm.
8897: (4)               """
8898: (4)               return _mupdf.ll_fz_new_pbm_band_writer(out)
8899: (0)           def ll_fz_new_pbm_pixmap_writer(path, options):
8900: (4)               r""" Low-level wrapper for `::fz_new_pbm_pixmap_writer()`."""
8901: (4)               return _mupdf.ll_fz_new_pbm_pixmap_writer(path, options)
8902: (0)           def ll_fz_new_pcl_writer(path, options):
8903: (4)               r""" Low-level wrapper for `::fz_new_pcl_writer()`."""
8904: (4)               return _mupdf.ll_fz_new_pcl_writer(path, options)
8905: (0)           def ll_fz_new_pcl_writer_with_output(out, options):
8906: (4)               r""" Low-level wrapper for `::fz_new_pcl_writer_with_output()`."""
8907: (4)               return _mupdf.ll_fz_new_pcl_writer_with_output(out, options)
8908: (0)           def ll_fz_new_pclm_band_writer(out, options):
8909: (4)               r"""
8910: (4)               Low-level wrapper for `::fz_new_pclm_band_writer()`.
8911: (4)               Create a new band writer, outputing pclm
8912: (4)               """
8913: (4)               return _mupdf.ll_fz_new_pclm_band_writer(out, options)
8914: (0)           def ll_fz_new_pclm_writer(path, options):
8915: (4)               r""" Low-level wrapper for `::fz_new_pclm_writer()`."""
8916: (4)               return _mupdf.ll_fz_new_pclm_writer(path, options)
8917: (0)           def ll_fz_new_pclm_writer_with_output(out, options):
8918: (4)               r""" Low-level wrapper for `::fz_new_pclm_writer_with_output()`."""
8919: (4)               return _mupdf.ll_fz_new_pclm_writer_with_output(out, options)
8920: (0)           def ll_fz_new_pdf_document_from_fz_document(ptr):
8921: (4)               r""" Low-level wrapper for `::fz_new_pdf_document_from_fz_document()`."""
8922: (4)               return _mupdf.ll_fz_new_pdf_document_from_fz_document(ptr)
8923: (0)           def ll_fz_new_pdf_writer(path, options):
8924: (4)               r"""
8925: (4)               Low-level wrapper for `::fz_new_pdf_writer()`.
8926: (4)               Document writers for various possible output formats.
8927: (4)               All of the "_with_output" variants pass the ownership of out in
8928: (4)               immediately upon calling. The writers are responsible for
8929: (4)               dropping the fz_output when they are finished with it (even
8930: (4)               if they throw an exception during creation).
8931: (4)               """
8932: (4)               return _mupdf.ll_fz_new_pdf_writer(path, options)
8933: (0)           def ll_fz_new_pdf_writer_with_output(out, options):
8934: (4)               r""" Low-level wrapper for `::fz_new_pdf_writer_with_output()`."""
8935: (4)               return _mupdf.ll_fz_new_pdf_writer_with_output(out, options)
8936: (0)           def ll_fz_new_pdfocr_band_writer(out, options):
8937: (4)               r"""
8938: (4)               Low-level wrapper for `::fz_new_pdfocr_band_writer()`.
8939: (4)               Create a new band writer, outputing pdfocr.
8940: (4)               Ownership of output stays with the caller, the band writer
8941: (4)               borrows the reference. The caller must keep the output around
8942: (4)               for the duration of the band writer, and then close/drop as
8943: (4)               appropriate.
8944: (4)               """
8945: (4)               return _mupdf.ll_fz_new_pdfocr_band_writer(out, options)
```

```
8946: (0)           def ll_fz_new_pdfocr_writer(path, options):
8947: (4)               r""" Low-level wrapper for `::fz_new_pdfocr_writer()`."""
8948: (4)               return _mupdf.ll_fz_new_pdfocr_writer(path, options)
8949: (0)           def ll_fz_new_pdfocr_writer_with_output(out, options):
8950: (4)               r""" Low-level wrapper for `::fz_new_pdfocr_writer_with_output()`."""
8951: (4)               return _mupdf.ll_fz_new_pdfocr_writer_with_output(out, options)
8952: (0)           def ll_fz_new_pgm_pixmap_writer(path, options):
8953: (4)               r""" Low-level wrapper for `::fz_new_pgm_pixmap_writer()`."""
8954: (4)               return _mupdf.ll_fz_new_pgm_pixmap_writer(path, options)
8955: (0)           def ll_fz_new_pixmap(cs, w, h, seps, alpha):
8956: (4)               r"""
8957: (4)               Low-level wrapper for `::fz_new_pixmap()`.
8958: (4)               Create a new pixmap, with its origin at (0,0)
8959: (4)               cs: The colorspace to use for the pixmap, or NULL for an alpha
8960: (4)               plane/mask.
8961: (4)               w: The width of the pixmap (in pixels)
8962: (4)               h: The height of the pixmap (in pixels)
8963: (4)               seps: Details of separations.
8964: (4)               alpha: 0 for no alpha, 1 for alpha.
8965: (4)               Returns a pointer to the new pixmap. Throws exception on failure
8966: (4)               to allocate.
8967: (4)               """
8968: (4)               return _mupdf.ll_fz_new_pixmap(cs, w, h, seps, alpha)
8969: (0)           def ll_fz_new_pixmap_from_alpha_channel(src):
8970: (4)               r""" Low-level wrapper for `::fz_new_pixmap_from_alpha_channel()`."""
8971: (4)               return _mupdf.ll_fz_new_pixmap_from_alpha_channel(src)
8972: (0)           def ll_fz_new_pixmap_from_color_and_mask(color, mask):
8973: (4)               r""" Low-level wrapper for `::fz_new_pixmap_from_color_and_mask()`."""
8974: (4)               return _mupdf.ll_fz_new_pixmap_from_color_and_mask(color, mask)
8975: (0)           def ll_fz_new_pixmap_from_display_list(list, ctm, cs, alpha):
8976: (4)               r"""
8977: (4)               Low-level wrapper for `::fz_new_pixmap_from_display_list()`.
8978: (4)               Render the page to a pixmap using the transform and colorspace.
8979: (4)               Ownership of the pixmap is returned to the caller.
8980: (4)               """
8981: (4)               return _mupdf.ll_fz_new_pixmap_from_display_list(list, ctm, cs, alpha)
8982: (0)           def ll_fz_new_pixmap_from_display_list_with_separations(list, ctm, cs, seps,
alpha):
8983: (4)               r"""
8984: (4)               Low-level wrapper for
`::fz_new_pixmap_from_display_list_with_separations()`.
8985: (4)               Render the page contents with control over spot colors.
8986: (4)               Ownership of the pixmap is returned to the caller.
8987: (4)               """
8988: (4)               return _mupdf.ll_fz_new_pixmap_from_display_list_with_separations(list,
ctm, cs, seps, alpha)
8989: (0)           def ll_fz_new_pixmap_from_page(page, ctm, cs, alpha):
8990: (4)               r""" Low-level wrapper for `::fz_new_pixmap_from_page()`."""
8991: (4)               return _mupdf.ll_fz_new_pixmap_from_page(page, ctm, cs, alpha)
8992: (0)           def ll_fz_new_pixmap_from_page_contents(page, ctm, cs, alpha):
8993: (4)               r"""
8994: (4)               Low-level wrapper for `::fz_new_pixmap_from_page_contents()`.
8995: (4)               Render the page contents without annotations.
8996: (4)               Ownership of the pixmap is returned to the caller.
8997: (4)               """
8998: (4)               return _mupdf.ll_fz_new_pixmap_from_page_contents(page, ctm, cs, alpha)
8999: (0)           def ll_fz_new_pixmap_from_page_contents_with_separations(page, ctm, cs, seps,
alpha):
9000: (4)               r""" Low-level wrapper for
`::fz_new_pixmap_from_page_contents_with_separations()`."""
9001: (4)               return _mupdf.ll_fz_new_pixmap_from_page_contents_with_separations(page,
ctm, cs, seps, alpha)
9002: (0)           def ll_fz_new_pixmap_from_page_number(doc, number, ctm, cs, alpha):
9003: (4)               r""" Low-level wrapper for `::fz_new_pixmap_from_page_number()`."""
9004: (4)               return _mupdf.ll_fz_new_pixmap_from_page_number(doc, number, ctm, cs,
alpha)
9005: (0)           def ll_fz_new_pixmap_from_page_number_with_separations(doc, number, ctm, cs,
seps, alpha):
9006: (4)               r""" Low-level wrapper for
```

```
`::fz_new_pixmap_from_page_number_with_separations()`."""
9007: (4)               return _mupdf.ll_fz_new_pixmap_from_page_number_with_separations(doc,
number, ctm, cs, seps, alpha)
9008: (0)           def ll_fz_new_pixmap_from_page_with_separations(page, ctm, cs, seps, alpha):
9009: (4)               r""" Low-level wrapper for
`::fz_new_pixmap_from_page_with_separations()`."""
9010: (4)               return _mupdf.ll_fz_new_pixmap_from_page_with_separations(page, ctm, cs,
seps, alpha)
9011: (0)           def ll_fz_new_pixmap_from_pixmap(pixmap, rect):
9012: (4)               r"""
9013: (4)               Low-level wrapper for `::fz_new_pixmap_from_pixmap()`.
9014: (4)               Create a new pixmap that represents a subarea of the specified
9015: (4)               pixmap. A reference is taken to this pixmap that will be dropped
9016: (4)               on destruction.
9017: (4)               The supplied rectangle must be wholly contained within the
9018: (4)               original pixmap.
9019: (4)               Returns a pointer to the new pixmap. Throws exception on failure
9020: (4)               to allocate.
9021: (4)               """
9022: (4)               return _mupdf.ll_fz_new_pixmap_from_pixmap(pixmap, rect)
9023: (0)           def ll_fz_new_pixmap_with_bbox(colorspace, bbox, seps, alpha):
9024: (4)               r"""
9025: (4)               Low-level wrapper for `::fz_new_pixmap_with_bbox()`.
9026: (4)               Create a pixmap of a given size, location and pixel format.
9027: (4)               The bounding box specifies the size of the created pixmap and
9028: (4)               where it will be located. The colorspace determines the number
9029: (4)               of components per pixel. Alpha is always present. Pixmaps are
9030: (4)               reference counted, so drop references using fz_drop_pixmap.
9031: (4)               colorspace: Colorspace format used for the created pixmap. The
9032: (4)               pixmap will keep a reference to the colorspace.
9033: (4)               bbox: Bounding box specifying location/size of created pixmap.
9034: (4)               seps: Details of separations.
9035: (4)               alpha: 0 for no alpha, 1 for alpha.
9036: (4)               Returns a pointer to the new pixmap. Throws exception on failure
9037: (4)               to allocate.
9038: (4)               """
9039: (4)               return _mupdf.ll_fz_new_pixmap_with_bbox(colorspace, bbox, seps, alpha)
9040: (0)           def ll_fz_new_pixmap_with_bbox_and_data(colorspace, rect, seps, alpha,
samples):
9041: (4)               r"""
9042: (4)               Low-level wrapper for `::fz_new_pixmap_with_bbox_and_data()`.
9043: (4)               Create a pixmap of a given size, location and pixel format,
9044: (4)               using the supplied data block.
9045: (4)               The bounding box specifies the size of the created pixmap and
9046: (4)               where it will be located. The colorspace determines the number
9047: (4)               of components per pixel. Alpha is always present. Pixmaps are
9048: (4)               reference counted, so drop references using fz_drop_pixmap.
9049: (4)               colorspace: Colorspace format used for the created pixmap. The
9050: (4)               pixmap will keep a reference to the colorspace.
9051: (4)               rect: Bounding box specifying location/size of created pixmap.
9052: (4)               seps: Details of separations.
9053: (4)               alpha: Number of alpha planes (0 or 1).
9054: (4)               samples: The data block to keep the samples in.
9055: (4)               Returns a pointer to the new pixmap. Throws exception on failure
9056: (4)               to allocate.
9057: (4)               """
9058: (4)               return _mupdf.ll_fz_new_pixmap_with_bbox_and_data(colorspace, rect, seps,
alpha, samples)
9059: (0)           def ll_fz_new_pixmap_with_data(colorspace, w, h, seps, alpha, stride,
samples):
9060: (4)               r"""
9061: (4)               Low-level wrapper for `::fz_new_pixmap_with_data()`.
9062: (4)               Create a new pixmap, with its origin at
9063: (4)               (0,0) using the supplied data block.
9064: (4)               cs: The colorspace to use for the pixmap, or NULL for an alpha
9065: (4)               plane/mask.
9066: (4)               w: The width of the pixmap (in pixels)
9067: (4)               h: The height of the pixmap (in pixels)
9068: (4)               seps: Details of separations.
```

```
9069: (4)               alpha: 0 for no alpha, 1 for alpha.
9070: (4)               stride: The byte offset from the pixel data in a row to the
9071: (4)               pixel data in the next row.
9072: (4)               samples: The data block to keep the samples in.
9073: (4)               Returns a pointer to the new pixmap. Throws exception on failure to
9074: (4)               allocate.
9075: (4)               """
9076: (4)               return _mupdf.ll_fz_new_pixmap_with_data(colorspace, w, h, seps, alpha,
stride, samples)
9077: (0)           def ll_fz_new_pixmap_writer(path, options, default_path, n, save):
9078: (4)               r""" Low-level wrapper for `::fz_new_pixmap_writer()`."""
9079: (4)               return _mupdf.ll_fz_new_pixmap_writer(path, options, default_path, n,
save)
9080: (0)           def ll_fz_new_pkm_band_writer(out):
9081: (4)               r"""
9082: (4)               Low-level wrapper for `::fz_new_pkm_band_writer()`.
9083: (4)               Create a new pkm band writer for CMYK pixmaps.
9084: (4)               """
9085: (4)               return _mupdf.ll_fz_new_pkm_band_writer(out)
9086: (0)           def ll_fz_new_pkm_pixmap_writer(path, options):
9087: (4)               r""" Low-level wrapper for `::fz_new_pkm_pixmap_writer()`."""
9088: (4)               return _mupdf.ll_fz_new_pkm_pixmap_writer(path, options)
9089: (0)           def ll_fz_new_png_band_writer(out):
9090: (4)               r"""
9091: (4)               Low-level wrapper for `::fz_new_png_band_writer()`.
9092: (4)               Create a new png band writer (greyscale or RGB, with or without
9093: (4)               alpha).
9094: (4)               """
9095: (4)               return _mupdf.ll_fz_new_png_band_writer(out)
9096: (0)           def ll_fz_new_png_pixmap_writer(path, options):
9097: (4)               r""" Low-level wrapper for `::fz_new_png_pixmap_writer()`."""
9098: (4)               return _mupdf.ll_fz_new_png_pixmap_writer(path, options)
9099: (0)           def ll_fz_new_pnm_band_writer(out):
9100: (4)               r"""
9101: (4)               Low-level wrapper for `::fz_new_pnm_band_writer()`.
9102: (4)               Create a band writer targetting pnm (greyscale or rgb, no
9103: (4)               alpha).
9104: (4)               """
9105: (4)               return _mupdf.ll_fz_new_pnm_band_writer(out)
9106: (0)           def ll_fz_new_pnm_pixmap_writer(path, options):
9107: (4)               r""" Low-level wrapper for `::fz_new_pnm_pixmap_writer()`."""
9108: (4)               return _mupdf.ll_fz_new_pnm_pixmap_writer(path, options)
9109: (0)           def ll_fz_new_pool():
9110: (4)               r"""
9111: (4)               Low-level wrapper for `::fz_new_pool()`.
9112: (4)               Create a new pool to allocate from.
9113: (4)               """
9114: (4)               return _mupdf.ll_fz_new_pool()
9115: (0)           def ll_fz_new_ppm_pixmap_writer(path, options):
9116: (4)               r""" Low-level wrapper for `::fz_new_ppm_pixmap_writer()`."""
9117: (4)               return _mupdf.ll_fz_new_ppm_pixmap_writer(path, options)
9118: (0)           def ll_fz_new_ps_band_writer(out):
9119: (4)               r"""
9120: (4)               Low-level wrapper for `::fz_new_ps_band_writer()`.
9121: (4)               Create a postscript band writer for gray, rgb, or cmyk, no
9122: (4)               alpha.
9123: (4)               """
9124: (4)               return _mupdf.ll_fz_new_ps_band_writer(out)
9125: (0)           def ll_fz_new_ps_writer(path, options):
9126: (4)               r""" Low-level wrapper for `::fz_new_ps_writer()`."""
9127: (4)               return _mupdf.ll_fz_new_ps_writer(path, options)
9128: (0)           def ll_fz_new_ps_writer_with_output(out, options):
9129: (4)               r""" Low-level wrapper for `::fz_new_ps_writer_with_output()`."""
9130: (4)               return _mupdf.ll_fz_new_ps_writer_with_output(out, options)
9131: (0)           def ll_fz_new_psd_band_writer(out):
9132: (4)               r"""
9133: (4)               Low-level wrapper for `::fz_new_psd_band_writer()`.
9134: (4)               Open a PSD band writer.
9135: (4)               """
```

```
9136: (4)                return _mupdf.ll_fz_new_psd_band_writer(out)
9137: (0)            def ll_fz_new_pwg_band_writer(out, pwg):
9138: (4)                r"""
9139: (4)                Low-level wrapper for `::fz_new_pwg_band_writer()`.
9140: (4)                Create a new color pwg band writer.
9141: (4)                """
9142: (4)                return _mupdf.ll_fz_new_pwg_band_writer(out, pwg)
9143: (0)            def ll_fz_new_pwg_writer(path, options):
9144: (4)                r""" Low-level wrapper for `::fz_new_pwg_writer()`."""
9145: (4)                return _mupdf.ll_fz_new_pwg_writer(path, options)
9146: (0)            def ll_fz_new_pwg_writer_with_output(out, options):
9147: (4)                r""" Low-level wrapper for `::fz_new_pwg_writer_with_output()`."""
9148: (4)                return _mupdf.ll_fz_new_pwg_writer_with_output(out, options)
9149: (0)            def ll_fz_new_rle_output(chain):
9150: (4)                r""" Low-level wrapper for `::fz_new_rle_output()`."""
9151: (4)                return _mupdf.ll_fz_new_rle_output(chain)
9152: (0)            def ll_fz_new_separations(controllable):
9153: (4)                r"""
9154: (4)                Low-level wrapper for `::fz_new_separations()`.
9155: (4)                Create a new separations structure (initially empty)
9156: (4)                """
9157: (4)                return _mupdf.ll_fz_new_separations(controllable)
9158: (0)            def ll_fz_new_stext_device(page, options):
9159: (4)                r"""
9160: (4)                Low-level wrapper for `::fz_new_stext_device()`.
9161: (4)                Create a device to extract the text on a page.
9162: (4)                Gather the text on a page into blocks and lines.
9163: (4)                The reading order is taken from the order the text is drawn in
9164: (4)                the source file, so may not be accurate.
9165: (4)                page: The text page to which content should be added. This will
9166: (4)                usually be a newly created (empty) text page, but it can be one
9167: (4)                containing data already (for example when merging multiple
9168: (4)                pages, or watermarking).
9169: (4)                options: Options to configure the stext device.
9170: (4)                """
9171: (4)                return _mupdf.ll_fz_new_stext_device(page, options)
9172: (0)            def ll_fz_new_stext_page(mediabox):
9173: (4)                r"""
9174: (4)                Low-level wrapper for `::fz_new_stext_page()`.
9175: (4)                Create an empty text page.
9176: (4)                The text page is filled out by the text device to contain the
9177: (4)                blocks and lines of text on the page.
9178: (4)                mediabox: optional mediabox information.
9179: (4)                """
9180: (4)                return _mupdf.ll_fz_new_stext_page(mediabox)
9181: (0)            def ll_fz_new_stext_page_from_chapter_page_number(doc, chapter, number,
options):
9182: (4)                r""" Low-level wrapper for
`::fz_new_stext_page_from_chapter_page_number()`."""
9183: (4)                return _mupdf.ll_fz_new_stext_page_from_chapter_page_number(doc, chapter,
number, options)
9184: (0)            def ll_fz_new_stext_page_from_display_list(list, options):
9185: (4)                r""" Low-level wrapper for `::fz_new_stext_page_from_display_list()`."""
9186: (4)                return _mupdf.ll_fz_new_stext_page_from_display_list(list, options)
9187: (0)            def ll_fz_new_stext_page_from_page(page, options):
9188: (4)                r"""
9189: (4)                Low-level wrapper for `::fz_new_stext_page_from_page()`.
9190: (4)                Extract text from page.
9191: (4)                Ownership of the fz_stext_page is returned to the caller.
9192: (4)                """
9193: (4)                return _mupdf.ll_fz_new_stext_page_from_page(page, options)
9194: (0)            def ll_fz_new_stext_page_from_page_number(doc, number, options):
9195: (4)                r""" Low-level wrapper for `::fz_new_stext_page_from_page_number()`."""
9196: (4)                return _mupdf.ll_fz_new_stext_page_from_page_number(doc, number, options)
9197: (0)            def ll_fz_new_store_context(max):
9198: (4)                r"""
9199: (4)                Low-level wrapper for `::fz_new_store_context()`.
9200: (4)                Create a new store inside the context
9201: (4)                max: The maximum size (in bytes) that the store is allowed to
```

```
9202: (4)               grow to. FZ_STORE_UNLIMITED means no limit.
9203: (4)               """
9204: (4)               return _mupdf.ll_fz_new_store_context(max)
9205: (0)           def ll_fz_new_story(buf, user_css, em, dir):
9206: (4)               r""" Low-level wrapper for `::fz_new_story()`."""
9207: (4)               return _mupdf.ll_fz_new_story(buf, user_css, em, dir)
9208: (0)           def ll_fz_new_stream(state, next, drop):
9209: (4)               r"""
9210: (4)               Low-level wrapper for `::fz_new_stream()`.
9211: (4)               Create a new stream object with the given
9212: (4)               internal state and function pointers.
9213: (4)               state: Internal state (opaque to everything but implementation).
9214: (4)               next: Should provide the next set of bytes (up to max) of stream
9215: (4)               data. Return the number of bytes read, or EOF when there is no
9216: (4)               more data.
9217: (4)               drop: Should clean up and free the internal state. May not
9218: (4)               throw exceptions.
9219: (4)               """
9220: (4)               return _mupdf.ll_fz_new_stream(state, next, drop)
9221: (0)           def ll_fz_new_string(str):
9222: (4)               r""" Low-level wrapper for `::fz_new_string()`."""
9223: (4)               return _mupdf.ll_fz_new_string(str)
9224: (0)           def ll_fz_new_stroke_state():
9225: (4)               r"""
9226: (4)               Low-level wrapper for `::fz_new_stroke_state()`.
9227: (4)               Create a new (empty) stroke state structure (with no dash
9228: (4)               data) and return a reference to it.
9229: (4)               Throws exception on failure to allocate.
9230: (4)               """
9231: (4)               return _mupdf.ll_fz_new_stroke_state()
9232: (0)           def ll_fz_new_stroke_state_with_dash_len(len):
9233: (4)               r"""
9234: (4)               Low-level wrapper for `::fz_new_stroke_state_with_dash_len()`.
9235: (4)               Create a new (empty) stroke state structure, with room for
9236: (4)               dash data of the given length, and return a reference to it.
9237: (4)               len: The number of dash elements to allow room for.
9238: (4)               Throws exception on failure to allocate.
9239: (4)               """
9240: (4)               return _mupdf.ll_fz_new_stroke_state_with_dash_len(len)
9241: (0)           def ll_fz_new_svg_device(out, page_width, page_height, text_format,
reuse_images):
9242: (4)               r"""
9243: (4)               Low-level wrapper for `::fz_new_svg_device()`.
9244: (4)               Create a device that outputs (single page) SVG files to
9245: (4)               the given output stream.
9246: (4)               Equivalent to fz_new_svg_device_with_id passing id = NULL.
9247: (4)               """
9248: (4)               return _mupdf.ll_fz_new_svg_device(out, page_width, page_height,
text_format, reuse_images)
9249: (0)           def ll_fz_new_svg_device_with_id(out, page_width, page_height, text_format,
reuse_images, id):
9250: (4)               r"""
9251: (5)                Low-level wrapper for `::fz_new_svg_device_with_id()`.
9252: (4)               This function has out-params. Python/C# wrappers look like:
9253: (8)                   `ll_fz_new_svg_device_with_id(::fz_output *out, float page_width,
float page_height, int text_format, int reuse_images)` => `(fz_device *, int id)`
9254: (8)                   Create a device that outputs (single page) SVG files to
9255: (8)                   the given output stream.
9256: (8)                   output: The output stream to send the constructed SVG page to.
9257: (8)                   page_width, page_height: The page dimensions to use (in points).
9258: (8)                   text_format: How to emit text. One of the following values:
9259: (12)                      FZ_SVG_TEXT_AS_TEXT: As <text> elements with possible
9260: (12)                      layout errors and mismatching fonts.
9261: (12)                      FZ_SVG_TEXT_AS_PATH: As <path> elements with exact
9262: (12)                      visual appearance.
9263: (8)                   reuse_images: Share image resources using <symbol> definitions.
9264: (8)                   id: ID parameter to keep generated IDs unique across SVG files.
9265: (4)               """
9266: (4)               return _mupdf.ll_fz_new_svg_device_with_id(out, page_width, page_height,
```

```
text_format, reuse_images, id)
9267: (0)           def ll_fz_new_svg_writer(path, options):
9268: (4)               r""" Low-level wrapper for `::fz_new_svg_writer()`."""
9269: (4)               return _mupdf.ll_fz_new_svg_writer(path, options)
9270: (0)           def ll_fz_new_svg_writer_with_output(out, options):
9271: (4)               r""" Low-level wrapper for `::fz_new_svg_writer_with_output()`."""
9272: (4)               return _mupdf.ll_fz_new_svg_writer_with_output(out, options)
9273: (0)           def ll_fz_new_test_device(is_color, threshold, options, passthrough):
9274: (4)               r"""
9275: (5)                Low-level wrapper for `::fz_new_test_device()`.
9276: (4)               This function has out-params. Python/C# wrappers look like:
9277: (8)                   `ll_fz_new_test_device(float threshold, int options, ::fz_device
*passthrough)` => `(fz_device *, int is_color)`
9278: (8)                   Create a device to test for features.
9279: (8)                   Currently only tests for the presence of non-grayscale colors.
9280: (8)                   is_color: Possible values returned:
9281: (12)                      0: Definitely greyscale
9282: (12)                      1: Probably color (all colors were grey, but there
9283: (12)                      were images or shadings in a non grey colorspace).
9284: (12)                      2: Definitely color
9285: (8)                   threshold: The difference from grayscale that will be tolerated.
9286: (8)                   Typical values to use are either 0 (be exact) and 0.02 (allow an
9287: (8)                   imperceptible amount of slop).
9288: (8)                   options: A set of bitfield options, from the FZ_TEST_OPT set.
9289: (8)                   passthrough: A device to pass all calls through to, or NULL.
9290: (8)                   If set, then the test device can both test and pass through to
9291: (8)                   an underlying device (like, say, the display list device). This
9292: (8)                   means that a display list can be created and at the end we'll
9293: (8)                   know if it's colored or not.
9294: (8)                   In the absence of a passthrough device, the device will throw
9295: (8)                   an exception to stop page interpretation when color is found.
9296: (4)               """
9297: (4)               return _mupdf.ll_fz_new_test_device(is_color, threshold, options,
passthrough)
9298: (0)           def ll_fz_new_text():
9299: (4)               r"""
9300: (4)               Low-level wrapper for `::fz_new_text()`.
9301: (4)               Create a new empty fz_text object.
9302: (4)               Throws exception on failure to allocate.
9303: (4)               """
9304: (4)               return _mupdf.ll_fz_new_text()
9305: (0)           def ll_fz_new_text_writer(format, path, options):
9306: (4)               r""" Low-level wrapper for `::fz_new_text_writer()`."""
9307: (4)               return _mupdf.ll_fz_new_text_writer(format, path, options)
9308: (0)           def ll_fz_new_text_writer_with_output(format, out, options):
9309: (4)               r""" Low-level wrapper for `::fz_new_text_writer_with_output()`."""
9310: (4)               return _mupdf.ll_fz_new_text_writer_with_output(format, out, options)
9311: (0)           def ll_fz_new_trace_device(out):
9312: (4)               r"""
9313: (4)               Low-level wrapper for `::fz_new_trace_device()`.
9314: (4)               Create a device to print a debug trace of all device calls.
9315: (4)               """
9316: (4)               return _mupdf.ll_fz_new_trace_device(out)
9317: (0)           def ll_fz_new_tree_archive(tree):
9318: (4)               r"""
9319: (4)               Low-level wrapper for `::fz_new_tree_archive()`.
9320: (4)               Create an archive that holds named buffers.
9321: (4)               tree can either be a preformed tree with fz_buffers as values,
9322: (4)               or it can be NULL for an empty tree.
9323: (4)               """
9324: (4)               return _mupdf.ll_fz_new_tree_archive(tree)
9325: (0)           def ll_fz_new_type3_font(name, matrix):
9326: (4)               r"""
9327: (4)               Low-level wrapper for `::fz_new_type3_font()`.
9328: (4)               Create a new (empty) type3 font.
9329: (4)               name: Name of font (or NULL).
9330: (4)               matrix: Font matrix.
9331: (4)               Returns a new font handle, or throws exception on
9332: (4)               allocation failure.
```

```
9333: (4)                """
9334: (4)                return _mupdf.ll_fz_new_type3_font(name, matrix)
9335: (0)            def ll_fz_new_xhtml_document_from_document(old_doc, opts):
9336: (4)                r"""
9337: (4)                Low-level wrapper for `::fz_new_xhtml_document_from_document()`.
9338: (4)                Use text extraction to convert the input document into XHTML,
9339: (4)                then open the result as a new document that can be reflowed.
9340: (4)                """
9341: (4)                return _mupdf.ll_fz_new_xhtml_document_from_document(old_doc, opts)
9342: (0)            def ll_fz_new_xmltext_device(out):
9343: (4)                r"""
9344: (4)                Low-level wrapper for `::fz_new_xmltext_device()`.
9345: (4)                Create a device to output raw information.
9346: (4)                """
9347: (4)                return _mupdf.ll_fz_new_xmltext_device(out)
9348: (0)            def ll_fz_new_zip_writer(filename):
9349: (4)                r"""
9350: (4)                Low-level wrapper for `::fz_new_zip_writer()`.
9351: (4)                Create a new zip writer that writes to a given file.
9352: (4)                Open an archive using a seekable stream object rather than
9353: (4)                opening a file or directory on disk.
9354: (4)                """
9355: (4)                return _mupdf.ll_fz_new_zip_writer(filename)
9356: (0)            def ll_fz_new_zip_writer_with_output(out):
9357: (4)                r"""
9358: (4)                Low-level wrapper for `::fz_new_zip_writer_with_output()`.
9359: (4)                Create a new zip writer that writes to a given output stream.
9360: (4)                Ownership of out passes in immediately upon calling this function.
9361: (4)                The caller should never drop the fz_output, even if this function throws
9362: (4)                an exception.
9363: (4)                """
9364: (4)                return _mupdf.ll_fz_new_zip_writer_with_output(out)
9365: (0)            def ll_fz_next_page(doc, loc):
9366: (4)                r"""
9367: (4)                Low-level wrapper for `::fz_next_page()`.
9368: (4)                Function to get the location of the next page (allowing for the
9369: (4)                end of chapters etc). If at the end of the document, returns the
9370: (4)                current location.
9371: (4)                """
9372: (4)                return _mupdf.ll_fz_next_page(doc, loc)
9373: (0)            def ll_fz_normalize_vector(p):
9374: (4)                r"""
9375: (4)                Low-level wrapper for `::fz_normalize_vector()`.
9376: (4)                Normalize a vector to length one.
9377: (4)                """
9378: (4)                return _mupdf.ll_fz_normalize_vector(p)
9379: (0)            def ll_fz_open_a85d(chain):
9380: (4)                r"""
9381: (4)                Low-level wrapper for `::fz_open_a85d()`.
9382: (4)                a85d filter performs ASCII 85 Decoding of data read
9383: (4)                from the chained filter.
9384: (4)                """
9385: (4)                return _mupdf.ll_fz_open_a85d(chain)
9386: (0)            def ll_fz_open_accelerated_document(filename, accel):
9387: (4)                r"""
9388: (4)                Low-level wrapper for `::fz_open_accelerated_document()`.
9389: (4)                Open a document file and read its basic structure so pages and
9390: (4)                objects can be located. MuPDF will try to repair broken
9391: (4)                documents (without actually changing the file contents).
9392: (4)                The returned fz_document is used when calling most other
9393: (4)                document related functions.
9394: (4)                filename: a path to a file as it would be given to open(2).
9395: (4)                """
9396: (4)                return _mupdf.ll_fz_open_accelerated_document(filename, accel)
9397: (0)            def ll_fz_open_accelerated_document_with_stream(magic, stream, accel):
9398: (4)                r"""
9399: (4)                Low-level wrapper for `::fz_open_accelerated_document_with_stream()`.
9400: (4)                Open a document using the specified stream object rather than
9401: (4)                opening a file on disk.
```

```
9402: (4)                magic: a string used to detect document type; either a file name
9403: (4)                or mime-type.
9404: (4)                stream: a stream of the document contents.
9405: (4)                accel: NULL, or a stream of the 'accelerator' contents for this document.
9406: (4)                NOTE: The caller retains ownership of 'stream' and 'accel' - the document
will
9407: (4)                take its own references if required.
9408: (4)                """
9409: (4)                return _mupdf.ll_fz_open_accelerated_document_with_stream(magic, stream,
accel)
9410: (0)            def ll_fz_open_accelerated_document_with_stream_and_dir(magic, stream, accel,
dir):
9411: (4)                r"""
9412: (4)                Low-level wrapper for
`::fz_open_accelerated_document_with_stream_and_dir()`.
9413: (4)                Open a document using the specified stream object rather than
9414: (4)                opening a file on disk.
9415: (4)                magic: a string used to detect document type; either a file name
9416: (4)                or mime-type.
9417: (4)                stream: a stream of the document contents.
9418: (4)                accel: NULL, or a stream of the 'accelerator' contents for this document.
9419: (4)                dir: NULL, or the 'directory context' for the stream contents.
9420: (4)                NOTE: The caller retains ownership of 'stream', 'accel' and 'dir' - the
document will
9421: (4)                take its own references if required.
9422: (4)                """
9423: (4)                return _mupdf.ll_fz_open_accelerated_document_with_stream_and_dir(magic,
stream, accel, dir)
9424: (0)            def ll_fz_open_aesd(chain, key, keylen):
9425: (4)                r"""
9426: (4)                Low-level wrapper for `::fz_open_aesd()`.
9427: (4)                aesd filter performs AES decoding of data read from the chained
9428: (4)                filter using the supplied key.
9429: (4)                """
9430: (4)                return _mupdf.ll_fz_open_aesd(chain, key, keylen)
9431: (0)            def ll_fz_open_ahxd(chain):
9432: (4)                r"""
9433: (4)                Low-level wrapper for `::fz_open_ahxd()`.
9434: (4)                ahxd filter performs ASCII Hex decoding of data read
9435: (4)                from the chained filter.
9436: (4)                """
9437: (4)                return _mupdf.ll_fz_open_ahxd(chain)
9438: (0)            def ll_fz_open_arc4(chain, key, keylen):
9439: (4)                r"""
9440: (4)                Low-level wrapper for `::fz_open_arc4()`.
9441: (4)                arc4 filter performs RC4 decoding of data read from the chained
9442: (4)                filter using the supplied key.
9443: (4)                """
9444: (4)                return _mupdf.ll_fz_open_arc4(chain, key, keylen)
9445: (0)            def ll_fz_open_archive(filename):
9446: (4)                r"""
9447: (4)                Low-level wrapper for `::fz_open_archive()`.
9448: (4)                Open a zip or tar archive
9449: (4)                Open a file and identify its archive type based on the archive
9450: (4)                signature contained inside.
9451: (4)                filename: a path to a file as it would be given to open(2).
9452: (4)                """
9453: (4)                return _mupdf.ll_fz_open_archive(filename)
9454: (0)            def ll_fz_open_archive_entry(arch, name):
9455: (4)                r"""
9456: (4)                Low-level wrapper for `::fz_open_archive_entry()`.
9457: (4)                Opens an archive entry as a stream.
9458: (4)                name: Entry name to look for, this must be an exact match to
9459: (4)                the entry name in the archive.
9460: (4)                Throws an exception if a matching entry cannot be found.
9461: (4)                """
9462: (4)                return _mupdf.ll_fz_open_archive_entry(arch, name)
9463: (0)            def ll_fz_open_archive_with_stream(file):
9464: (4)                r"""
```

```
9465: (4)               Low-level wrapper for `::fz_open_archive_with_stream()`.
9466: (4)               Open zip or tar archive stream.
9467: (4)               Open an archive using a seekable stream object rather than
9468: (4)               opening a file or directory on disk.
9469: (4)               """
9470: (4)               return _mupdf.ll_fz_open_archive_with_stream(file)
9471: (0)           def ll_fz_open_buffer(buf):
9472: (4)               r"""
9473: (4)               Low-level wrapper for `::fz_open_buffer()`.
9474: (4)               Open a buffer as a stream.
9475: (4)               buf: The buffer to open. Ownership of the buffer is NOT passed
9476: (4)               in (this function takes its own reference).
9477: (4)               Returns pointer to newly created stream. May throw exceptions on
9478: (4)               failure to allocate.
9479: (4)               """
9480: (4)               return _mupdf.ll_fz_open_buffer(buf)
9481: (0)           def ll_fz_open_cfb_archive(filename):
9482: (4)               r"""
9483: (4)               Low-level wrapper for `::fz_open_cfb_archive()`.
9484: (4)               Open a cfb file as an archive.
9485: (4)               An exception is thrown if the file is not recognised as a cfb.
9486: (4)               filename: a path to an archive file as it would be given to
9487: (4)               open(2).
9488: (4)               """
9489: (4)               return _mupdf.ll_fz_open_cfb_archive(filename)
9490: (0)           def ll_fz_open_cfb_archive_with_stream(file):
9491: (4)               r"""
9492: (4)               Low-level wrapper for `::fz_open_cfb_archive_with_stream()`.
9493: (4)               Open a cfb file as an archive.
9494: (4)               Open an archive using a seekable stream object rather than
9495: (4)               opening a file or directory on disk.
9496: (4)               An exception is thrown if the file is not recognised as a chm.
9497: (4)               """
9498: (4)               return _mupdf.ll_fz_open_cfb_archive_with_stream(file)
9499: (0)           def ll_fz_open_compressed_buffer(arg_1):
9500: (4)               r"""
9501: (4)               Low-level wrapper for `::fz_open_compressed_buffer()`.
9502: (4)               Open a stream to read the decompressed version of a buffer.
9503: (4)               """
9504: (4)               return _mupdf.ll_fz_open_compressed_buffer(arg_1)
9505: (0)           def ll_fz_open_concat(max, pad):
9506: (4)               r"""
9507: (4)               Low-level wrapper for `::fz_open_concat()`.
9508: (4)               Concat filter concatenates several streams into one.
9509: (4)               """
9510: (4)               return _mupdf.ll_fz_open_concat(max, pad)
9511: (0)           def ll_fz_open_dctd(chain, color_transform, invert_cmyk, l2factor,
jpegtables):
9512: (4)               r"""
9513: (4)               Low-level wrapper for `::fz_open_dctd()`.
9514: (4)               dctd filter performs DCT (JPEG) decoding of data read
9515: (4)               from the chained filter.
9516: (4)               color_transform implements the PDF color_transform option
9517: (8)                   use -1 for default behavior
9518: (8)                   use 0 to disable YUV-RGB / YCCK-CMYK transforms
9519: (8)                   use 1 to enable YUV-RGB / YCCK-CMYK transforms
9520: (4)               invert_cmyk implements the necessary inversion for Photoshop CMYK images
9521: (8)                   use 0 if embedded in PDF
9522: (8)                   use 1 if not embedded in PDF
9523: (4)               For subsampling on decode, set l2factor to the log2 of the
9524: (4)               reduction required (therefore 0 = full size decode).
9525: (4)               jpegtables is an optional stream from which the JPEG tables
9526: (4)               can be read. Use NULL if not required.
9527: (4)               """
9528: (4)               return _mupdf.ll_fz_open_dctd(chain, color_transform, invert_cmyk,
l2factor, jpegtables)
9529: (0)           def ll_fz_open_directory(path):
9530: (4)               r"""
9531: (4)               Low-level wrapper for `::fz_open_directory()`.
```

```
9532: (4)               Open a directory as if it was an archive.
9533: (4)               A special case where a directory is opened as if it was an
9534: (4)               archive.
9535: (4)               Note that for directories it is not possible to retrieve the
9536: (4)               number of entries or list the entries. It is however possible
9537: (4)               to check if the archive has a particular entry.
9538: (4)               path: a path to a directory as it would be given to opendir(3).
9539: (4)               """
9540: (4)               return _mupdf.ll_fz_open_directory(path)
9541: (0)           def ll_fz_open_document(filename):
9542: (4)               r"""
9543: (4)               Low-level wrapper for `::fz_open_document()`.
9544: (4)               Open a document file and read its basic structure so pages and
9545: (4)               objects can be located. MuPDF will try to repair broken
9546: (4)               documents (without actually changing the file contents).
9547: (4)               The returned fz_document is used when calling most other
9548: (4)               document related functions.
9549: (4)               filename: a path to a file as it would be given to open(2).
9550: (4)               """
9551: (4)               return _mupdf.ll_fz_open_document(filename)
9552: (0)           def ll_fz_open_document_with_buffer(magic, buffer):
9553: (4)               r"""
9554: (4)               Low-level wrapper for `::fz_open_document_with_buffer()`.
9555: (4)               Open a document using a buffer rather than opening a file on disk.
9556: (4)               """
9557: (4)               return _mupdf.ll_fz_open_document_with_buffer(magic, buffer)
9558: (0)           def ll_fz_open_document_with_stream(magic, stream):
9559: (4)               r"""
9560: (4)               Low-level wrapper for `::fz_open_document_with_stream()`.
9561: (4)               Open a document using the specified stream object rather than
9562: (4)               opening a file on disk.
9563: (4)               magic: a string used to detect document type; either a file name
9564: (4)               or mime-type.
9565: (4)               stream: a stream representing the contents of the document file.
9566: (4)               NOTE: The caller retains ownership of 'stream' - the document will take
its
9567: (4)               own reference if required.
9568: (4)               """
9569: (4)               return _mupdf.ll_fz_open_document_with_stream(magic, stream)
9570: (0)           def ll_fz_open_document_with_stream_and_dir(magic, stream, dir):
9571: (4)               r"""
9572: (4)               Low-level wrapper for `::fz_open_document_with_stream_and_dir()`.
9573: (4)               Open a document using the specified stream object rather than
9574: (4)               opening a file on disk.
9575: (4)               magic: a string used to detect document type; either a file name
9576: (4)               or mime-type.
9577: (4)               stream: a stream representing the contents of the document file.
9578: (4)               dir: a 'directory context' for those filetypes that need it.
9579: (4)               NOTE: The caller retains ownership of 'stream' and 'dir' - the document
will
9580: (4)               take its own references if required.
9581: (4)               """
9582: (4)               return _mupdf.ll_fz_open_document_with_stream_and_dir(magic, stream, dir)
9583: (0)           def ll_fz_open_endstream_filter(chain, len, offset):
9584: (4)               r"""
9585: (4)               Low-level wrapper for `::fz_open_endstream_filter()`.
9586: (4)               The endstream filter reads a PDF substream, and starts to look
9587: (4)               for an 'endstream' token after the specified length.
9588: (4)               """
9589: (4)               return _mupdf.ll_fz_open_endstream_filter(chain, len, offset)
9590: (0)           def ll_fz_open_faxd(chain, k, end_of_line, encoded_byte_align, columns, rows,
end_of_block, black_is_1):
9591: (4)               r"""
9592: (4)               Low-level wrapper for `::fz_open_faxd()`.
9593: (4)               faxd filter performs FAX decoding of data read from
9594: (4)               the chained filter.
9595: (4)               k: see fax specification (fax default is 0).
9596: (4)               end_of_line: whether we expect end of line markers (fax default
9597: (4)               is 0).
```

```
9598: (4)               encoded_byte_align: whether we align to bytes after each line
9599: (4)               (fax default is 0).
9600: (4)               columns: how many columns in the image (fax default is 1728).
9601: (4)               rows: 0 for unspecified or the number of rows of data to expect.
9602: (4)               end_of_block: whether we expect end of block markers (fax
9603: (4)               default is 1).
9604: (4)               black_is_1: determines the polarity of the image (fax default is
9605: (4)               0).
9606: (4)               """
9607: (4)               return _mupdf.ll_fz_open_faxd(chain, k, end_of_line, encoded_byte_align,
columns, rows, end_of_block, black_is_1)
9608: (0)           def ll_fz_open_file(filename):
9609: (4)               r"""
9610: (4)               Low-level wrapper for `::fz_open_file()`.
9611: (4)               Open the named file and wrap it in a stream.
9612: (4)               filename: Path to a file. On non-Windows machines the filename
9613: (4)               should be exactly as it would be passed to fopen(2). On Windows
9614: (4)               machines, the path should be UTF-8 encoded so that non-ASCII
9615: (4)               characters can be represented. Other platforms do the encoding
9616: (4)               as standard anyway (and in most cases, particularly for MacOS
9617: (4)               and Linux, the encoding they use is UTF-8 anyway).
9618: (4)               """
9619: (4)               return _mupdf.ll_fz_open_file(filename)
9620: (0)           def ll_fz_open_file_autodelete(filename):
9621: (4)               r"""
9622: (4)               Low-level wrapper for `::fz_open_file_autodelete()`.
9623: (4)               Do the same as fz_open_file, but delete the file upon close.
9624: (4)               """
9625: (4)               return _mupdf.ll_fz_open_file_autodelete(filename)
9626: (0)           def ll_fz_open_file_ptr_no_close(file):
9627: (4)               r"""
9628: (4)               Low-level wrapper for `::fz_open_file_ptr_no_close()`.
9629: (4)               Create a stream from a FILE * that will not be closed
9630: (4)               when the stream is dropped.
9631: (4)               """
9632: (4)               return _mupdf.ll_fz_open_file_ptr_no_close(file)
9633: (0)           def ll_fz_open_flated(chain, window_bits):
9634: (4)               r"""
9635: (4)               Low-level wrapper for `::fz_open_flated()`.
9636: (4)               flated filter performs LZ77 decoding (inflating) of data read
9637: (4)               from the chained filter.
9638: (4)               window_bits: How large a decompression window to use. Typically
9639: (4)               15. A negative number, -n, means to use n bits, but to expect
9640: (4)               raw data with no header.
9641: (4)               """
9642: (4)               return _mupdf.ll_fz_open_flated(chain, window_bits)
9643: (0)           def ll_fz_open_image_decomp_stream(arg_1, arg_2, l2factor):
9644: (4)               r"""
9645: (5)                Low-level wrapper for `::fz_open_image_decomp_stream()`.
9646: (4)               This function has out-params. Python/C# wrappers look like:
9647: (8)                   `ll_fz_open_image_decomp_stream(::fz_stream *arg_0,
::fz_compression_params *arg_1)` => `(fz_stream *, int l2factor)`
9648: (8)                   Open a stream to read the decompressed version of another stream
9649: (8)                   with optional log2 subsampling.
9650: (4)               """
9651: (4)               return _mupdf.ll_fz_open_image_decomp_stream(arg_1, arg_2, l2factor)
9652: (0)           def ll_fz_open_image_decomp_stream_from_buffer(arg_1, l2factor):
9653: (4)               r"""
9654: (5)                Low-level wrapper for `::fz_open_image_decomp_stream_from_buffer()`.
9655: (4)               This function has out-params. Python/C# wrappers look like:
9656: (8)                   `ll_fz_open_image_decomp_stream_from_buffer(::fz_compressed_buffer
*arg_0)` => `(fz_stream *, int l2factor)`
9657: (8)                   Open a stream to read the decompressed version of a buffer,
9658: (8)                   with optional log2 subsampling.
9659: (8)                   l2factor = NULL for no subsampling, or a pointer to an integer
9660: (8)                   containing the maximum log2 subsample factor acceptable (0 =
9661: (8)                   none, 1 = halve dimensions, 2 = quarter dimensions etc). If
9662: (8)                   non-NULL, then *l2factor will be updated on exit with the actual
9663: (8)                   log2 subsample factor achieved.
```

```
9664: (4)                """
9665: (4)                return _mupdf.ll_fz_open_image_decomp_stream_from_buffer(arg_1, l2factor)
9666: (0)            def ll_fz_open_jbig2d(chain, globals, embedded):
9667: (4)                r"""
9668: (4)                Low-level wrapper for `::fz_open_jbig2d()`.
9669: (4)                Open a filter that performs jbig2 decompression on the chained
9670: (4)                stream, using the optional globals record.
9671: (4)                """
9672: (4)                return _mupdf.ll_fz_open_jbig2d(chain, globals, embedded)
9673: (0)            def ll_fz_open_leecher(chain, buf):
9674: (4)                r"""
9675: (4)                Low-level wrapper for `::fz_open_leecher()`.
9676: (4)                Attach a filter to a stream that will store any
9677: (4)                characters read from the stream into the supplied buffer.
9678: (4)                chain: The underlying stream to leech from.
9679: (4)                buf: The buffer into which the read data should be appended.
9680: (4)                The buffer will be resized as required.
9681: (4)                Returns pointer to newly created stream. May throw exceptions on
9682: (4)                failure to allocate.
9683: (4)                """
9684: (4)                return _mupdf.ll_fz_open_leecher(chain, buf)
9685: (0)            def ll_fz_open_libarchive_archive(filename):
9686: (4)                r"""
9687: (4)                Low-level wrapper for `::fz_open_libarchive_archive()`.
9688: (4)                Open an archive using libarchive.
9689: (4)                An exception is thrown if the file is not supported by libarchive.
9690: (4)                filename: a path to an archive file as it would be given to
9691: (4)                open(2).
9692: (4)                """
9693: (4)                return _mupdf.ll_fz_open_libarchive_archive(filename)
9694: (0)            def ll_fz_open_libarchive_archive_with_stream(file):
9695: (4)                r"""
9696: (4)                Low-level wrapper for `::fz_open_libarchive_archive_with_stream()`.
9697: (4)                Open an archive using libarchive.
9698: (4)                Open an archive using a seekable stream object rather than
9699: (4)                opening a file or directory on disk.
9700: (4)                An exception is thrown if the stream is not supported by libarchive.
9701: (4)                """
9702: (4)                return _mupdf.ll_fz_open_libarchive_archive_with_stream(file)
9703: (0)            def ll_fz_open_libarchived(chain):
9704: (4)                r"""
9705: (4)                Low-level wrapper for `::fz_open_libarchived()`.
9706: (4)                libarchived filter performs generic compressed decoding of data
9707: (4)                in any format understood by libarchive from the chained filter.
9708: (4)                This will throw an exception if libarchive is not built in, or
9709: (4)                if the compression format is not recognised.
9710: (4)                """
9711: (4)                return _mupdf.ll_fz_open_libarchived(chain)
9712: (0)            def ll_fz_open_lzwd(chain, early_change, min_bits, reverse_bits, old_tiff):
9713: (4)                r"""
9714: (4)                Low-level wrapper for `::fz_open_lzwd()`.
9715: (4)                lzwd filter performs LZW decoding of data read from the chained
9716: (4)                filter.
9717: (4)                early_change: (Default 1) specifies whether to change codes 1
9718: (4)                bit early.
9719: (4)                min_bits: (Default 9) specifies the minimum number of bits to
9720: (4)                use.
9721: (4)                reverse_bits: (Default 0) allows for compatibility with gif and
9722: (4)                old style tiffs (1).
9723: (4)                old_tiff: (Default 0) allows for different handling of the clear
9724: (4)                code, as found in old style tiffs.
9725: (4)                """
9726: (4)                return _mupdf.ll_fz_open_lzwd(chain, early_change, min_bits, reverse_bits,
old_tiff)
9727: (0)            def ll_fz_open_memory(data, len):
9728: (4)                r"""
9729: (4)                Low-level wrapper for `::fz_open_memory()`.
9730: (4)                Open a block of memory as a stream.
9731: (4)                data: Pointer to start of data block. Ownership of the data
```

```
9732: (4)                block is NOT passed in.
9733: (4)                len: Number of bytes in data block.
9734: (4)                Returns pointer to newly created stream. May throw exceptions on
9735: (4)                failure to allocate.
9736: (4)                """
9737: (4)                return _mupdf.ll_fz_open_memory(data, len)
9738: (0)            def ll_fz_open_null_filter(chain, len, offset):
9739: (4)                r"""
9740: (4)                Low-level wrapper for `::fz_open_null_filter()`.
9741: (4)                The null filter reads a specified amount of data from the
9742: (4)                substream.
9743: (4)                """
9744: (4)                return _mupdf.ll_fz_open_null_filter(chain, len, offset)
9745: (0)            def ll_fz_open_predict(chain, predictor, columns, colors, bpc):
9746: (4)                r"""
9747: (4)                Low-level wrapper for `::fz_open_predict()`.
9748: (4)                predict filter performs pixel prediction on data read from
9749: (4)                the chained filter.
9750: (4)                predictor: 1 = copy, 2 = tiff, other = inline PNG predictor
9751: (4)                columns: width of image in pixels
9752: (4)                colors: number of components.
9753: (4)                bpc: bits per component (typically 8)
9754: (4)                """
9755: (4)                return _mupdf.ll_fz_open_predict(chain, predictor, columns, colors, bpc)
9756: (0)            def ll_fz_open_range_filter(chain, ranges, nranges):
9757: (4)                r"""
9758: (4)                Low-level wrapper for `::fz_open_range_filter()`.
9759: (4)                The range filter copies data from specified ranges of the
9760: (4)                chained stream.
9761: (4)                """
9762: (4)                return _mupdf.ll_fz_open_range_filter(chain, ranges, nranges)
9763: (0)            def ll_fz_open_reflowed_document(underdoc, opts):
9764: (4)                r""" Low-level wrapper for `::fz_open_reflowed_document()`."""
9765: (4)                return _mupdf.ll_fz_open_reflowed_document(underdoc, opts)
9766: (0)            def ll_fz_open_rld(chain):
9767: (4)                r"""
9768: (4)                Low-level wrapper for `::fz_open_rld()`.
9769: (4)                rld filter performs Run Length Decoding of data read
9770: (4)                from the chained filter.
9771: (4)                """
9772: (4)                return _mupdf.ll_fz_open_rld(chain)
9773: (0)            def ll_fz_open_sgilog16(chain, w):
9774: (4)                r"""
9775: (4)                Low-level wrapper for `::fz_open_sgilog16()`.
9776: (4)                SGI Log 16bit (greyscale) decode from the chained filter.
9777: (4)                Decodes lines of w pixels to 8bpp greyscale.
9778: (4)                """
9779: (4)                return _mupdf.ll_fz_open_sgilog16(chain, w)
9780: (0)            def ll_fz_open_sgilog24(chain, w):
9781: (4)                r"""
9782: (4)                Low-level wrapper for `::fz_open_sgilog24()`.
9783: (4)                SGI Log 24bit (LUV) decode from the chained filter.
9784: (4)                Decodes lines of w pixels to 8bpc rgb.
9785: (4)                """
9786: (4)                return _mupdf.ll_fz_open_sgilog24(chain, w)
9787: (0)            def ll_fz_open_sgilog32(chain, w):
9788: (4)                r"""
9789: (4)                Low-level wrapper for `::fz_open_sgilog32()`.
9790: (4)                SGI Log 32bit (LUV) decode from the chained filter.
9791: (4)                Decodes lines of w pixels to 8bpc rgb.
9792: (4)                """
9793: (4)                return _mupdf.ll_fz_open_sgilog32(chain, w)
9794: (0)            def ll_fz_open_tar_archive(filename):
9795: (4)                r"""
9796: (4)                Low-level wrapper for `::fz_open_tar_archive()`.
9797: (4)                Open a tar archive file.
9798: (4)                An exception is thrown if the file is not a tar archive as
9799: (4)                indicated by the presence of a tar signature.
9800: (4)                filename: a path to a tar archive file as it would be given to
```

```
9801: (4)                open(2).
9802: (4)                """
9803: (4)                return _mupdf.ll_fz_open_tar_archive(filename)
9804: (0)            def ll_fz_open_tar_archive_with_stream(file):
9805: (4)                r"""
9806: (4)                Low-level wrapper for `::fz_open_tar_archive_with_stream()`.
9807: (4)                Open a tar archive stream.
9808: (4)                Open an archive using a seekable stream object rather than
9809: (4)                opening a file or directory on disk.
9810: (4)                An exception is thrown if the stream is not a tar archive as
9811: (4)                indicated by the presence of a tar signature.
9812: (4)                """
9813: (4)                return _mupdf.ll_fz_open_tar_archive_with_stream(file)
9814: (0)            def ll_fz_open_thunder(chain, w):
9815: (4)                r"""
9816: (4)                Low-level wrapper for `::fz_open_thunder()`.
9817: (4)                4bit greyscale Thunderscan decoding from the chained filter.
9818: (4)                Decodes lines of w pixels to 8bpp greyscale.
9819: (4)                """
9820: (4)                return _mupdf.ll_fz_open_thunder(chain, w)
9821: (0)            def ll_fz_open_zip_archive(path):
9822: (4)                r"""
9823: (4)                Low-level wrapper for `::fz_open_zip_archive()`.
9824: (4)                Open a zip archive file.
9825: (4)                An exception is thrown if the file is not a zip archive as
9826: (4)                indicated by the presence of a zip signature.
9827: (4)                filename: a path to a zip archive file as it would be given to
9828: (4)                open(2).
9829: (4)                """
9830: (4)                return _mupdf.ll_fz_open_zip_archive(path)
9831: (0)            def ll_fz_open_zip_archive_with_stream(file):
9832: (4)                r"""
9833: (4)                Low-level wrapper for `::fz_open_zip_archive_with_stream()`.
9834: (4)                Open a zip archive stream.
9835: (4)                Open an archive using a seekable stream object rather than
9836: (4)                opening a file or directory on disk.
9837: (4)                An exception is thrown if the stream is not a zip archive as
9838: (4)                indicated by the presence of a zip signature.
9839: (4)                """
9840: (4)                return _mupdf.ll_fz_open_zip_archive_with_stream(file)
9841: (0)            def ll_fz_opt_from_list(opt, optlist):
9842: (4)                r"""
9843: (4)                Low-level wrapper for `::fz_opt_from_list()`.
9844: (4)                Return the index of a (case-insensitive) option within an optlist.
9845: (4)                For instance for optlist = "Foo|Bar|Baz", and  opt = "bar",
9846: (4)                this would return 1.
9847: (4)                If the optlist ends with "|*" then that is a catch all case and
9848: (4)                matches all options allowing the caller to process it itself.
9849: (4)                fz_optarg will be set to point to the option, and the return
9850: (4)                value will be the index of the '*' option within that list.
9851: (4)                If an optlist entry ends with ':' (e.g. "Foo:") then that option
9852: (4)                may have suboptions appended to it (for example "JPG:80") and
9853: (4)                fz_optarg will be set to point at "80". Otherwise fz_optarg will
9854: (4)                be set to NULL.
9855: (4)                In the event of no-match found, prints an error and returns -1.
9856: (4)                """
9857: (4)                return _mupdf.ll_fz_opt_from_list(opt, optlist)
9858: (0)            def ll_fz_option_eq(a, b):
9859: (4)                r"""
9860: (4)                Low-level wrapper for `::fz_option_eq()`.
9861: (4)                Check to see if an option, a, from a string matches a reference
9862: (4)                option, b.
9863: (4)                (i.e. a could be 'foo' or 'foo,bar...' etc, but b can only be
9864: (4)                'foo'.)
9865: (4)                """
9866: (4)                return _mupdf.ll_fz_option_eq(a, b)
9867: (0)            def ll_fz_outline_glyph(font, gid, ctm):
9868: (4)                r"""
9869: (4)                Low-level wrapper for `::fz_outline_glyph()`.
```

```
9870: (4)                Look a glyph up from a font, and return the outline of the
9871: (4)                glyph using the given transform.
9872: (4)                The caller owns the returned path, and so is responsible for
9873: (4)                ensuring that it eventually gets dropped.
9874: (4)                """
9875: (4)                return _mupdf.ll_fz_outline_glyph(font, gid, ctm)
9876: (0)            def ll_fz_outline_iterator_delete(iter):
9877: (4)                r"""
9878: (4)                Low-level wrapper for `::fz_outline_iterator_delete()`.
9879: (4)                Delete the current item.
9880: (4)                This implicitly moves us to the 'next' item, and the return code is as for
fz_outline_iterator_next.
9881: (4)                """
9882: (4)                return _mupdf.ll_fz_outline_iterator_delete(iter)
9883: (0)            def ll_fz_outline_iterator_down(iter):
9884: (4)                r""" Low-level wrapper for `::fz_outline_iterator_down()`."""
9885: (4)                return _mupdf.ll_fz_outline_iterator_down(iter)
9886: (0)            def ll_fz_outline_iterator_from_outline(outline):
9887: (4)                r""" Low-level wrapper for `::fz_outline_iterator_from_outline()`."""
9888: (4)                return _mupdf.ll_fz_outline_iterator_from_outline(outline)
9889: (0)            def ll_fz_outline_iterator_insert(iter, item):
9890: (4)                r"""
9891: (4)                Low-level wrapper for `::fz_outline_iterator_insert()`.
9892: (4)                Call to insert a new item BEFORE the current point.
9893: (4)                Ownership of pointers are retained by the caller. The item data will be
copied.
9894: (4)                After an insert, we do not change where we are pointing.
9895: (4)                The return code is the same as for next, it indicates the current iterator
position.
9896: (4)                """
9897: (4)                return _mupdf.ll_fz_outline_iterator_insert(iter, item)
9898: (0)            def ll_fz_outline_iterator_item(iter):
9899: (4)                r"""
9900: (4)                Low-level wrapper for `::fz_outline_iterator_item()`.
9901: (4)                Call to get the current outline item.
9902: (4)                Can return NULL. The item is only valid until the next call.
9903: (4)                """
9904: (4)                return _mupdf.ll_fz_outline_iterator_item(iter)
9905: (0)            def ll_fz_outline_iterator_next(iter):
9906: (4)                r"""
9907: (4)                Low-level wrapper for `::fz_outline_iterator_next()`.
9908: (4)                Calls to move the iterator position.
9909: (4)                A negative return value means we could not move as requested. Otherwise:
9910: (4)                0 = the final position has a valid item.
9911: (4)                1 = not a valid item, but we can insert an item here.
9912: (4)                """
9913: (4)                return _mupdf.ll_fz_outline_iterator_next(iter)
9914: (0)            def ll_fz_outline_iterator_prev(iter):
9915: (4)                r""" Low-level wrapper for `::fz_outline_iterator_prev()`."""
9916: (4)                return _mupdf.ll_fz_outline_iterator_prev(iter)
9917: (0)            def ll_fz_outline_iterator_up(iter):
9918: (4)                r""" Low-level wrapper for `::fz_outline_iterator_up()`."""
9919: (4)                return _mupdf.ll_fz_outline_iterator_up(iter)
9920: (0)            def ll_fz_outline_iterator_update(iter, item):
9921: (4)                r"""
9922: (4)                Low-level wrapper for `::fz_outline_iterator_update()`.
9923: (4)                Update the current item properties according to the given item.
9924: (4)                """
9925: (4)                return _mupdf.ll_fz_outline_iterator_update(iter, item)
9926: (0)            def ll_fz_output_accelerator(doc, accel):
9927: (4)                r"""
9928: (4)                Low-level wrapper for `::fz_output_accelerator()`.
9929: (4)                Output accelerator data for the document to a given output
9930: (4)                stream.
9931: (4)                """
9932: (4)                return _mupdf.ll_fz_output_accelerator(doc, accel)
9933: (0)            def ll_fz_output_supports_stream(out):
9934: (4)                r"""
9935: (4)                Low-level wrapper for `::fz_output_supports_stream()`.
```

```
9936: (4)               Query whether a given fz_output supports fz_stream_from_output.
9937: (4)               """
9938: (4)               return _mupdf.ll_fz_output_supports_stream(out)
9939: (0)           def ll_fz_output_xml(out, item, level):
9940: (4)               r"""
9941: (4)               Low-level wrapper for `::fz_output_xml()`.
9942: (4)               Pretty-print an XML tree to given output.
9943: (4)               """
9944: (4)               return _mupdf.ll_fz_output_xml(out, item, level)
9945: (0)           def ll_fz_pack_path(pack, path):
9946: (4)               r"""
9947: (4)               Low-level wrapper for `::fz_pack_path()`.
9948: (4)               Pack a path into the given block.
9949: (4)               To minimise the size of paths, this function allows them to be
9950: (4)               packed into a buffer with other information. Paths can be used
9951: (4)               interchangeably regardless of how they are packed.
9952: (4)               pack: Pointer to a block of data to pack the path into. Should
9953: (4)               be aligned by the caller to the same alignment as required for
9954: (4)               a fz_path pointer.
9955: (4)               path: The path to pack.
9956: (4)               Returns the number of bytes within the block used. Callers can
9957: (4)               access the packed path data by casting the value of pack on
9958: (4)               entry to be a fz_path *.
9959: (4)               Throws exceptions on failure to allocate.
9960: (4)               Implementation details: Paths can be 'unpacked', 'flat', or
9961: (4)               'open'. Standard paths, as created are 'unpacked'. Paths
9962: (4)               will be packed as 'flat', unless they are too large
9963: (4)               (where large indicates that they exceed some private
9964: (4)               implementation defined limits, currently including having
9965: (4)               more than 256 coordinates or commands).
9966: (4)               Large paths are 'open' packed as a header into the given block,
9967: (4)               plus pointers to other data blocks.
9968: (4)               Users should not have to care about whether paths are 'open'
9969: (4)               or 'flat' packed. Simply pack a path (if required), and then
9970: (4)               forget about the details.
9971: (4)               """
9972: (4)               return _mupdf.ll_fz_pack_path(pack, path)
9973: (0)           def ll_fz_packed_path_size(path):
9974: (4)               r"""
9975: (4)               Low-level wrapper for `::fz_packed_path_size()`.
9976: (4)               Return the number of bytes required to pack a path.
9977: (4)               """
9978: (4)               return _mupdf.ll_fz_packed_path_size(path)
9979: (0)           def ll_fz_page_label(page, buf, size):
9980: (4)               r"""
9981: (4)               Low-level wrapper for `::fz_page_label()`.
9982: (4)               Get page label for a given page.
9983: (4)               """
9984: (4)               return _mupdf.ll_fz_page_label(page, buf, size)
9985: (0)           def ll_fz_page_number_from_location(doc, loc):
9986: (4)               r"""
9987: (4)               Low-level wrapper for `::fz_page_number_from_location()`.
9988: (4)               Converts from chapter+page to page number. This may cause many
9989: (4)               chapters to be laid out in order to calculate the number of
9990: (4)               pages within those chapters.
9991: (4)               """
9992: (4)               return _mupdf.ll_fz_page_number_from_location(doc, loc)
9993: (0)           def ll_fz_page_presentation(page, transition, duration):
9994: (4)               r"""
9995: (5)                Low-level wrapper for `::fz_page_presentation()`.
9996: (4)               This function has out-params. Python/C# wrappers look like:
9997: (8)                   `ll_fz_page_presentation(::fz_page *page, ::fz_transition
*transition)` => `(fz_transition *, float duration)`
9998: (8)                   Get the presentation details for a given page.
9999: (8)                   transition: A pointer to a transition struct to fill out.
10000: (8)                  duration: A pointer to a place to set the page duration in
10001: (8)                  seconds. Will be set to 0 if no transition is specified for the
10002: (8)                  page.
10003: (8)                  Returns: a pointer to the transition structure, or NULL if there
```

```
10004: (8)                  is no transition specified for the page.
10005: (4)              """
10006: (4)              return _mupdf.ll_fz_page_presentation(page, transition, duration)
10007: (0)          def ll_fz_page_separations(page):
10008: (4)              r"""
10009: (4)              Low-level wrapper for `::fz_page_separations()`.
10010: (4)              Get the separations details for a page.
10011: (4)              This will be NULL, unless the format specifically supports
10012: (4)              separations (such as PDF files). May be NULL even
10013: (4)              so, if there are no separations on a page.
10014: (4)              Returns a reference that must be dropped.
10015: (4)              """
10016: (4)              return _mupdf.ll_fz_page_separations(page)
10017: (0)          def ll_fz_page_uses_overprint(page):
10018: (4)              r"""
10019: (4)              Low-level wrapper for `::fz_page_uses_overprint()`.
10020: (4)              Query if a given page requires overprint.
10021: (4)              """
10022: (4)              return _mupdf.ll_fz_page_uses_overprint(page)
10023: (0)          def ll_fz_paint_shade(shade, override_cs, ctm, dest, color_params, bbox, eop,
cache):
10024: (4)              r"""
10025: (5)               Low-level wrapper for `::fz_paint_shade()`.
10026: (4)              This function has out-params. Python/C# wrappers look like:
10027: (8)                  `ll_fz_paint_shade(::fz_shade *shade, ::fz_colorspace *override_cs,
::fz_matrix ctm, ::fz_pixmap *dest, ::fz_color_params color_params, ::fz_irect bbox, const
::fz_overprint *eop, ::fz_shade_color_cache **cache)` =>
10028: (8)                  Render a shade to a given pixmap.
10029: (8)                  shade: The shade to paint.
10030: (8)                  override_cs: NULL, or colorspace to override the shades
10031: (8)                  inbuilt colorspace.
10032: (8)                  ctm: The transform to apply.
10033: (8)                  dest: The pixmap to render into.
10034: (8)                  color_params: The color rendering settings
10035: (8)                  bbox: Pointer to a bounding box to limit the rendering
10036: (8)                  of the shade.
10037: (8)                  eop: NULL, or pointer to overprint bitmap.
10038: (8)                  cache: *cache is used to cache color information. If *cache is NULL it
10039: (8)                  is set to point to a new fz_shade_color_cache. If cache is NULL it is
10040: (8)                  ignored.
10041: (4)              """
10042: (4)              return _mupdf.ll_fz_paint_shade(shade, override_cs, ctm, dest,
color_params, bbox, eop, cache)
10043: (0)          def ll_fz_parse_draw_options(options, string):
10044: (4)              r"""
10045: (4)              Low-level wrapper for `::fz_parse_draw_options()`.
10046: (4)              Parse draw device options from a comma separated key-value string.
10047: (4)              """
10048: (4)              return _mupdf.ll_fz_parse_draw_options(options, string)
10049: (0)          def ll_fz_parse_page_range(s, a, b, n):
10050: (4)              r"""
10051: (5)               Low-level wrapper for `::fz_parse_page_range()`.
10052: (4)              This function has out-params. Python/C# wrappers look like:
10053: (8)                  `ll_fz_parse_page_range(const char *s, int n)` => `(const char *, int
a, int b)`
10054: (4)              """
10055: (4)              return _mupdf.ll_fz_parse_page_range(s, a, b, n)
10056: (0)          def ll_fz_parse_pcl_options(opts, args):
10057: (4)              r"""
10058: (4)              Low-level wrapper for `::fz_parse_pcl_options()`.
10059: (4)              Parse PCL options.
10060: (4)              Currently defined options and values are as follows:
10061: (8)                  preset=X    Either "generic" or one of the presets as for
fz_pcl_preset.
10062: (8)                  spacing=0    No vertical spacing capability
10063: (8)                  spacing=1    PCL 3 spacing (<ESC>*p+<n>Y)
10064: (8)                  spacing=2    PCL 4 spacing (<ESC>*b<n>Y)
10065: (8)                  spacing=3    PCL 5 spacing (<ESC>*b<n>Y and clear seed row)
10066: (8)                  mode2        Disable/Enable mode 2 graphics compression
```

```
10067: (8)                  mode3        Disable/Enable mode 3 graphics compression
10068: (8)                  eog_reset    End of graphics (<ESC>*rB) resets all parameters
10069: (8)                  has_duplex    Duplex supported (<ESC>&l<duplex>S)
10070: (8)                  has_papersize    Papersize setting supported (<ESC>&l<sizecode>A)
10071: (8)                  has_copies    Number of copies supported (<ESC>&l<copies>X)
10072: (8)                  is_ljet4pjl    Disable/Enable HP 4PJL model-specific output
10073: (8)                  is_oce9050    Disable/Enable Oce 9050 model-specific output
10074: (4)              """
10075: (4)              return _mupdf.ll_fz_parse_pcl_options(opts, args)
10076: (0)          def ll_fz_parse_pclm_options(opts, args):
10077: (4)              r"""
10078: (4)              Low-level wrapper for `::fz_parse_pclm_options()`.
10079: (4)              Parse PCLm options.
10080: (4)              Currently defined options and values are as follows:
10081: (8)                  compression=none: No compression
10082: (8)                  compression=flate: Flate compression
10083: (8)                  strip-height=n: Strip height (default 16)
10084: (4)              """
10085: (4)              return _mupdf.ll_fz_parse_pclm_options(opts, args)
10086: (0)          def ll_fz_parse_pdfocr_options(opts, args):
10087: (4)              r"""
10088: (4)              Low-level wrapper for `::fz_parse_pdfocr_options()`.
10089: (4)              Parse PDFOCR options.
10090: (4)              Currently defined options and values are as follows:
10091: (8)                  compression=none: No compression
10092: (8)                  compression=flate: Flate compression
10093: (8)                  strip-height=n: Strip height (default 16)
10094: (8)                  ocr-language=<lang>: OCR Language (default eng)
10095: (8)                  ocr-datadir=<datadir>: OCR data path (default rely on TESSDATA_PREFIX)
10096: (4)              """
10097: (4)              return _mupdf.ll_fz_parse_pdfocr_options(opts, args)
10098: (0)          def ll_fz_parse_stext_options(opts, string):
10099: (4)              r"""
10100: (4)              Low-level wrapper for `::fz_parse_stext_options()`.
10101: (4)              Parse stext device options from a comma separated key-value
10102: (4)              string.
10103: (4)              """
10104: (4)              return _mupdf.ll_fz_parse_stext_options(opts, string)
10105: (0)          def ll_fz_parse_xml(buf, preserve_white):
10106: (4)              r"""
10107: (4)              Low-level wrapper for `::fz_parse_xml()`.
10108: (4)              Parse the contents of buffer into a tree of xml nodes.
10109: (4)              preserve_white: whether to keep or delete all-whitespace nodes.
10110: (4)              """
10111: (4)              return _mupdf.ll_fz_parse_xml(buf, preserve_white)
10112: (0)          def ll_fz_parse_xml_archive_entry(dir, filename, preserve_white):
10113: (4)              r"""
10114: (4)              Low-level wrapper for `::fz_parse_xml_archive_entry()`.
10115: (4)              Parse the contents of an archive entry into a tree of xml nodes.
10116: (4)              preserve_white: whether to keep or delete all-whitespace nodes.
10117: (4)              """
10118: (4)              return _mupdf.ll_fz_parse_xml_archive_entry(dir, filename, preserve_white)
10119: (0)          def ll_fz_parse_xml_from_html5(buf):
10120: (4)              r"""
10121: (4)              Low-level wrapper for `::fz_parse_xml_from_html5()`.
10122: (4)              Parse the contents of a buffer into a tree of XML nodes,
10123: (4)              using the HTML5 parsing algorithm.
10124: (4)              """
10125: (4)              return _mupdf.ll_fz_parse_xml_from_html5(buf)
10126: (0)          def ll_fz_parse_xml_stream(stream, preserve_white):
10127: (4)              r"""
10128: (4)              Low-level wrapper for `::fz_parse_xml_stream()`.
10129: (4)              Parse the contents of buffer into a tree of xml nodes.
10130: (4)              preserve_white: whether to keep or delete all-whitespace nodes.
10131: (4)              """
10132: (4)              return _mupdf.ll_fz_parse_xml_stream(stream, preserve_white)
10133: (0)          def ll_fz_pcl_preset(opts, preset):
10134: (4)              r"""
10135: (4)              Low-level wrapper for `::fz_pcl_preset()`.
```

```
10136: (4)             Initialize PCL option struct for a given preset.
10137: (4)             Currently defined presets include:
10138: (8)                 generic    Generic PCL printer
10139: (8)                 ljet4    HP DeskJet
10140: (8)                 dj500    HP DeskJet 500
10141: (8)                 fs600    Kyocera FS-600
10142: (8)                 lj    HP LaserJet, HP LaserJet Plus
10143: (8)                 lj2    HP LaserJet IIp, HP LaserJet IId
10144: (8)                 lj3    HP LaserJet III
10145: (8)                 lj3d    HP LaserJet IIId
10146: (8)                 lj4    HP LaserJet 4
10147: (8)                 lj4pl    HP LaserJet 4 PL
10148: (8)                 lj4d    HP LaserJet 4d
10149: (8)                 lp2563b    HP 2563B line printer
10150: (8)                 oce9050    Oce 9050 Line printer
10151: (4)             """
10152: (4)             return _mupdf.ll_fz_pcl_preset(opts, preset)
10153: (0)         def ll_fz_pdfocr_band_writer_set_progress(writer, progress_fn, progress_arg):
10154: (4)             r"""
10155: (4)             Low-level wrapper for `::fz_pdfocr_band_writer_set_progress()`.
10156: (4)             Set the progress callback for a pdfocr bandwriter.
10157: (4)             """
10158: (4)             return _mupdf.ll_fz_pdfocr_band_writer_set_progress(writer, progress_fn,
progress_arg)
10159: (0)         def ll_fz_pdfocr_writer_set_progress(writer, progress, arg_3):
10160: (4)             r""" Low-level wrapper for `::fz_pdfocr_writer_set_progress()`."""
10161: (4)             return _mupdf.ll_fz_pdfocr_writer_set_progress(writer, progress, arg_3)
10162: (0)         def ll_fz_peek_byte(stm):
10163: (4)             r"""
10164: (4)             Low-level wrapper for `::fz_peek_byte()`.
10165: (4)             Peek at the next byte in a stream.
10166: (4)             stm: The stream to peek at.
10167: (4)             Returns -1 for EOF, or the next byte that will be read.
10168: (4)             """
10169: (4)             return _mupdf.ll_fz_peek_byte(stm)
10170: (0)         def ll_fz_pixmap_alpha(pix):
10171: (4)             r"""
10172: (4)             Low-level wrapper for `::fz_pixmap_alpha()`.
10173: (4)             Return the number of alpha planes in a pixmap.
10174: (4)             Returns the number of alphas. Does not throw exceptions.
10175: (4)             """
10176: (4)             return _mupdf.ll_fz_pixmap_alpha(pix)
10177: (0)         def ll_fz_pixmap_bbox(pix):
10178: (4)             r"""
10179: (4)             Low-level wrapper for `::fz_pixmap_bbox()`.
10180: (4)             Return the bounding box for a pixmap.
10181: (4)             """
10182: (4)             return _mupdf.ll_fz_pixmap_bbox(pix)
10183: (0)         def ll_fz_pixmap_colorants(pix):
10184: (4)             r"""
10185: (4)             Low-level wrapper for `::fz_pixmap_colorants()`.
10186: (4)             Return the number of colorants in a pixmap.
10187: (4)             Returns the number of colorants (components, less any spots and
10188: (4)             alpha).
10189: (4)             """
10190: (4)             return _mupdf.ll_fz_pixmap_colorants(pix)
10191: (0)         def ll_fz_pixmap_colorspace(pix):
10192: (4)             r"""
10193: (4)             Low-level wrapper for `::fz_pixmap_colorspace()`.
10194: (4)             Return the colorspace of a pixmap
10195: (4)             Returns colorspace.
10196: (4)             """
10197: (4)             return _mupdf.ll_fz_pixmap_colorspace(pix)
10198: (0)         def ll_fz_pixmap_components(pix):
10199: (4)             r"""
10200: (4)             Low-level wrapper for `::fz_pixmap_components()`.
10201: (4)             Return the number of components in a pixmap.
10202: (4)             Returns the number of components (including spots and alpha).
10203: (4)             """
```

```
10204: (4)             return _mupdf.ll_fz_pixmap_components(pix)
10205: (0)         def ll_fz_pixmap_height(pix):
10206: (4)             r"""
10207: (4)             Low-level wrapper for `::fz_pixmap_height()`.
10208: (4)             Return the height of the pixmap in pixels.
10209: (4)             """
10210: (4)             return _mupdf.ll_fz_pixmap_height(pix)
10211: (0)         def ll_fz_pixmap_image_tile(cimg):
10212: (4)             r"""
10213: (4)             Low-level wrapper for `::fz_pixmap_image_tile()`.
10214: (4)             Retrieve the underlying fz_pixmap for an image.
10215: (4)             Returns a pointer to the underlying fz_pixmap for an image,
10216: (4)             or NULL if this image is not based upon an fz_pixmap.
10217: (4)             No reference is returned. Lifespan is limited to that of
10218: (4)             the image itself. If required, use fz_keep_pixmap to take
10219: (4)             a reference to keep it longer.
10220: (4)             """
10221: (4)             return _mupdf.ll_fz_pixmap_image_tile(cimg)
10222: (0)         def ll_fz_pixmap_samples(pix):
10223: (4)             r"""
10224: (4)             Low-level wrapper for `::fz_pixmap_samples()`.
10225: (4)             Returns a pointer to the pixel data of a pixmap.
10226: (4)             Returns the pointer.
10227: (4)             """
10228: (4)             return _mupdf.ll_fz_pixmap_samples(pix)
10229: (0)         def ll_fz_pixmap_samples_int(pixmap):
10230: (4)             r""" Low-level wrapper for `::fz_pixmap_samples_int()`."""
10231: (4)             return _mupdf.ll_fz_pixmap_samples_int(pixmap)
10232: (0)         def ll_fz_pixmap_size(pix):
10233: (4)             r"""
10234: (4)             Low-level wrapper for `::fz_pixmap_size()`.
10235: (4)             Return sizeof fz_pixmap plus size of data, in bytes.
10236: (4)             """
10237: (4)             return _mupdf.ll_fz_pixmap_size(pix)
10238: (0)         def ll_fz_pixmap_spots(pix):
10239: (4)             r"""
10240: (4)             Low-level wrapper for `::fz_pixmap_spots()`.
10241: (4)             Return the number of spots in a pixmap.
10242: (4)             Returns the number of spots (components, less colorants and
10243: (4)             alpha). Does not throw exceptions.
10244: (4)             """
10245: (4)             return _mupdf.ll_fz_pixmap_spots(pix)
10246: (0)         def ll_fz_pixmap_stride(pix):
10247: (4)             r"""
10248: (4)             Low-level wrapper for `::fz_pixmap_stride()`.
10249: (4)             Return the number of bytes in a row in the pixmap.
10250: (4)             """
10251: (4)             return _mupdf.ll_fz_pixmap_stride(pix)
10252: (0)         def ll_fz_pixmap_width(pix):
10253: (4)             r"""
10254: (4)             Low-level wrapper for `::fz_pixmap_width()`.
10255: (4)             Return the width of the pixmap in pixels.
10256: (4)             """
10257: (4)             return _mupdf.ll_fz_pixmap_width(pix)
10258: (0)         def ll_fz_pixmap_x(pix):
10259: (4)             r"""
10260: (4)             Low-level wrapper for `::fz_pixmap_x()`.
10261: (4)             Return the x value of the pixmap in pixels.
10262: (4)             """
10263: (4)             return _mupdf.ll_fz_pixmap_x(pix)
10264: (0)         def ll_fz_pixmap_y(pix):
10265: (4)             r"""
10266: (4)             Low-level wrapper for `::fz_pixmap_y()`.
10267: (4)             Return the y value of the pixmap in pixels.
10268: (4)             """
10269: (4)             return _mupdf.ll_fz_pixmap_y(pix)
10270: (0)         def ll_fz_place_story(story, where, filled):
10271: (4)             r""" Low-level wrapper for `::fz_place_story()`."""
10272: (4)             return _mupdf.ll_fz_place_story(story, where, filled)
```

```
(0)          def ll_fz_place_story_flags(story, where, filled, flags):
(4)              r""" Low-level wrapper for `::fz_place_story_flags()`."""
(4)              return _mupdf.ll_fz_place_story_flags(story, where, filled, flags)
(0)          def ll_fz_pool_alloc(pool, size):
(4)              r"""
(4)              Low-level wrapper for `::fz_pool_alloc()`.
(4)              Allocate a block of size bytes from the pool.
(4)              """
(4)              return _mupdf.ll_fz_pool_alloc(pool, size)
(0)          def ll_fz_pool_size(pool):
(4)              r"""
(4)              Low-level wrapper for `::fz_pool_size()`.
(4)              The current size of the pool.
(4)              The number of bytes of storage currently allocated to the pool.
(4)              This is the total of the storage used for the blocks making
(4)              up the pool, rather then total of the allocated blocks so far,
(4)              so it will increase in 'lumps'.
(4)              from the pool, then the pool size may still be X
(4)              """
(4)              return _mupdf.ll_fz_pool_size(pool)
(0)          def ll_fz_pool_strdup(pool, s):
(4)              r"""
(4)              Low-level wrapper for `::fz_pool_strdup()`.
(4)              strdup equivalent allocating from the pool.
(4)              """
(4)              return _mupdf.ll_fz_pool_strdup(pool, s)
(0)          def ll_fz_pop_clip(dev):
(4)              r""" Low-level wrapper for `::fz_pop_clip()`."""
(4)              return _mupdf.ll_fz_pop_clip(dev)
(0)          def ll_fz_post_scale(m, sx, sy):
(4)              r"""
(4)              Low-level wrapper for `::fz_post_scale()`.
(4)              Scale a matrix by postmultiplication.
(4)              m: Pointer to the matrix to scale
(4)              sx, sy: Scaling factors along the X- and Y-axes. A scaling
(4)              factor of 1.0 will not cause any scaling along the relevant
(4)              axis.
(4)              Returns m (updated).
(4)              """
(4)              return _mupdf.ll_fz_post_scale(m, sx, sy)
(0)          def ll_fz_pre_rotate(m, degrees):
(4)              r"""
(4)              Low-level wrapper for `::fz_pre_rotate()`.
(4)              Rotate a transformation by premultiplying.
(4)              The premultiplied matrix is of the form
(4)              [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
(4)              m: Pointer to matrix to premultiply.
(4)              degrees: Degrees of counter clockwise rotation. Values less
(4)              than zero and greater than 360 are handled as expected.
(4)              Returns m (updated).
(4)              """
(4)              return _mupdf.ll_fz_pre_rotate(m, degrees)
(0)          def ll_fz_pre_scale(m, sx, sy):
(4)              r"""
(4)              Low-level wrapper for `::fz_pre_scale()`.
(4)              Scale a matrix by premultiplication.
(4)              m: Pointer to the matrix to scale
(4)              sx, sy: Scaling factors along the X- and Y-axes. A scaling
(4)              factor of 1.0 will not cause any scaling along the relevant
(4)              axis.
(4)              Returns m (updated).
(4)              """
(4)              return _mupdf.ll_fz_pre_scale(m, sx, sy)
(0)          def ll_fz_pre_shear(m, sx, sy):
(4)              r"""
(4)              Low-level wrapper for `::fz_pre_shear()`.
(4)              Premultiply a matrix with a shearing matrix.
(4)              The shearing matrix is of the form [ 1 sy sx 1 0 0 ].
(4)              m: pointer to matrix to premultiply
```

```
10342: (4)             sx, sy: Shearing factors. A shearing factor of 0.0 will not
10343: (4)             cause any shearing along the relevant axis.
10344: (4)             Returns m (updated).
10345: (4)             """
10346: (4)             return _mupdf.ll_fz_pre_shear(m, sx, sy)
10347: (0)         def ll_fz_pre_translate(m, tx, ty):
10348: (4)             r"""
10349: (4)             Low-level wrapper for `::fz_pre_translate()`.
10350: (4)             Translate a matrix by premultiplication.
10351: (4)             m: The matrix to translate
10352: (4)             tx, ty: Translation distances along the X- and Y-axes. A
10353: (4)             translation of 0 will not cause any translation along the
10354: (4)             relevant axis.
10355: (4)             Returns m.
10356: (4)             """
10357: (4)             return _mupdf.ll_fz_pre_translate(m, tx, ty)
10358: (0)         def ll_fz_prepare_t3_glyph(font, gid):
10359: (4)             r"""
10360: (4)             Low-level wrapper for `::fz_prepare_t3_glyph()`.
10361: (4)             Force a type3 font to cache the displaylist for a given glyph
10362: (4)             id.
10363: (4)             This caching can involve reading the underlying file, so must
10364: (4)             happen ahead of time, so we aren't suddenly forced to read the
10365: (4)             file while playing a displaylist back.
10366: (4)             """
10367: (4)             return _mupdf.ll_fz_prepare_t3_glyph(font, gid)
10368: (0)         def ll_fz_previous_page(doc, loc):
10369: (4)             r"""
10370: (4)             Low-level wrapper for `::fz_previous_page()`.
10371: (4)             Function to get the location of the previous page (allowing for
10372: (4)             the end of chapters etc). If already at the start of the
10373: (4)             document, returns the current page.
10374: (4)             """
10375: (4)             return _mupdf.ll_fz_previous_page(doc, loc)
10376: (0)         def ll_fz_print_stext_header_as_html(out):
10377: (4)             r""" Low-level wrapper for `::fz_print_stext_header_as_html()`."""
10378: (4)             return _mupdf.ll_fz_print_stext_header_as_html(out)
10379: (0)         def ll_fz_print_stext_header_as_xhtml(out):
10380: (4)             r""" Low-level wrapper for `::fz_print_stext_header_as_xhtml()`."""
10381: (4)             return _mupdf.ll_fz_print_stext_header_as_xhtml(out)
10382: (0)         def ll_fz_print_stext_page_as_html(out, page, id):
10383: (4)             r"""
10384: (4)             Low-level wrapper for `::fz_print_stext_page_as_html()`.
10385: (4)             Output structured text to a file in HTML (visual) format.
10386: (4)             """
10387: (4)             return _mupdf.ll_fz_print_stext_page_as_html(out, page, id)
10388: (0)         def ll_fz_print_stext_page_as_json(out, page, scale):
10389: (4)             r"""
10390: (4)             Low-level wrapper for `::fz_print_stext_page_as_json()`.
10391: (4)             Output structured text to a file in JSON format.
10392: (4)             """
10393: (4)             return _mupdf.ll_fz_print_stext_page_as_json(out, page, scale)
10394: (0)         def ll_fz_print_stext_page_as_text(out, page):
10395: (4)             r"""
10396: (4)             Low-level wrapper for `::fz_print_stext_page_as_text()`.
10397: (4)             Output structured text to a file in plain-text UTF-8 format.
10398: (4)             """
10399: (4)             return _mupdf.ll_fz_print_stext_page_as_text(out, page)
10400: (0)         def ll_fz_print_stext_page_as_xhtml(out, page, id):
10401: (4)             r"""
10402: (4)             Low-level wrapper for `::fz_print_stext_page_as_xhtml()`.
10403: (4)             Output structured text to a file in XHTML (semantic) format.
10404: (4)             """
10405: (4)             return _mupdf.ll_fz_print_stext_page_as_xhtml(out, page, id)
10406: (0)         def ll_fz_print_stext_page_as_xml(out, page, id):
10407: (4)             r"""
10408: (4)             Low-level wrapper for `::fz_print_stext_page_as_xml()`.
10409: (4)             Output structured text to a file in XML format.
10410: (4)             """
```

```
10411: (4)             return _mupdf.ll_fz_print_stext_page_as_xml(out, page, id)
10412: (0)         def ll_fz_print_stext_trailer_as_html(out):
10413: (4)             r""" Low-level wrapper for `::fz_print_stext_trailer_as_html()`."""
10414: (4)             return _mupdf.ll_fz_print_stext_trailer_as_html(out)
10415: (0)         def ll_fz_print_stext_trailer_as_xhtml(out):
10416: (4)             r""" Low-level wrapper for `::fz_print_stext_trailer_as_xhtml()`."""
10417: (4)             return _mupdf.ll_fz_print_stext_trailer_as_xhtml(out)
10418: (0)         def ll_fz_process_opened_pages(doc, process_openend_page, state):
10419: (4)             r"""
10420: (4)             Low-level wrapper for `::fz_process_opened_pages()`.
10421: (4)             Iterates over all opened pages of the document, calling the
10422: (4)             provided callback for each page for processing. If the callback
10423: (4)             returns non-NULL then the iteration stops and that value is returned
10424: (4)             to the called of fz_process_opened_pages().
10425: (4)             The state pointer provided to fz_process_opened_pages() is
10426: (4)             passed on to the callback but is owned by the caller.
10427: (4)             Returns the first non-NULL value returned by the callback,
10428: (4)             or NULL if the callback returned NULL for all opened pages.
10429: (4)             """
10430: (4)             return _mupdf.ll_fz_process_opened_pages(doc, process_openend_page, state)
10431: (0)         def ll_fz_process_shade(shade, ctm, scissor, prepare, process, process_arg):
10432: (4)             r"""
10433: (4)             Low-level wrapper for `::fz_process_shade()`.
10434: (4)             Process a shade, using supplied callback functions. This
10435: (4)             decomposes the shading to a mesh (even ones that are not
10436: (4)             natively meshes, such as linear or radial shadings), and
10437: (4)             processes triangles from those meshes.
10438: (4)             shade: The shade to process.
10439: (4)             ctm: The transform to use
10440: (4)             prepare: Callback function to 'prepare' each vertex.
10441: (4)             This function is passed an array of floats, and populates
10442: (4)             a fz_vertex structure.
10443: (4)             process: This function is passed 3 pointers to vertex
10444: (4)             structures, and actually performs the processing (typically
10445: (4)             filling the area between the vertexes).
10446: (4)             process_arg: An opaque argument passed through from caller
10447: (4)             to callback functions.
10448: (4)             """
10449: (4)             return _mupdf.ll_fz_process_shade(shade, ctm, scissor, prepare, process,
process_arg)
10450: (0)         def ll_fz_ptr_heap_insert(heap, v, HEAP_CMP):
10451: (4)             r""" Low-level wrapper for `::fz_ptr_heap_insert()`."""
10452: (4)             return _mupdf.ll_fz_ptr_heap_insert(heap, v, HEAP_CMP)
10453: (0)         def ll_fz_ptr_heap_sort(heap, HEAP_CMP):
10454: (4)             r""" Low-level wrapper for `::fz_ptr_heap_sort()`."""
10455: (4)             return _mupdf.ll_fz_ptr_heap_sort(heap, HEAP_CMP)
10456: (0)         def ll_fz_ptr_heap_uniq(heap, HEAP_CMP):
10457: (4)             r""" Low-level wrapper for `::fz_ptr_heap_uniq()`."""
10458: (4)             return _mupdf.ll_fz_ptr_heap_uniq(heap, HEAP_CMP)
10459: (0)         def ll_fz_purge_glyph_cache():
10460: (4)             r"""
10461: (4)             Low-level wrapper for `::fz_purge_glyph_cache()`.
10462: (4)             Purge all the glyphs from the cache.
10463: (4)             """
10464: (4)             return _mupdf.ll_fz_purge_glyph_cache()
10465: (0)         def ll_fz_quad_from_rect(r):
10466: (4)             r"""
10467: (4)             Low-level wrapper for `::fz_quad_from_rect()`.
10468: (4)             Convert a rect to a quad (losslessly).
10469: (4)             """
10470: (4)             return _mupdf.ll_fz_quad_from_rect(r)
10471: (0)         def ll_fz_quadto(path, x0, y0, x1, y1):
10472: (4)             r"""
10473: (4)             Low-level wrapper for `::fz_quadto()`.
10474: (4)             Append a 'quadto' command to an open path. (For a
10475: (4)             quadratic bezier).
10476: (4)             path: The path to modify.
10477: (4)             x0, y0: The control coordinates for the quadratic curve.
10478: (4)             x1, y1: The end coordinates for the quadratic curve.
```

```
10479: (4)              Throws exceptions on failure to allocate, or attempting to
10480: (4)              modify a packed path.
10481: (4)              """
10482: (4)              return _mupdf.ll_fz_quadto(path, x0, y0, x1, y1)
10483: (0)          def ll_fz_read(stm, data, len):
10484: (4)              r"""
10485: (4)              Low-level wrapper for `::fz_read()`.
10486: (4)              Read from a stream into a given data block.
10487: (4)              stm: The stream to read from.
10488: (4)              data: The data block to read into.
10489: (4)              len: The length of the data block (in bytes).
10490: (4)              Returns the number of bytes read. May throw exceptions.
10491: (4)              """
10492: (4)              return _mupdf.ll_fz_read(stm, data, len)
10493: (0)          def ll_fz_read_all(stm, initial):
10494: (4)              r"""
10495: (4)              Low-level wrapper for `::fz_read_all()`.
10496: (4)              Read all of a stream into a buffer.
10497: (4)              stm: The stream to read from
10498: (4)              initial: Suggested initial size for the buffer.
10499: (4)              Returns a buffer created from reading from the stream. May throw
10500: (4)              exceptions on failure to allocate.
10501: (4)              """
10502: (4)              return _mupdf.ll_fz_read_all(stm, initial)
10503: (0)          def ll_fz_read_archive_entry(arch, name):
10504: (4)              r"""
10505: (4)              Low-level wrapper for `::fz_read_archive_entry()`.
10506: (4)              Reads all bytes in an archive entry
10507: (4)              into a buffer.
10508: (4)              name: Entry name to look for, this must be an exact match to
10509: (4)              the entry name in the archive.
10510: (4)              Throws an exception if a matching entry cannot be found.
10511: (4)              """
10512: (4)              return _mupdf.ll_fz_read_archive_entry(arch, name)
10513: (0)          def ll_fz_read_best(stm, initial, truncated, worst_case):
10514: (4)              r"""
10515: (5)               Low-level wrapper for `::fz_read_best()`.
10516: (4)              This function has out-params. Python/C# wrappers look like:
10517: (8)                  `ll_fz_read_best(::fz_stream *stm, size_t initial, size_t worst_case)`
=> `(fz_buffer *, int truncated)`
10518: (8)                  Attempt to read a stream into a buffer. If truncated
10519: (8)                  is NULL behaves as fz_read_all, sets a truncated flag in case of
10520: (8)                  error.
10521: (8)                  stm: The stream to read from.
10522: (8)                  initial: Suggested initial size for the buffer.
10523: (8)                  truncated: Flag to store success/failure indication in.
10524: (8)                  worst_case: 0 for unknown, otherwise an upper bound for the
10525: (8)                  size of the stream.
10526: (8)                  Returns a buffer created from reading from the stream.
10527: (4)              """
10528: (4)              return _mupdf.ll_fz_read_best(stm, initial, truncated, worst_case)
10529: (0)          def ll_fz_read_bits(stm, n):
10530: (4)              r"""
10531: (4)              Low-level wrapper for `::fz_read_bits()`.
10532: (4)              Read the next n bits from a stream (assumed to
10533: (4)              be packed most significant bit first).
10534: (4)              stm: The stream to read from.
10535: (4)              n: The number of bits to read, between 1 and 8*sizeof(int)
10536: (4)              inclusive.
10537: (4)              Returns -1 for EOF, or the required number of bits.
10538: (4)              """
10539: (4)              return _mupdf.ll_fz_read_bits(stm, n)
10540: (0)          def ll_fz_read_byte(stm):
10541: (4)              r"""
10542: (4)              Low-level wrapper for `::fz_read_byte()`.
10543: (4)              Read the next byte from a stream.
10544: (4)              stm: The stream t read from.
10545: (4)              Returns -1 for end of stream, or the next byte. May
10546: (4)              throw exceptions.
```

```
10547: (4)             """
10548: (4)             return _mupdf.ll_fz_read_byte(stm)
10549: (0)         def ll_fz_read_file(filename):
10550: (4)             r"""
10551: (4)             Low-level wrapper for `::fz_read_file()`.
10552: (4)             Read all the contents of a file into a buffer.
10553: (4)             """
10554: (4)             return _mupdf.ll_fz_read_file(filename)
10555: (0)         def ll_fz_read_float(stm):
10556: (4)             r""" Low-level wrapper for `::fz_read_float()`."""
10557: (4)             return _mupdf.ll_fz_read_float(stm)
10558: (0)         def ll_fz_read_float_le(stm):
10559: (4)             r""" Low-level wrapper for `::fz_read_float_le()`."""
10560: (4)             return _mupdf.ll_fz_read_float_le(stm)
10561: (0)         def ll_fz_read_int16(stm):
10562: (4)             r""" Low-level wrapper for `::fz_read_int16()`."""
10563: (4)             return _mupdf.ll_fz_read_int16(stm)
10564: (0)         def ll_fz_read_int16_le(stm):
10565: (4)             r""" Low-level wrapper for `::fz_read_int16_le()`."""
10566: (4)             return _mupdf.ll_fz_read_int16_le(stm)
10567: (0)         def ll_fz_read_int32(stm):
10568: (4)             r""" Low-level wrapper for `::fz_read_int32()`."""
10569: (4)             return _mupdf.ll_fz_read_int32(stm)
10570: (0)         def ll_fz_read_int32_le(stm):
10571: (4)             r""" Low-level wrapper for `::fz_read_int32_le()`."""
10572: (4)             return _mupdf.ll_fz_read_int32_le(stm)
10573: (0)         def ll_fz_read_int64(stm):
10574: (4)             r""" Low-level wrapper for `::fz_read_int64()`."""
10575: (4)             return _mupdf.ll_fz_read_int64(stm)
10576: (0)         def ll_fz_read_int64_le(stm):
10577: (4)             r""" Low-level wrapper for `::fz_read_int64_le()`."""
10578: (4)             return _mupdf.ll_fz_read_int64_le(stm)
10579: (0)         def ll_fz_read_line(stm, buf, max):
10580: (4)             r"""
10581: (4)             Low-level wrapper for `::fz_read_line()`.
10582: (4)             Read a line from stream into the buffer until either a
10583: (4)             terminating newline or EOF, which it replaces with a null byte
10584: (4)             ('').
10585: (4)             Returns buf on success, and NULL when end of file occurs while
10586: (4)             no characters have been read.
10587: (4)             """
10588: (4)             return _mupdf.ll_fz_read_line(stm, buf, max)
10589: (0)         def ll_fz_read_rbits(stm, n):
10590: (4)             r"""
10591: (4)             Low-level wrapper for `::fz_read_rbits()`.
10592: (4)             Read the next n bits from a stream (assumed to
10593: (4)             be packed least significant bit first).
10594: (4)             stm: The stream to read from.
10595: (4)             n: The number of bits to read, between 1 and 8*sizeof(int)
10596: (4)             inclusive.
10597: (4)             Returns (unsigned int)-1 for EOF, or the required number of bits.
10598: (4)             """
10599: (4)             return _mupdf.ll_fz_read_rbits(stm, n)
10600: (0)         def ll_fz_read_rune(_in):
10601: (4)             r"""
10602: (4)             Low-level wrapper for `::fz_read_rune()`.
10603: (4)             Read a utf-8 rune from a stream.
10604: (4)             In the event of encountering badly formatted utf-8 codes
10605: (4)             (such as a leading code with an unexpected number of following
10606: (4)             codes) no error/exception is given, but undefined values may be
10607: (4)             returned.
10608: (4)             """
10609: (4)             return _mupdf.ll_fz_read_rune(_in)
10610: (0)         def ll_fz_read_string(stm, buffer, len):
10611: (4)             r"""
10612: (4)             Low-level wrapper for `::fz_read_string()`.
10613: (4)             Read a null terminated string from the stream into
10614: (4)             a buffer of a given length. The buffer will be null terminated.
10615: (4)             Throws on failure (including the failure to fit the entire
```

```
10616: (4)             string including the terminator into the buffer).
10617: (4)             """
10618: (4)             return _mupdf.ll_fz_read_string(stm, buffer, len)
10619: (0)         def ll_fz_read_uint16(stm):
10620: (4)             r"""
10621: (4)             Low-level wrapper for `::fz_read_uint16()`.
10622: (4)             fz_read_[u]int(16|24|32|64)(_le)?
10623: (4)             Read a 16/32/64 bit signed/unsigned integer from stream,
10624: (4)             in big or little-endian byte orders.
10625: (4)             Throws an exception if EOF is encountered.
10626: (4)             """
10627: (4)             return _mupdf.ll_fz_read_uint16(stm)
10628: (0)         def ll_fz_read_uint16_le(stm):
10629: (4)             r""" Low-level wrapper for `::fz_read_uint16_le()`."""
10630: (4)             return _mupdf.ll_fz_read_uint16_le(stm)
10631: (0)         def ll_fz_read_uint24(stm):
10632: (4)             r""" Low-level wrapper for `::fz_read_uint24()`."""
10633: (4)             return _mupdf.ll_fz_read_uint24(stm)
10634: (0)         def ll_fz_read_uint24_le(stm):
10635: (4)             r""" Low-level wrapper for `::fz_read_uint24_le()`."""
10636: (4)             return _mupdf.ll_fz_read_uint24_le(stm)
10637: (0)         def ll_fz_read_uint32(stm):
10638: (4)             r""" Low-level wrapper for `::fz_read_uint32()`."""
10639: (4)             return _mupdf.ll_fz_read_uint32(stm)
10640: (0)         def ll_fz_read_uint32_le(stm):
10641: (4)             r""" Low-level wrapper for `::fz_read_uint32_le()`."""
10642: (4)             return _mupdf.ll_fz_read_uint32_le(stm)
10643: (0)         def ll_fz_read_uint64(stm):
10644: (4)             r""" Low-level wrapper for `::fz_read_uint64()`."""
10645: (4)             return _mupdf.ll_fz_read_uint64(stm)
10646: (0)         def ll_fz_read_uint64_le(stm):
10647: (4)             r""" Low-level wrapper for `::fz_read_uint64_le()`."""
10648: (4)             return _mupdf.ll_fz_read_uint64_le(stm)
10649: (0)         def ll_fz_read_utf16_be(stm):
10650: (4)             r""" Low-level wrapper for `::fz_read_utf16_be()`."""
10651: (4)             return _mupdf.ll_fz_read_utf16_be(stm)
10652: (0)         def ll_fz_read_utf16_le(stm):
10653: (4)             r"""
10654: (4)             Low-level wrapper for `::fz_read_utf16_le()`.
10655: (4)             Read a utf-16 rune from a stream. (little endian and
10656: (4)             big endian respectively).
10657: (4)             In the event of encountering badly formatted utf-16 codes
10658: (4)             (mismatched surrogates) no error/exception is given, but
10659: (4)             undefined values may be returned.
10660: (4)             """
10661: (4)             return _mupdf.ll_fz_read_utf16_le(stm)
10662: (0)         def ll_fz_realloc(p, size):
10663: (4)             r"""
10664: (4)             Low-level wrapper for `::fz_realloc()`.
10665: (4)             Reallocates a block of memory to given size. Existing contents
10666: (4)             up to min(old_size,new_size) are maintained. The rest of the
10667: (4)             block is uninitialised.
10668: (4)             fz_realloc(ctx, NULL, size) behaves like fz_malloc(ctx, size).
10669: (4)             fz_realloc(ctx, p, 0); behaves like fz_free(ctx, p).
10670: (4)             Throws exception in the event of failure to allocate.
10671: (4)             """
10672: (4)             return _mupdf.ll_fz_realloc(p, size)
10673: (0)         def ll_fz_realloc_no_throw(p, size):
10674: (4)             r"""
10675: (4)             Low-level wrapper for `::fz_realloc_no_throw()`.
10676: (4)             fz_realloc equivalent that returns NULL rather than throwing
10677: (4)             exceptions.
10678: (4)             """
10679: (4)             return _mupdf.ll_fz_realloc_no_throw(p, size)
10680: (0)         def ll_fz_realpath(path, resolved_path):
10681: (4)             r"""
10682: (4)             Low-level wrapper for `::fz_realpath()`.
10683: (4)             Resolve a path to an absolute file name.
10684: (4)             The resolved path buffer must be of at least PATH_MAX size.
```

```
10685: (4)             """
10686: (4)             return _mupdf.ll_fz_realpath(path, resolved_path)
10687: (0)         def ll_fz_recognize_document(magic):
10688: (4)             r"""
10689: (4)             Low-level wrapper for `::fz_recognize_document()`.
10690: (4)             Given a magic find a document handler that can handle a
10691: (4)             document of this type.
10692: (4)             magic: Can be a filename extension (including initial period) or
10693: (4)             a mimetype.
10694: (4)             """
10695: (4)             return _mupdf.ll_fz_recognize_document(magic)
10696: (0)         def ll_fz_recognize_document_content(filename):
10697: (4)             r"""
10698: (4)             Low-level wrapper for `::fz_recognize_document_content()`.
10699: (4)             Given a filename find a document handler that can handle a
10700: (4)             document of this type.
10701: (4)             filename: The filename of the document. This will be opened and sampled
10702: (4)             to check data.
10703: (4)             """
10704: (4)             return _mupdf.ll_fz_recognize_document_content(filename)
10705: (0)         def ll_fz_recognize_document_stream_and_dir_content(stream, dir, magic):
10706: (4)             r"""
10707: (4)             Low-level wrapper for `::fz_recognize_document_stream_and_dir_content()`.
10708: (4)             Given a magic find a document handler that can handle a
10709: (4)             document of this type.
10710: (4)             stream: the file stream to sample. May be NULL if the document is
10711: (4)             a directory.
10712: (4)             dir: an fz_archive representing the directory from which the
10713: (4)             stream was opened (or NULL).
10714: (4)             magic: Can be a filename extension (including initial period) or
10715: (4)             a mimetype.
10716: (4)             """
10717: (4)             return _mupdf.ll_fz_recognize_document_stream_and_dir_content(stream, dir,
magic)
10718: (0)         def ll_fz_recognize_document_stream_content(stream, magic):
10719: (4)             r"""
10720: (4)             Low-level wrapper for `::fz_recognize_document_stream_content()`.
10721: (4)             Given a magic find a document handler that can handle a
10722: (4)             document of this type.
10723: (4)             stream: the file stream to sample. May be NULL if the document is
10724: (4)             a directory.
10725: (4)             magic: Can be a filename extension (including initial period) or
10726: (4)             a mimetype.
10727: (4)             """
10728: (4)             return _mupdf.ll_fz_recognize_document_stream_content(stream, magic)
10729: (0)         def ll_fz_recognize_image_format(p):
10730: (4)             r"""
10731: (4)             Low-level wrapper for `::fz_recognize_image_format()`.
10732: (4)             Recognise image format strings in the first 8 bytes from image
10733: (4)             data.
10734: (4)             """
10735: (4)             return _mupdf.ll_fz_recognize_image_format(p)
10736: (0)         def ll_fz_rect_from_irect(bbox):
10737: (4)             r"""
10738: (4)             Low-level wrapper for `::fz_rect_from_irect()`.
10739: (4)             Convert a bbox into a rect.
10740: (4)             For our purposes, a rect can represent all the values we meet in
10741: (4)             a bbox, so nothing can go wrong.
10742: (4)             rect: A place to store the generated rectangle.
10743: (4)             bbox: The bbox to convert.
10744: (4)             Returns rect (updated).
10745: (4)             """
10746: (4)             return _mupdf.ll_fz_rect_from_irect(bbox)
10747: (0)         def ll_fz_rect_from_quad(q):
10748: (4)             r"""
10749: (4)             Low-level wrapper for `::fz_rect_from_quad()`.
10750: (4)             Convert a quad to the smallest rect that covers it.
10751: (4)             """
10752: (4)             return _mupdf.ll_fz_rect_from_quad(q)
```

```
(0)          def ll_fz_rectto(path, x0, y0, x1, y1):
(4)              r"""
(4)              Low-level wrapper for `::fz_rectto()`.
(4)              Append a 'rectto' command to an open path.
(4)              The rectangle is equivalent to:
(8)                  moveto x0 y0
(8)                  lineto x1 y0
(8)                  lineto x1 y1
(8)                  lineto x0 y1
(8)                  closepath
(4)              path: The path to modify.
(4)              x0, y0: First corner of the rectangle.
(4)              x1, y1: Second corner of the rectangle.
(4)              Throws exceptions on failure to allocate, or attempting to
(4)              modify a packed path.
(4)              """
(4)              return _mupdf.ll_fz_rectto(path, x0, y0, x1, y1)
(0)          def ll_fz_register_archive_handler(handler):
(4)              r""" Low-level wrapper for `::fz_register_archive_handler()`."""
(4)              return _mupdf.ll_fz_register_archive_handler(handler)
(0)          def ll_fz_register_document_handler(handler):
(4)              r"""
(4)              Low-level wrapper for `::fz_register_document_handler()`.
(4)              Register a handler for a document type.
(4)              handler: The handler to register. This must live on for the duration of
the
(4)              use of this handler. It will be passed back to the handler for calls so
(4)              the caller can use it to retrieve state.
(4)              """
(4)              return _mupdf.ll_fz_register_document_handler(handler)
(0)          def ll_fz_register_document_handlers():
(4)              r"""
(4)              Low-level wrapper for `::fz_register_document_handlers()`.
(4)              Register handlers for all the standard document types supported in
(4)              this build.
(4)              """
(4)              return _mupdf.ll_fz_register_document_handlers()
(0)          def ll_fz_remove_item(drop, key, type):
(4)              r"""
(4)              Low-level wrapper for `::fz_remove_item()`.
(4)              Remove an item from the store.
(4)              If an item indexed by the given key exists in the store, remove
(4)              it.
(4)              drop: The function used to free the value (to ensure we get a
(4)              value of the correct type).
(4)              key: The key used to find the item to remove.
(4)              type: Functions used to manipulate the key.
(4)              """
(4)              return _mupdf.ll_fz_remove_item(drop, key, type)
(0)          def ll_fz_render_flags(dev, set, clear):
(4)              r""" Low-level wrapper for `::fz_render_flags()`."""
(4)              return _mupdf.ll_fz_render_flags(dev, set, clear)
(0)          def ll_fz_render_glyph_pixmap(font, gid, ctm, scissor, aa):
(4)              r"""
(4)              Low-level wrapper for `::fz_render_glyph_pixmap()`.
(4)              Create a pixmap containing a rendered glyph.
(4)              Lookup gid from font, clip it with scissor, and rendering it
(4)              with aa bits of antialiasing into a new pixmap.
(4)              The caller takes ownership of the pixmap and so must free it.
(4)              Note: This function is no longer used for normal rendering
(4)              operations, and is kept around just because we use it in the
(4)              app. It should be considered "at risk" of removal from the API.
(4)              """
(4)              return _mupdf.ll_fz_render_glyph_pixmap(font, gid, ctm, scissor, aa)
(0)          def ll_fz_render_t3_glyph_direct(dev, font, gid, trm, gstate, def_cs):
(4)              r"""
(4)              Low-level wrapper for `::fz_render_t3_glyph_direct()`.
(4)              Nasty PDF interpreter specific hernia, required to allow the
(4)              interpreter to replay glyphs from a type3 font directly into
```

```
10821: (4)              the target device.
10822: (4)              This is only used in exceptional circumstances (such as type3
10823: (4)              glyphs that inherit current graphics state, or nested type3
10824: (4)              glyphs).
10825: (4)              """
10826: (4)              return _mupdf.ll_fz_render_t3_glyph_direct(dev, font, gid, trm, gstate,
def_cs)
10827: (0)          def ll_fz_rendering_intent_name(ri):
10828: (4)              r"""
10829: (4)              Low-level wrapper for `::fz_rendering_intent_name()`.
10830: (4)              Map from enumerated rendering intent to string.
10831: (4)              The returned string is static and therefore must not be freed.
10832: (4)              """
10833: (4)              return _mupdf.ll_fz_rendering_intent_name(ri)
10834: (0)          def ll_fz_report_error():
10835: (4)              r""" Low-level wrapper for `::fz_report_error()`."""
10836: (4)              return _mupdf.ll_fz_report_error()
10837: (0)          def ll_fz_reset_output(arg_1):
10838: (4)              r"""
10839: (4)              Low-level wrapper for `::fz_reset_output()`.
10840: (4)              Reset a closed output stream. Returns state to
10841: (4)              (broadly) that which it was in when opened. Not
10842: (4)              all outputs can be reset, so this may throw an
10843: (4)              exception.
10844: (4)              """
10845: (4)              return _mupdf.ll_fz_reset_output(arg_1)
10846: (0)          def ll_fz_reset_story(story):
10847: (4)              r""" Low-level wrapper for `::fz_reset_story()`."""
10848: (4)              return _mupdf.ll_fz_reset_story(story)
10849: (0)          def ll_fz_resize_buffer(buf, capacity):
10850: (4)              r"""
10851: (4)              Low-level wrapper for `::fz_resize_buffer()`.
10852: (4)              Ensure that a buffer has a given capacity,
10853: (4)              truncating data if required.
10854: (4)              capacity: The desired capacity for the buffer. If the current
10855: (4)              size of the buffer contents is smaller than capacity, it is
10856: (4)              truncated.
10857: (4)              """
10858: (4)              return _mupdf.ll_fz_resize_buffer(buf, capacity)
10859: (0)          def ll_fz_resolve_link(doc, uri, xp, yp):
10860: (4)              r"""
10861: (5)               Low-level wrapper for `::fz_resolve_link()`.
10862: (4)              This function has out-params. Python/C# wrappers look like:
10863: (8)                  `ll_fz_resolve_link(::fz_document *doc, const char *uri)` =>
`(fz_location, float xp, float yp)`
10864: (8)                  Resolve an internal link to a page number.
10865: (8)                  xp, yp: Pointer to store coordinate of destination on the page.
10866: (8)                  Returns (-1,-1) if the URI cannot be resolved.
10867: (4)              """
10868: (4)              return _mupdf.ll_fz_resolve_link(doc, uri, xp, yp)
10869: (0)          def ll_fz_resolve_link_dest(doc, uri):
10870: (4)              r"""
10871: (4)              Low-level wrapper for `::fz_resolve_link_dest()`.
10872: (4)              Resolve an internal link to a page number, location, and possible viewing
parameters.
10873: (4)              Returns location (-1,-1) if the URI cannot be resolved.
10874: (4)              """
10875: (4)              return _mupdf.ll_fz_resolve_link_dest(doc, uri)
10876: (0)          def ll_fz_rethrow():
10877: (4)              r""" Low-level wrapper for `::fz_rethrow()`."""
10878: (4)              return _mupdf.ll_fz_rethrow()
10879: (0)          def ll_fz_rethrow_if(errcode):
10880: (4)              r"""
10881: (4)              Low-level wrapper for `::fz_rethrow_if()`.
10882: (4)              Within an fz_catch() block, rethrow the current exception
10883: (4)              if the errcode of the current exception matches.
10884: (4)              This assumes no intervening use of fz_try/fz_catch.
10885: (4)              """
10886: (4)              return _mupdf.ll_fz_rethrow_if(errcode)
```

```
(0)         def ll_fz_rethrow_unless(errcode):
(4)             r""" Low-level wrapper for `::fz_rethrow_unless()`."""
(4)             return _mupdf.ll_fz_rethrow_unless(errcode)
(0)         def ll_fz_rotate(degrees):
(4)             r"""
(4)             Low-level wrapper for `::fz_rotate()`.
(4)             Create a rotation matrix.
(4)             The returned matrix is of the form
(4)             [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
(4)             m: Pointer to place to store matrix
(4)             degrees: Degrees of counter clockwise rotation. Values less
(4)             than zero and greater than 360 are handled as expected.
(4)             Returns m.
(4)             """
(4)             return _mupdf.ll_fz_rotate(degrees)
(0)         def ll_fz_round_rect(rect):
(4)             r"""
(4)             Low-level wrapper for `::fz_round_rect()`.
(4)             Round rectangle coordinates.
(4)             Coordinates in a bounding box are integers, so rounding of the
(4)             rects coordinates takes place. The top left corner is rounded
(4)             upwards and left while the bottom right corner is rounded
(4)             downwards and to the right.
(4)             This differs from fz_irect_from_rect, in that fz_irect_from_rect
(4)             slavishly follows the numbers (i.e any slight over/under
(4)             calculations can cause whole extra pixels to be added).
(4)             fz_round_rect allows for a small amount of rounding error when
(4)             calculating the bbox.
(4)             """
(4)             return _mupdf.ll_fz_round_rect(rect)
(0)         def ll_fz_run_display_list(list, dev, ctm, scissor, cookie):
(4)             r"""
(4)             Low-level wrapper for `::fz_run_display_list()`.
(4)             (Re)-run a display list through a device.
(4)             list: A display list, created by fz_new_display_list and
(4)             populated with objects from a page by running fz_run_page on a
(4)             device obtained from fz_new_list_device.
(4)             ctm: Transform to apply to display list contents. May include
(4)             for example scaling and rotation, see fz_scale, fz_rotate and
(4)             fz_concat. Set to fz_identity if no transformation is desired.
(4)             scissor: Only the part of the contents of the display list
(4)             visible within this area will be considered when the list is
(4)             run through the device. This does not imply for tile objects
(4)             contained in the display list.
(4)             cookie: Communication mechanism between caller and library
(4)             running the page. Intended for multi-threaded applications,
(4)             while single-threaded applications set cookie to NULL. The
(4)             caller may abort an ongoing page run. Cookie also communicates
(4)             progress information back to the caller. The fields inside
(4)             cookie are continually updated while the page is being run.
(4)             """
(4)             return _mupdf.ll_fz_run_display_list(list, dev, ctm, scissor, cookie)
(0)         def ll_fz_run_document_structure(doc, dev, cookie):
(4)             r"""
(4)             Low-level wrapper for `::fz_run_document_structure()`.
(4)             Run the document structure through a device.
(4)             doc: Document in question.
(4)             dev: Device obtained from fz_new_*_device.
(4)             cookie: Communication mechanism between caller and library.
(4)             Intended for multi-threaded applications, while
(4)             single-threaded applications set cookie to NULL. The
(4)             caller may abort an ongoing rendering of a page. Cookie also
(4)             communicates progress information back to the caller. The
(4)             fields inside cookie are continually updated while the page is
(4)             rendering.
(4)             """
(4)             return _mupdf.ll_fz_run_document_structure(doc, dev, cookie)
(0)         def ll_fz_run_page(page, dev, transform, cookie):
(4)             r"""
```

```
10956: (4)             Low-level wrapper for `::fz_run_page()`.
10957: (4)             Run a page through a device.
10958: (4)             page: Page obtained from fz_load_page.
10959: (4)             dev: Device obtained from fz_new_*_device.
10960: (4)             transform: Transform to apply to page. May include for example
10961: (4)             scaling and rotation, see fz_scale, fz_rotate and fz_concat.
10962: (4)             Set to fz_identity if no transformation is desired.
10963: (4)             cookie: Communication mechanism between caller and library
10964: (4)             rendering the page. Intended for multi-threaded applications,
10965: (4)             while single-threaded applications set cookie to NULL. The
10966: (4)             caller may abort an ongoing rendering of a page. Cookie also
10967: (4)             communicates progress information back to the caller. The
10968: (4)             fields inside cookie are continually updated while the page is
10969: (4)             rendering.
10970: (4)             """
10971: (4)             return _mupdf.ll_fz_run_page(page, dev, transform, cookie)
10972: (0)         def ll_fz_run_page_annots(page, dev, transform, cookie):
10973: (4)             r"""
10974: (4)             Low-level wrapper for `::fz_run_page_annots()`.
10975: (4)             Run the annotations on a page through a device.
10976: (4)             """
10977: (4)             return _mupdf.ll_fz_run_page_annots(page, dev, transform, cookie)
10978: (0)         def ll_fz_run_page_contents(page, dev, transform, cookie):
10979: (4)             r"""
10980: (4)             Low-level wrapper for `::fz_run_page_contents()`.
10981: (4)             Run a page through a device. Just the main
10982: (4)             page content, without the annotations, if any.
10983: (4)             page: Page obtained from fz_load_page.
10984: (4)             dev: Device obtained from fz_new_*_device.
10985: (4)             transform: Transform to apply to page. May include for example
10986: (4)             scaling and rotation, see fz_scale, fz_rotate and fz_concat.
10987: (4)             Set to fz_identity if no transformation is desired.
10988: (4)             cookie: Communication mechanism between caller and library
10989: (4)             rendering the page. Intended for multi-threaded applications,
10990: (4)             while single-threaded applications set cookie to NULL. The
10991: (4)             caller may abort an ongoing rendering of a page. Cookie also
10992: (4)             communicates progress information back to the caller. The
10993: (4)             fields inside cookie are continually updated while the page is
10994: (4)             rendering.
10995: (4)             """
10996: (4)             return _mupdf.ll_fz_run_page_contents(page, dev, transform, cookie)
10997: (0)         def ll_fz_run_page_widgets(page, dev, transform, cookie):
10998: (4)             r"""
10999: (4)             Low-level wrapper for `::fz_run_page_widgets()`.
11000: (4)             Run the widgets on a page through a device.
11001: (4)             """
11002: (4)             return _mupdf.ll_fz_run_page_widgets(page, dev, transform, cookie)
11003: (0)         def ll_fz_run_t3_glyph(font, gid, trm, dev):
11004: (4)             r"""
11005: (4)             Low-level wrapper for `::fz_run_t3_glyph()`.
11006: (4)             Run a glyph from a Type3 font to
11007: (4)             a given device.
11008: (4)             font: The font to find the glyph in.
11009: (4)             gid: The glyph to run.
11010: (4)             trm: The transform to apply.
11011: (4)             dev: The device to render onto.
11012: (4)             """
11013: (4)             return _mupdf.ll_fz_run_t3_glyph(font, gid, trm, dev)
11014: (0)         def ll_fz_runeidx(str, p):
11015: (4)             r"""
11016: (4)             Low-level wrapper for `::fz_runeidx()`.
11017: (4)             Compute the index of a rune in a string.
11018: (4)             str: Pointer to beginning of a string.
11019: (4)             p: Pointer to a char in str.
11020: (4)             Returns the index of the rune pointed to by p in str.
11021: (4)             """
11022: (4)             return _mupdf.ll_fz_runeidx(str, p)
11023: (0)         def ll_fz_runelen(rune):
11024: (4)             r"""
```

```
(4)             Low-level wrapper for `::fz_runelen()`.
(4)             Count how many chars are required to represent a rune.
(4)             rune: The rune to encode.
(4)             Returns the number of bytes required to represent this run in
(4)             UTF8.
(4)             """
(4)             return _mupdf.ll_fz_runelen(rune)
(0)         def ll_fz_runeptr(str, idx):
(4)             r"""
(4)             Low-level wrapper for `::fz_runeptr()`.
(4)             Obtain a pointer to the char representing the rune
(4)             at a given index.
(4)             str: Pointer to beginning of a string.
(4)             idx: Index of a rune to return a char pointer to.
(4)             Returns a pointer to the char where the desired rune starts,
(4)             or NULL if the string ends before the index is reached.
(4)             """
(4)             return _mupdf.ll_fz_runeptr(str, idx)
(0)         def ll_fz_runetochar(str, rune):
(4)             r"""
(4)             Low-level wrapper for `::fz_runetochar()`.
(4)             UTF8 encode a rune to a sequence of chars.
(4)             str: Pointer to a place to put the UTF8 encoded character.
(4)             rune: Pointer to a 'rune'.
(4)             Returns the number of bytes the rune took to output.
(4)             """
(4)             return _mupdf.ll_fz_runetochar(str, rune)
(0)         def ll_fz_samples_get(pixmap, offset):
(4)             r"""
(5)              Low-level wrapper for `::fz_samples_get()`.
(4)             Provides simple (but slow) access to pixmap data from Python and C#.
(4)             """
(4)             return _mupdf.ll_fz_samples_get(pixmap, offset)
(0)         def ll_fz_samples_set(pixmap, offset, value):
(4)             r"""
(5)              Low-level wrapper for `::fz_samples_set()`.
(4)             Provides simple (but slow) write access to pixmap data from Python and
(4)             C#.
(4)             """
(4)             return _mupdf.ll_fz_samples_set(pixmap, offset, value)
(0)         def ll_fz_save_accelerator(doc, accel):
(4)             r"""
(4)             Low-level wrapper for `::fz_save_accelerator()`.
(4)             Save accelerator data for the document to a given file.
(4)             """
(4)             return _mupdf.ll_fz_save_accelerator(doc, accel)
(0)         def ll_fz_save_bitmap_as_pbm(bitmap, filename):
(4)             r"""
(4)             Low-level wrapper for `::fz_save_bitmap_as_pbm()`.
(4)             Save a bitmap as a pbm.
(4)             """
(4)             return _mupdf.ll_fz_save_bitmap_as_pbm(bitmap, filename)
(0)         def ll_fz_save_bitmap_as_pcl(bitmap, filename, append, pcl):
(4)             r"""
(4)             Low-level wrapper for `::fz_save_bitmap_as_pcl()`.
(4)             Save a bitmap as mono PCL.
(4)             """
(4)             return _mupdf.ll_fz_save_bitmap_as_pcl(bitmap, filename, append, pcl)
(0)         def ll_fz_save_bitmap_as_pkm(bitmap, filename):
(4)             r"""
(4)             Low-level wrapper for `::fz_save_bitmap_as_pkm()`.
(4)             Save a CMYK bitmap as a pkm.
(4)             """
(4)             return _mupdf.ll_fz_save_bitmap_as_pkm(bitmap, filename)
(0)         def ll_fz_save_bitmap_as_pwg(bitmap, filename, append, pwg):
(4)             r"""
(4)             Low-level wrapper for `::fz_save_bitmap_as_pwg()`.
(4)             Save a bitmap as a PWG.
(4)             """
```

```
11094: (4)             return _mupdf.ll_fz_save_bitmap_as_pwg(bitmap, filename, append, pwg)
11095: (0)         def ll_fz_save_buffer(buf, filename):
11096: (4)             r"""
11097: (4)             Low-level wrapper for `::fz_save_buffer()`.
11098: (4)             Save the contents of a buffer to a file.
11099: (4)             """
11100: (4)             return _mupdf.ll_fz_save_buffer(buf, filename)
11101: (0)         def ll_fz_save_pixmap_as_jpeg(pixmap, filename, quality):
11102: (4)             r"""
11103: (4)             Low-level wrapper for `::fz_save_pixmap_as_jpeg()`.
11104: (4)             Save a pixmap as a JPEG.
11105: (4)             """
11106: (4)             return _mupdf.ll_fz_save_pixmap_as_jpeg(pixmap, filename, quality)
11107: (0)         def ll_fz_save_pixmap_as_jpx(pixmap, filename, q):
11108: (4)             r"""
11109: (4)             Low-level wrapper for `::fz_save_pixmap_as_jpx()`.
11110: (4)             Save pixmap data as JP2K with no subsampling.
11111: (4)             quality = 100 = lossless
11112: (4)             otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
11113: (4)             """
11114: (4)             return _mupdf.ll_fz_save_pixmap_as_jpx(pixmap, filename, q)
11115: (0)         def ll_fz_save_pixmap_as_pam(pixmap, filename):
11116: (4)             r"""
11117: (4)             Low-level wrapper for `::fz_save_pixmap_as_pam()`.
11118: (4)             Save a pixmap as a pnm (greyscale, rgb or cmyk, with or without
11119: (4)             alpha).
11120: (4)             """
11121: (4)             return _mupdf.ll_fz_save_pixmap_as_pam(pixmap, filename)
11122: (0)         def ll_fz_save_pixmap_as_pbm(pixmap, filename):
11123: (4)             r"""
11124: (4)             Low-level wrapper for `::fz_save_pixmap_as_pbm()`.
11125: (4)             Save a pixmap as a pbm. (Performing halftoning).
11126: (4)             """
11127: (4)             return _mupdf.ll_fz_save_pixmap_as_pbm(pixmap, filename)
11128: (0)         def ll_fz_save_pixmap_as_pcl(pixmap, filename, append, pcl):
11129: (4)             r"""
11130: (4)             Low-level wrapper for `::fz_save_pixmap_as_pcl()`.
11131: (4)             Save an (RGB) pixmap as color PCL.
11132: (4)             """
11133: (4)             return _mupdf.ll_fz_save_pixmap_as_pcl(pixmap, filename, append, pcl)
11134: (0)         def ll_fz_save_pixmap_as_pclm(pixmap, filename, append, options):
11135: (4)             r"""
11136: (4)             Low-level wrapper for `::fz_save_pixmap_as_pclm()`.
11137: (4)             Save a (Greyscale or RGB) pixmap as pclm.
11138: (4)             """
11139: (4)             return _mupdf.ll_fz_save_pixmap_as_pclm(pixmap, filename, append, options)
11140: (0)         def ll_fz_save_pixmap_as_pdfocr(pixmap, filename, append, options):
11141: (4)             r"""
11142: (4)             Low-level wrapper for `::fz_save_pixmap_as_pdfocr()`.
11143: (4)             Save a (Greyscale or RGB) pixmap as pdfocr.
11144: (4)             """
11145: (4)             return _mupdf.ll_fz_save_pixmap_as_pdfocr(pixmap, filename, append,
options)
11146: (0)         def ll_fz_save_pixmap_as_pkm(pixmap, filename):
11147: (4)             r"""
11148: (4)             Low-level wrapper for `::fz_save_pixmap_as_pkm()`.
11149: (4)             Save a CMYK pixmap as a pkm. (Performing halftoning).
11150: (4)             """
11151: (4)             return _mupdf.ll_fz_save_pixmap_as_pkm(pixmap, filename)
11152: (0)         def ll_fz_save_pixmap_as_png(pixmap, filename):
11153: (4)             r"""
11154: (4)             Low-level wrapper for `::fz_save_pixmap_as_png()`.
11155: (4)             Save a (Greyscale or RGB) pixmap as a png.
11156: (4)             """
11157: (4)             return _mupdf.ll_fz_save_pixmap_as_png(pixmap, filename)
11158: (0)         def ll_fz_save_pixmap_as_pnm(pixmap, filename):
11159: (4)             r"""
11160: (4)             Low-level wrapper for `::fz_save_pixmap_as_pnm()`.
```

```
11161: (4)              Save a pixmap as a pnm (greyscale or rgb, no alpha).
11162: (4)              """
11163: (4)              return _mupdf.ll_fz_save_pixmap_as_pnm(pixmap, filename)
11164: (0)          def ll_fz_save_pixmap_as_ps(pixmap, filename, append):
11165: (4)              r"""
11166: (4)              Low-level wrapper for `::fz_save_pixmap_as_ps()`.
11167: (4)              Save a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
11168: (4)              """
11169: (4)              return _mupdf.ll_fz_save_pixmap_as_ps(pixmap, filename, append)
11170: (0)          def ll_fz_save_pixmap_as_psd(pixmap, filename):
11171: (4)              r"""
11172: (4)              Low-level wrapper for `::fz_save_pixmap_as_psd()`.
11173: (4)              Save a pixmap as a PSD file.
11174: (4)              """
11175: (4)              return _mupdf.ll_fz_save_pixmap_as_psd(pixmap, filename)
11176: (0)          def ll_fz_save_pixmap_as_pwg(pixmap, filename, append, pwg):
11177: (4)              r"""
11178: (4)              Low-level wrapper for `::fz_save_pixmap_as_pwg()`.
11179: (4)              Save a pixmap as a PWG.
11180: (4)              """
11181: (4)              return _mupdf.ll_fz_save_pixmap_as_pwg(pixmap, filename, append, pwg)
11182: (0)          def ll_fz_scale(sx, sy):
11183: (4)              r"""
11184: (4)              Low-level wrapper for `::fz_scale()`.
11185: (4)              Create a scaling matrix.
11186: (4)              The returned matrix is of the form [ sx 0 0 sy 0 0 ].
11187: (4)              m: Pointer to the matrix to populate
11188: (4)              sx, sy: Scaling factors along the X- and Y-axes. A scaling
11189: (4)              factor of 1.0 will not cause any scaling along the relevant
11190: (4)              axis.
11191: (4)              Returns m.
11192: (4)              """
11193: (4)              return _mupdf.ll_fz_scale(sx, sy)
11194: (0)          def ll_fz_scale_pixmap(src, x, y, w, h, clip):
11195: (4)              r""" Low-level wrapper for `::fz_scale_pixmap()`."""
11196: (4)              return _mupdf.ll_fz_scale_pixmap(src, x, y, w, h, clip)
11197: (0)          def ll_fz_search_chapter_page_number(doc, chapter, page, needle, hit_mark,
hit_bbox, hit_max):
11198: (4)              r"""
11199: (5)               Low-level wrapper for `::fz_search_chapter_page_number()`.
11200: (4)              This function has out-params. Python/C# wrappers look like:
11201: (8)                  `ll_fz_search_chapter_page_number(::fz_document *doc, int chapter, int
page, const char *needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
11202: (4)              """
11203: (4)              return _mupdf.ll_fz_search_chapter_page_number(doc, chapter, page, needle,
hit_mark, hit_bbox, hit_max)
11204: (0)          def ll_fz_search_display_list(list, needle, hit_mark, hit_bbox, hit_max):
11205: (4)              r"""
11206: (5)               Low-level wrapper for `::fz_search_display_list()`.
11207: (4)              This function has out-params. Python/C# wrappers look like:
11208: (8)                  `ll_fz_search_display_list(::fz_display_list *list, const char
*needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
11209: (4)              """
11210: (4)              return _mupdf.ll_fz_search_display_list(list, needle, hit_mark, hit_bbox,
hit_max)
11211: (0)          def ll_fz_search_page(page, needle, hit_mark, hit_bbox, hit_max):
11212: (4)              r"""
11213: (5)               Low-level wrapper for `::fz_search_page()`.
11214: (4)              This function has out-params. Python/C# wrappers look like:
11215: (8)                  `ll_fz_search_page(::fz_page *page, const char *needle, ::fz_quad
*hit_bbox, int hit_max)` => `(int, int hit_mark)`
11216: (8)                  Search for the 'needle' text on the page.
11217: (8)                  Record the hits in the hit_bbox array and return the number of
11218: (8)                  hits. Will stop looking once it has filled hit_max rectangles.
11219: (4)              """
11220: (4)              return _mupdf.ll_fz_search_page(page, needle, hit_mark, hit_bbox, hit_max)
11221: (0)          def ll_fz_search_page2(doc, number, needle, hit_max):
11222: (4)              r"""
11223: (5)               Low-level wrapper for `::fz_search_page2()`.
```

```
11224: (4)              C++ alternative to fz_search_page() that returns information in a
std::vector.
11225: (4)              """
11226: (4)              return _mupdf.ll_fz_search_page2(doc, number, needle, hit_max)
11227: (0)          def ll_fz_search_page_number(doc, number, needle, hit_mark, hit_bbox,
hit_max):
11228: (4)              r"""
11229: (5)               Low-level wrapper for `::fz_search_page_number()`.
11230: (4)              This function has out-params. Python/C# wrappers look like:
11231: (8)                  `ll_fz_search_page_number(::fz_document *doc, int number, const char
*needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
11232: (4)              """
11233: (4)              return _mupdf.ll_fz_search_page_number(doc, number, needle, hit_mark,
hit_bbox, hit_max)
11234: (0)          def ll_fz_search_stext_page(text, needle, hit_mark, hit_bbox, hit_max):
11235: (4)              r"""
11236: (5)               Low-level wrapper for `::fz_search_stext_page()`.
11237: (4)              This function has out-params. Python/C# wrappers look like:
11238: (8)                  `ll_fz_search_stext_page(::fz_stext_page *text, const char *needle,
::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
11239: (8)                  Search for occurrence of 'needle' in text page.
11240: (8)                  Return the number of hits and store hit quads in the passed in
11241: (8)                  array.
11242: (8)                  NOTE: This is an experimental interface and subject to change
11243: (8)                  without notice.
11244: (4)              """
11245: (4)              return _mupdf.ll_fz_search_stext_page(text, needle, hit_mark, hit_bbox,
hit_max)
11246: (0)          def ll_fz_seek(stm, offset, whence):
11247: (4)              r"""
11248: (4)              Low-level wrapper for `::fz_seek()`.
11249: (4)              Seek within a stream.
11250: (4)              stm: The stream to seek within.
11251: (4)              offset: The offset to seek to.
11252: (4)              whence: From where the offset is measured (see fseek).
11253: (4)              SEEK_SET - start of stream.
11254: (4)              SEEK_CUR - current position.
11255: (4)              SEEK_END - end of stream.
11256: (4)              """
11257: (4)              return _mupdf.ll_fz_seek(stm, offset, whence)
11258: (0)          def ll_fz_seek_output(out, off, whence):
11259: (4)              r"""
11260: (4)              Low-level wrapper for `::fz_seek_output()`.
11261: (4)              Seek to the specified file position.
11262: (4)              See fseek for arguments.
11263: (4)              Throw an error on unseekable outputs.
11264: (4)              """
11265: (4)              return _mupdf.ll_fz_seek_output(out, off, whence)
11266: (0)          def ll_fz_separation_current_behavior(sep, separation):
11267: (4)              r"""
11268: (4)              Low-level wrapper for `::fz_separation_current_behavior()`.
11269: (4)              Test for the current behavior of a separation.
11270: (4)              """
11271: (4)              return _mupdf.ll_fz_separation_current_behavior(sep, separation)
11272: (0)          def ll_fz_separation_equivalent(seps, idx, dst_cs, dst_color, prf,
color_params):
11273: (4)              r"""
11274: (5)               Low-level wrapper for `::fz_separation_equivalent()`.
11275: (4)              This function has out-params. Python/C# wrappers look like:
11276: (8)                  `ll_fz_separation_equivalent(const ::fz_separations *seps, int idx,
::fz_colorspace *dst_cs, ::fz_colorspace *prf, ::fz_color_params color_params)` => float dst_color
11277: (8)                  Get the equivalent separation color in a given colorspace.
11278: (4)              """
11279: (4)              return _mupdf.ll_fz_separation_equivalent(seps, idx, dst_cs, dst_color,
prf, color_params)
11280: (0)          def ll_fz_separation_name(sep, separation):
11281: (4)              r""" Low-level wrapper for `::fz_separation_name()`."""
11282: (4)              return _mupdf.ll_fz_separation_name(sep, separation)
11283: (0)          def ll_fz_set_aa_level(bits):
```

```
11284: (4)             r"""
11285: (4)             Low-level wrapper for `::fz_set_aa_level()`.
11286: (4)             Set the number of bits of antialiasing we should
11287: (4)             use (for both text and graphics).
11288: (4)             bits: The number of bits of antialiasing to use (values are
11289: (4)             clamped to within the 0 to 8 range).
11290: (4)             """
11291: (4)             return _mupdf.ll_fz_set_aa_level(bits)
11292: (0)         def ll_fz_set_compressed_image_buffer(cimg, buf):
11293: (4)             r""" Low-level wrapper for `::fz_set_compressed_image_buffer()`."""
11294: (4)             return _mupdf.ll_fz_set_compressed_image_buffer(cimg, buf)
11295: (0)         def ll_fz_set_default_cmyk(default_cs, cs):
11296: (4)             r""" Low-level wrapper for `::fz_set_default_cmyk()`."""
11297: (4)             return _mupdf.ll_fz_set_default_cmyk(default_cs, cs)
11298: (0)         def ll_fz_set_default_colorspaces(dev, default_cs):
11299: (4)             r""" Low-level wrapper for `::fz_set_default_colorspaces()`."""
11300: (4)             return _mupdf.ll_fz_set_default_colorspaces(dev, default_cs)
11301: (0)         def ll_fz_set_default_gray(default_cs, cs):
11302: (4)             r"""
11303: (4)             Low-level wrapper for `::fz_set_default_gray()`.
11304: (4)             Set new defaults within the default colorspace structure.
11305: (4)             New references are taken to the new default, and references to
11306: (4)             the old defaults dropped.
11307: (4)             Never throws exceptions.
11308: (4)             """
11309: (4)             return _mupdf.ll_fz_set_default_gray(default_cs, cs)
11310: (0)         def ll_fz_set_default_output_intent(default_cs, cs):
11311: (4)             r""" Low-level wrapper for `::fz_set_default_output_intent()`."""
11312: (4)             return _mupdf.ll_fz_set_default_output_intent(default_cs, cs)
11313: (0)         def ll_fz_set_default_rgb(default_cs, cs):
11314: (4)             r""" Low-level wrapper for `::fz_set_default_rgb()`."""
11315: (4)             return _mupdf.ll_fz_set_default_rgb(default_cs, cs)
11316: (0)         def ll_fz_set_error_callback(error_cb, user):
11317: (4)             r"""
11318: (4)             Low-level wrapper for `::fz_set_error_callback()`.
11319: (4)             Set the error callback. This will be called as part of the
11320: (4)             exception handling.
11321: (4)             The callback must not throw exceptions!
11322: (4)             """
11323: (4)             return _mupdf.ll_fz_set_error_callback(error_cb, user)
11324: (0)         def ll_fz_set_font_bbox(font, xmin, ymin, xmax, ymax):
11325: (4)             r"""
11326: (4)             Low-level wrapper for `::fz_set_font_bbox()`.
11327: (4)             Set the font bbox.
11328: (4)             font: The font to set the bbox for.
11329: (4)             xmin, ymin, xmax, ymax: The bounding box.
11330: (4)             """
11331: (4)             return _mupdf.ll_fz_set_font_bbox(font, xmin, ymin, xmax, ymax)
11332: (0)         def ll_fz_set_font_embedding(font, embed):
11333: (4)             r"""
11334: (4)             Low-level wrapper for `::fz_set_font_embedding()`.
11335: (4)             Control whether a given font should be embedded or not when writing.
11336: (4)             """
11337: (4)             return _mupdf.ll_fz_set_font_embedding(font, embed)
11338: (0)         def ll_fz_set_graphics_aa_level(bits):
11339: (4)             r"""
11340: (4)             Low-level wrapper for `::fz_set_graphics_aa_level()`.
11341: (4)             Set the number of bits of antialiasing we
11342: (4)             should use for graphics.
11343: (4)             bits: The number of bits of antialiasing to use (values are
11344: (4)             clamped to within the 0 to 8 range).
11345: (4)             """
11346: (4)             return _mupdf.ll_fz_set_graphics_aa_level(bits)
11347: (0)         def ll_fz_set_graphics_min_line_width(min_line_width):
11348: (4)             r"""
11349: (4)             Low-level wrapper for `::fz_set_graphics_min_line_width()`.
11350: (4)             Set the minimum line width to be
11351: (4)             used for stroked lines.
11352: (4)             min_line_width: The minimum line width to use (in pixels).
```

```
(4)             """
(4)             return _mupdf.ll_fz_set_graphics_min_line_width(min_line_width)
(0)         def ll_fz_set_link_rect(link, rect):
(4)             r""" Low-level wrapper for `::fz_set_link_rect()`."""
(4)             return _mupdf.ll_fz_set_link_rect(link, rect)
(0)         def ll_fz_set_link_uri(link, uri):
(4)             r""" Low-level wrapper for `::fz_set_link_uri()`."""
(4)             return _mupdf.ll_fz_set_link_uri(link, uri)
(0)         def ll_fz_set_metadata(doc, key, value):
(4)             r""" Low-level wrapper for `::fz_set_metadata()`."""
(4)             return _mupdf.ll_fz_set_metadata(doc, key, value)
(0)         def ll_fz_set_pixmap_image_tile(cimg, pix):
(4)             r""" Low-level wrapper for `::fz_set_pixmap_image_tile()`."""
(4)             return _mupdf.ll_fz_set_pixmap_image_tile(cimg, pix)
(0)         def ll_fz_set_pixmap_resolution(pix, xres, yres):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_pixmap_resolution()`.
(4)             Set the pixels per inch resolution of the pixmap.
(4)             """
(4)             return _mupdf.ll_fz_set_pixmap_resolution(pix, xres, yres)
(0)         def ll_fz_set_separation_behavior(sep, separation, behavior):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_separation_behavior()`.
(4)             Control the rendering of a given separation.
(4)             """
(4)             return _mupdf.ll_fz_set_separation_behavior(sep, separation, behavior)
(0)         def ll_fz_set_stddbg(out):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_stddbg()`.
(4)             Set the output stream to be used for fz_stddbg. Set to NULL to
(4)             reset to default (stderr).
(4)             """
(4)             return _mupdf.ll_fz_set_stddbg(out)
(0)         def ll_fz_set_text_aa_level(bits):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_text_aa_level()`.
(4)             Set the number of bits of antialiasing we
(4)             should use for text.
(4)             bits: The number of bits of antialiasing to use (values are
(4)             clamped to within the 0 to 8 range).
(4)             """
(4)             return _mupdf.ll_fz_set_text_aa_level(bits)
(0)         def ll_fz_set_use_document_css(use):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_use_document_css()`.
(4)             Toggle whether to respect document styles in HTML and EPUB.
(4)             """
(4)             return _mupdf.ll_fz_set_use_document_css(use)
(0)         def ll_fz_set_user_context(user):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_user_context()`.
(4)             Set the user field in the context.
(4)             NULL initially, this field can be set to any opaque value
(4)             required by the user. It is copied on clones.
(4)             """
(4)             return _mupdf.ll_fz_set_user_context(user)
(0)         def ll_fz_set_user_css(text):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_user_css()`.
(4)             Set the user stylesheet source text for use with HTML and EPUB.
(4)             """
(4)             return _mupdf.ll_fz_set_user_css(text)
(0)         def ll_fz_set_warning_callback(warning_cb, user):
(4)             r"""
(4)             Low-level wrapper for `::fz_set_warning_callback()`.
(4)             Set the warning callback. This will be called as part of the
(4)             exception handling.
(4)             The callback must not throw exceptions!
(4)             """
```

```
(4)             return _mupdf.ll_fz_set_warning_callback(warning_cb, user)
(0)         def ll_fz_sha256_final(state, digest):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha256_final()`.
(4)             MD5 finalization. Ends an MD5 message-digest operation, writing
(4)             the message digest and zeroizing the context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha256_final(state, digest)
(0)         def ll_fz_sha256_init(state):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha256_init()`.
(4)             SHA256 initialization. Begins an SHA256 operation, initialising
(4)             the supplied context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha256_init(state)
(0)         def ll_fz_sha256_update(state, input, inlen):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha256_update()`.
(4)             SHA256 block update operation. Continues an SHA256 message-
(4)             digest operation, processing another message block, and updating
(4)             the context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha256_update(state, input, inlen)
(0)         def ll_fz_sha384_final(state, digest):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha384_final()`.
(4)             SHA384 finalization. Ends an SHA384 message-digest operation,
(4)             writing the message digest and zeroizing the context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha384_final(state, digest)
(0)         def ll_fz_sha384_init(state):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha384_init()`.
(4)             SHA384 initialization. Begins an SHA384 operation, initialising
(4)             the supplied context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha384_init(state)
(0)         def ll_fz_sha384_update(state, input, inlen):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha384_update()`.
(4)             SHA384 block update operation. Continues an SHA384 message-
(4)             digest operation, processing another message block, and updating
(4)             the context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha384_update(state, input, inlen)
(0)         def ll_fz_sha512_final(state, digest):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha512_final()`.
(4)             SHA512 finalization. Ends an SHA512 message-digest operation,
(4)             writing the message digest and zeroizing the context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha512_final(state, digest)
(0)         def ll_fz_sha512_init(state):
(4)             r"""
(4)             Low-level wrapper for `::fz_sha512_init()`.
(4)             SHA512 initialization. Begins an SHA512 operation, initialising
(4)             the supplied context.
(4)             Never throws an exception.
(4)             """
(4)             return _mupdf.ll_fz_sha512_init(state)
(0)         def ll_fz_sha512_update(state, input, inlen):
(4)             r"""
```

```
11491: (4)              Low-level wrapper for `::fz_sha512_update()`.
11492: (4)              SHA512 block update operation. Continues an SHA512 message-
11493: (4)              digest operation, processing another message block, and updating
11494: (4)              the context.
11495: (4)              Never throws an exception.
11496: (4)              """
11497: (4)              return _mupdf.ll_fz_sha512_update(state, input, inlen)
11498: (0)          def ll_fz_shear(sx, sy):
11499: (4)              r"""
11500: (4)              Low-level wrapper for `::fz_shear()`.
11501: (4)              Create a shearing matrix.
11502: (4)              The returned matrix is of the form [ 1 sy sx 1 0 0 ].
11503: (4)              m: pointer to place to store returned matrix
11504: (4)              sx, sy: Shearing factors. A shearing factor of 0.0 will not
11505: (4)              cause any shearing along the relevant axis.
11506: (4)              Returns m.
11507: (4)              """
11508: (4)              return _mupdf.ll_fz_shear(sx, sy)
11509: (0)          def ll_fz_show_glyph(text, font, trm, glyph, unicode, wmode, bidi_level,
markup_dir, language):
11510: (4)              r"""
11511: (4)              Low-level wrapper for `::fz_show_glyph()`.
11512: (4)              Add a glyph/unicode value to a text object.
11513: (4)              text: Text object to add to.
11514: (4)              font: The font the glyph should be added in.
11515: (4)              trm: The transform to use for the glyph.
11516: (4)              glyph: The glyph id to add.
11517: (4)              unicode: The unicode character for the glyph.
11518: (4)              cid: The CJK CID value or raw character code.
11519: (4)              wmode: 1 for vertical mode, 0 for horizontal.
11520: (4)              bidi_level: The bidirectional level for this glyph.
11521: (4)              markup_dir: The direction of the text as specified in the
11522: (4)              markup.
11523: (4)              language: The language in use (if known, 0 otherwise)
11524: (4)              (e.g. FZ_LANG_zh_Hans).
11525: (4)              Throws exception on failure to allocate.
11526: (4)              """
11527: (4)              return _mupdf.ll_fz_show_glyph(text, font, trm, glyph, unicode, wmode,
bidi_level, markup_dir, language)
11528: (0)          def ll_fz_show_glyph_aux(text, font, trm, glyph, unicode, cid, wmode,
bidi_level, markup_dir, lang):
11529: (4)              r""" Low-level wrapper for `::fz_show_glyph_aux()`."""
11530: (4)              return _mupdf.ll_fz_show_glyph_aux(text, font, trm, glyph, unicode, cid,
wmode, bidi_level, markup_dir, lang)
11531: (0)          def ll_fz_show_string(text, font, trm, s, wmode, bidi_level, markup_dir,
language):
11532: (4)              r"""
11533: (4)              Low-level wrapper for `::fz_show_string()`.
11534: (4)              Add a UTF8 string to a text object.
11535: (4)              text: Text object to add to.
11536: (4)              font: The font the string should be added in.
11537: (4)              trm: The transform to use.
11538: (4)              s: The utf-8 string to add.
11539: (4)              wmode: 1 for vertical mode, 0 for horizontal.
11540: (4)              bidi_level: The bidirectional level for this glyph.
11541: (4)              markup_dir: The direction of the text as specified in the markup.
11542: (4)              language: The language in use (if known, 0 otherwise)
11543: (8)                  (e.g. FZ_LANG_zh_Hans).
11544: (4)              Returns the transform updated with the advance width of the
11545: (4)              string.
11546: (4)              """
11547: (4)              return _mupdf.ll_fz_show_string(text, font, trm, s, wmode, bidi_level,
markup_dir, language)
11548: (0)          def ll_fz_shrink_store(percent):
11549: (4)              r"""
11550: (4)              Low-level wrapper for `::fz_shrink_store()`.
11551: (4)              Evict items from the store until the total size of
11552: (4)              the objects in the store is reduced to a given percentage of its
11553: (4)              current size.
```

```
11554: (4)             percent: %age of current size to reduce the store to.
11555: (4)             Returns non zero if we managed to free enough memory, zero
11556: (4)             otherwise.
11557: (4)             """
11558: (4)             return _mupdf.ll_fz_shrink_store(percent)
11559: (0)         def ll_fz_skip(stm, len):
11560: (4)             r"""
11561: (4)             Low-level wrapper for `::fz_skip()`.
11562: (4)             Read from a stream discarding data.
11563: (4)             stm: The stream to read from.
11564: (4)             len: The number of bytes to read.
11565: (4)             Returns the number of bytes read. May throw exceptions.
11566: (4)             """
11567: (4)             return _mupdf.ll_fz_skip(stm, len)
11568: (0)         def ll_fz_skip_space(stm):
11569: (4)             r"""
11570: (4)             Low-level wrapper for `::fz_skip_space()`.
11571: (4)             Skip over whitespace (bytes <= 32) in a stream.
11572: (4)             """
11573: (4)             return _mupdf.ll_fz_skip_space(stm)
11574: (0)         def ll_fz_skip_string(stm, str):
11575: (4)             r"""
11576: (4)             Low-level wrapper for `::fz_skip_string()`.
11577: (4)             Skip over a given string in a stream. Return 0 if successfully
11578: (4)             skipped, non-zero otherwise. As many characters will be skipped
11579: (4)             over as matched in the string.
11580: (4)             """
11581: (4)             return _mupdf.ll_fz_skip_string(stm, str)
11582: (0)         def ll_fz_slice_buffer(buf, start, end):
11583: (4)             r"""
11584: (4)             Low-level wrapper for `::fz_slice_buffer()`.
11585: (4)             Create a new buffer with a (subset of) the data from the buffer.
11586: (4)             start: if >= 0, offset from start of buffer, if < 0 offset from end of
buffer.
11587: (4)             end: if >= 0, offset from start of buffer, if < 0 offset from end of
buffer.
11588: (4)             """
11589: (4)             return _mupdf.ll_fz_slice_buffer(buf, start, end)
11590: (0)         def ll_fz_snap_selection(page, ap, bp, mode):
11591: (4)             r""" Low-level wrapper for `::fz_snap_selection()`."""
11592: (4)             return _mupdf.ll_fz_snap_selection(page, ap, bp, mode)
11593: (0)         def ll_fz_start_throw_on_repair():
11594: (4)             r""" Low-level wrapper for `::fz_start_throw_on_repair()`."""
11595: (4)             return _mupdf.ll_fz_start_throw_on_repair()
11596: (0)         def ll_fz_stat_ctime(path):
11597: (4)             r""" Low-level wrapper for `::fz_stat_ctime()`."""
11598: (4)             return _mupdf.ll_fz_stat_ctime(path)
11599: (0)         def ll_fz_stat_mtime(path):
11600: (4)             r""" Low-level wrapper for `::fz_stat_mtime()`."""
11601: (4)             return _mupdf.ll_fz_stat_mtime(path)
11602: (0)         def ll_fz_stddbg():
11603: (4)             r"""
11604: (4)             Low-level wrapper for `::fz_stddbg()`.
11605: (4)             Retrieve an fz_output for the default debugging stream. On
11606: (4)             Windows this will be OutputDebugString for non-console apps.
11607: (4)             Otherwise, it is always fz_stderr.
11608: (4)             Optionally may be fz_dropped when finished with.
11609: (4)             """
11610: (4)             return _mupdf.ll_fz_stddbg()
11611: (0)         def ll_fz_stderr():
11612: (4)             r"""
11613: (4)             Low-level wrapper for `::fz_stderr()`.
11614: (4)             Retrieve an fz_output that directs to stdout.
11615: (4)             Optionally may be fz_dropped when finished with.
11616: (4)             """
11617: (4)             return _mupdf.ll_fz_stderr()
11618: (0)         def ll_fz_stdout():
11619: (4)             r"""
11620: (4)             Low-level wrapper for `::fz_stdout()`.
```

```
11621: (4)             Retrieve an fz_output that directs to stdout.
11622: (4)             Optionally may be fz_dropped when finished with.
11623: (4)             """
11624: (4)             return _mupdf.ll_fz_stdout()
11625: (0)         def ll_fz_store_item(key, val, itemsize, type):
11626: (4)             r"""
11627: (4)             Low-level wrapper for `::fz_store_item()`.
11628: (4)             Add an item to the store.
11629: (4)             Add an item into the store, returning NULL for success. If an
11630: (4)             item with the same key is found in the store, then our item will
11631: (4)             not be inserted, and the function will return a pointer to that
11632: (4)             value instead. This function takes its own reference to val, as
11633: (4)             required (i.e. the caller maintains ownership of its own
11634: (4)             reference).
11635: (4)             key: The key used to index the item.
11636: (4)             val: The value to store.
11637: (4)             itemsize: The size in bytes of the value (as counted towards the
11638: (4)             store size).
11639: (4)             type: Functions used to manipulate the key.
11640: (4)             """
11641: (4)             return _mupdf.ll_fz_store_item(key, val, itemsize, type)
11642: (0)         def ll_fz_store_scavenge(size, phase):
11643: (4)             r"""
11644: (5)              Low-level wrapper for `::fz_store_scavenge()`.
11645: (4)             This function has out-params. Python/C# wrappers look like:
11646: (8)                 `ll_fz_store_scavenge(size_t size)` => `(int, int phase)`
11647: (8)                 Internal function used as part of the scavenging
11648: (8)                 allocator; when we fail to allocate memory, before returning a
11649: (8)                 failure to the caller, we try to scavenge space within the store
11650: (8)                 by evicting at least 'size' bytes. The allocator then retries.
11651: (8)                 size: The number of bytes we are trying to have free.
11652: (8)                 phase: What phase of the scavenge we are in. Updated on exit.
11653: (8)                 Returns non zero if we managed to free any memory.
11654: (4)             """
11655: (4)             return _mupdf.ll_fz_store_scavenge(size, phase)
11656: (0)         def ll_fz_store_scavenge_external(size, phase):
11657: (4)             r"""
11658: (5)              Low-level wrapper for `::fz_store_scavenge_external()`.
11659: (4)             This function has out-params. Python/C# wrappers look like:
11660: (8)                 `ll_fz_store_scavenge_external(size_t size)` => `(int, int phase)`
11661: (8)                 External function for callers to use
11662: (8)                 to scavenge while trying allocations.
11663: (8)                 size: The number of bytes we are trying to have free.
11664: (8)                 phase: What phase of the scavenge we are in. Updated on exit.
11665: (8)                 Returns non zero if we managed to free any memory.
11666: (4)             """
11667: (4)             return _mupdf.ll_fz_store_scavenge_external(size, phase)
11668: (0)         def ll_fz_story_document(story):
11669: (4)             r""" Low-level wrapper for `::fz_story_document()`."""
11670: (4)             return _mupdf.ll_fz_story_document(story)
11671: (0)         def ll_fz_story_positions(story, cb, arg):
11672: (4)             r""" Low-level wrapper for `::fz_story_positions()`."""
11673: (4)             return _mupdf.ll_fz_story_positions(story, cb, arg)
11674: (0)         def ll_fz_story_warnings(story):
11675: (4)             r""" Low-level wrapper for `::fz_story_warnings()`."""
11676: (4)             return _mupdf.ll_fz_story_warnings(story)
11677: (0)         def ll_fz_strcasecmp(a, b):
11678: (4)             r"""
11679: (4)             Low-level wrapper for `::fz_strcasecmp()`.
11680: (4)             Case insensitive (ASCII only) string comparison.
11681: (4)             """
11682: (4)             return _mupdf.ll_fz_strcasecmp(a, b)
11683: (0)         def ll_fz_strdup(s):
11684: (4)             r"""
11685: (4)             Low-level wrapper for `::fz_strdup()`.
11686: (4)             Portable strdup implementation, using fz allocators.
11687: (4)             """
11688: (4)             return _mupdf.ll_fz_strdup(s)
11689: (0)         def ll_fz_stream_filename(stm):
```

```
11690: (4)              r"""
11691: (4)              Low-level wrapper for `::fz_stream_filename()`.
11692: (4)              Return the filename (UTF-8 encoded) from which a stream was opened.
11693: (4)              Returns NULL if the filename is not available (or the stream was
11694: (4)              opened from a source other than a file).
11695: (4)              """
11696: (4)              return _mupdf.ll_fz_stream_filename(stm)
11697: (0)          def ll_fz_stream_from_output(arg_1):
11698: (4)              r"""
11699: (4)              Low-level wrapper for `::fz_stream_from_output()`.
11700: (4)              Obtain the fz_output in the form of a fz_stream.
11701: (4)              This allows data to be read back from some forms of fz_output
11702: (4)              object. When finished reading, the fz_stream should be released
11703: (4)              by calling fz_drop_stream. Until the fz_stream is dropped, no
11704: (4)              further operations should be performed on the fz_output object.
11705: (4)              """
11706: (4)              return _mupdf.ll_fz_stream_from_output(arg_1)
11707: (0)          def ll_fz_string_from_box_type(box):
11708: (4)              r""" Low-level wrapper for `::fz_string_from_box_type()`."""
11709: (4)              return _mupdf.ll_fz_string_from_box_type(box)
11710: (0)          def ll_fz_string_from_buffer(buf):
11711: (4)              r"""
11712: (4)              Low-level wrapper for `::fz_string_from_buffer()`.
11713: (4)              Ensure that a buffer's data ends in a
11714: (4)              0 byte, and return a pointer to it.
11715: (4)              """
11716: (4)              return _mupdf.ll_fz_string_from_buffer(buf)
11717: (0)          def ll_fz_string_from_text_language(str, lang):
11718: (4)              r"""
11719: (4)              Low-level wrapper for `::fz_string_from_text_language()`.
11720: (4)              Recover ISO 639 (639-{1,2,3,5}) language specification
11721: (4)              strings losslessly from a 15 bit fz_text_language code.
11722: (4)              No validation is carried out. See note above.
11723: (4)              """
11724: (4)              return _mupdf.ll_fz_string_from_text_language(str, lang)
11725: (0)          def ll_fz_string_from_text_language2(lang):
11726: (4)              r"""
11727: (5)               Low-level wrapper for `::fz_string_from_text_language2()`.
11728: (4)              C++ alternative to fz_string_from_text_language() that returns information
in a std::string.
11729: (4)              """
11730: (4)              return _mupdf.ll_fz_string_from_text_language2(lang)
11731: (0)          def ll_fz_strlcat(dst, src, n):
11732: (4)              r"""
11733: (4)              Low-level wrapper for `::fz_strlcat()`.
11734: (4)              Concatenate 2 strings, with a maximum length.
11735: (4)              dst: pointer to first string in a buffer of n bytes.
11736: (4)              src: pointer to string to concatenate.
11737: (4)              n: Size (in bytes) of buffer that dst is in.
11738: (4)              Returns the real length that a concatenated dst + src would have
11739: (4)              been (not including terminator).
11740: (4)              """
11741: (4)              return _mupdf.ll_fz_strlcat(dst, src, n)
11742: (0)          def ll_fz_strlcpy(dst, src, n):
11743: (4)              r"""
11744: (4)              Low-level wrapper for `::fz_strlcpy()`.
11745: (4)              Copy at most n-1 chars of a string into a destination
11746: (4)              buffer with null termination, returning the real length of the
11747: (4)              initial string (excluding terminator).
11748: (4)              dst: Destination buffer, at least n bytes long.
11749: (4)              src: C string (non-NULL).
11750: (4)              n: Size of dst buffer in bytes.
11751: (4)              Returns the length (excluding terminator) of src.
11752: (4)              """
11753: (4)              return _mupdf.ll_fz_strlcpy(dst, src, n)
11754: (0)          def ll_fz_strncasecmp(a, b, n):
11755: (4)              r""" Low-level wrapper for `::fz_strncasecmp()`."""
11756: (4)              return _mupdf.ll_fz_strncasecmp(a, b, n)
11757: (0)          def ll_fz_strnlen(s, maxlen):
```

```
11758: (4)              r"""
11759: (4)              Low-level wrapper for `::fz_strnlen()`.
11760: (4)              Return strlen(s), if that is less than maxlen, or maxlen if
11761: (4)              there is no null byte ('') among the first maxlen bytes.
11762: (4)              """
11763: (4)              return _mupdf.ll_fz_strnlen(s, maxlen)
11764: (0)          def ll_fz_stroke_path(dev, path, stroke, ctm, colorspace, color, alpha,
color_params):
11765: (4)              r""" Low-level wrapper for `::fz_stroke_path()`."""
11766: (4)              return _mupdf.ll_fz_stroke_path(dev, path, stroke, ctm, colorspace, color,
alpha, color_params)
11767: (0)          def ll_fz_stroke_text(dev, text, stroke, ctm, colorspace, color, alpha,
color_params):
11768: (4)              r""" Low-level wrapper for `::fz_stroke_text()`."""
11769: (4)              return _mupdf.ll_fz_stroke_text(dev, text, stroke, ctm, colorspace, color,
alpha, color_params)
11770: (0)          def ll_fz_strsep(stringp, delim):
11771: (4)              r"""
11772: (5)               Low-level wrapper for `::fz_strsep()`.
11773: (4)              This function has out-params. Python/C# wrappers look like:
11774: (8)                  `ll_fz_strsep(const char *delim)` => `(char *, char *stringp)`
11775: (8)                  Given a pointer to a C string (or a pointer to NULL) break
11776: (8)                  it at the first occurrence of a delimiter char (from a given
11777: (8)                  set).
11778: (8)                  stringp: Pointer to a C string pointer (or NULL). Updated on
11779: (8)                  exit to point to the first char of the string after the
11780: (8)                  delimiter that was found. The string pointed to by stringp will
11781: (8)                  be corrupted by this call (as the found delimiter will be
11782: (8)                  overwritten by 0).
11783: (8)                  delim: A C string of acceptable delimiter characters.
11784: (8)                  Returns a pointer to a C string containing the chars of stringp
11785: (8)                  up to the first delimiter char (or the end of the string), or
11786: (8)                  NULL.
11787: (4)              """
11788: (4)              return _mupdf.ll_fz_strsep(stringp, delim)
11789: (0)          def ll_fz_strtof(s, es):
11790: (4)              r"""
11791: (5)               Low-level wrapper for `::fz_strtof()`.
11792: (4)              This function has out-params. Python/C# wrappers look like:
11793: (8)                  `ll_fz_strtof(const char *s)` => `(float, char *es)`
11794: (8)                  Locale-independent decimal to binary conversion. On overflow
11795: (8)                  return (-)INFINITY and set errno to ERANGE. On underflow return
11796: (8)                  0 and set errno to ERANGE. Special inputs (case insensitive):
11797: (8)                  "NAN", "INF" or "INFINITY".
11798: (4)              """
11799: (4)              return _mupdf.ll_fz_strtof(s, es)
11800: (0)          def ll_fz_structure_from_string(str):
11801: (4)              r""" Low-level wrapper for `::fz_structure_from_string()`."""
11802: (4)              return _mupdf.ll_fz_structure_from_string(str)
11803: (0)          def ll_fz_structure_to_string(type):
11804: (4)              r""" Low-level wrapper for `::fz_structure_to_string()`."""
11805: (4)              return _mupdf.ll_fz_structure_to_string(type)
11806: (0)          def ll_fz_subpixel_adjust(ctm, subpix_ctm, qe, qf):
11807: (4)              r"""
11808: (4)              Low-level wrapper for `::fz_subpixel_adjust()`.
11809: (4)              Perform subpixel quantisation and adjustment on a glyph matrix.
11810: (4)              ctm: On entry, the desired 'ideal' transformation for a glyph.
11811: (4)              On exit, adjusted to a (very similar) transformation quantised
11812: (4)              for subpixel caching.
11813: (4)              subpix_ctm: Initialised by the routine to the transform that
11814: (4)              should be used to render the glyph.
11815: (4)              qe, qf: which subpixel position we quantised to.
11816: (4)              Returns: the size of the glyph.
11817: (4)              Note: This is currently only exposed for use in our app. It
11818: (4)              should be considered "at risk" of removal from the API.
11819: (4)              """
11820: (4)              return _mupdf.ll_fz_subpixel_adjust(ctm, subpix_ctm, qe, qf)
11821: (0)          def ll_fz_subsample_pixmap(tile, factor):
11822: (4)              r""" Low-level wrapper for `::fz_subsample_pixmap()`."""
```

```
11823: (4)              return _mupdf.ll_fz_subsample_pixmap(tile, factor)
11824: (0)          def ll_fz_subset_cff_for_gids(orig, gids, num_gids, symbolic, cidfont):
11825: (4)              r"""
11826: (5)               Low-level wrapper for `::fz_subset_cff_for_gids()`.
11827: (4)              This function has out-params. Python/C# wrappers look like:
11828: (8)                  `ll_fz_subset_cff_for_gids(::fz_buffer *orig, int num_gids, int
symbolic, int cidfont)` => `(fz_buffer *, int gids)`
11829: (4)              """
11830: (4)              return _mupdf.ll_fz_subset_cff_for_gids(orig, gids, num_gids, symbolic,
cidfont)
11831: (0)          def ll_fz_subset_ttf_for_gids(orig, gids, num_gids, symbolic, cidfont):
11832: (4)              r"""
11833: (5)               Low-level wrapper for `::fz_subset_ttf_for_gids()`.
11834: (4)              This function has out-params. Python/C# wrappers look like:
11835: (8)                  `ll_fz_subset_ttf_for_gids(::fz_buffer *orig, int num_gids, int
symbolic, int cidfont)` => `(fz_buffer *, int gids)`
11836: (4)              """
11837: (4)              return _mupdf.ll_fz_subset_ttf_for_gids(orig, gids, num_gids, symbolic,
cidfont)
11838: (0)          def ll_fz_sync_bits(stm):
11839: (4)              r"""
11840: (4)              Low-level wrapper for `::fz_sync_bits()`.
11841: (4)              Called after reading bits to tell the stream
11842: (4)              that we are about to return to reading bytewise. Resyncs
11843: (4)              the stream to whole byte boundaries.
11844: (4)              """
11845: (4)              return _mupdf.ll_fz_sync_bits(stm)
11846: (0)          def ll_fz_tell(stm):
11847: (4)              r"""
11848: (4)              Low-level wrapper for `::fz_tell()`.
11849: (4)              return the current reading position within a stream
11850: (4)              """
11851: (4)              return _mupdf.ll_fz_tell(stm)
11852: (0)          def ll_fz_tell_output(out):
11853: (4)              r"""
11854: (4)              Low-level wrapper for `::fz_tell_output()`.
11855: (4)              Return the current file position.
11856: (4)              Throw an error on untellable outputs.
11857: (4)              """
11858: (4)              return _mupdf.ll_fz_tell_output(out)
11859: (0)          def ll_fz_terminate_buffer(buf):
11860: (4)              r"""
11861: (4)              Low-level wrapper for `::fz_terminate_buffer()`.
11862: (4)              Zero-terminate buffer in order to use as a C string.
11863: (4)              This byte is invisible and does not affect the length of the
11864: (4)              buffer as returned by fz_buffer_storage. The zero byte is
11865: (4)              written *after* the data, and subsequent writes will overwrite
11866: (4)              the terminating byte.
11867: (4)              Subsequent changes to the size of the buffer (such as by
11868: (4)              fz_buffer_trim, fz_buffer_grow, fz_resize_buffer, etc) may
11869: (4)              invalidate this.
11870: (4)              """
11871: (4)              return _mupdf.ll_fz_terminate_buffer(buf)
11872: (0)          def ll_fz_text_aa_level():
11873: (4)              r"""
11874: (4)              Low-level wrapper for `::fz_text_aa_level()`.
11875: (4)              Get the number of bits of antialiasing we are
11876: (4)              using for text. Between 0 and 8.
11877: (4)              """
11878: (4)              return _mupdf.ll_fz_text_aa_level()
11879: (0)          def ll_fz_text_language_from_string(str):
11880: (4)              r"""
11881: (4)              Low-level wrapper for `::fz_text_language_from_string()`.
11882: (4)              Convert ISO 639 (639-{1,2,3,5}) language specification
11883: (4)              strings losslessly to a 15 bit fz_text_language code.
11884: (4)              No validation is carried out. Obviously invalid (out
11885: (4)              of spec) codes will be mapped to FZ_LANG_UNSET, but
11886: (4)              well-formed (but undefined) codes will be blithely
11887: (4)              accepted.
```

```
11888: (4)             """
11889: (4)             return _mupdf.ll_fz_text_language_from_string(str)
11890: (0)         def ll_fz_tint_pixmap(pix, black, white):
11891: (4)             r"""
11892: (4)             Low-level wrapper for `::fz_tint_pixmap()`.
11893: (4)             Tint all the pixels in an RGB, BGR, or Gray pixmap.
11894: (4)             black: Map black to this hexadecimal RGB color.
11895: (4)             white: Map white to this hexadecimal RGB color.
11896: (4)             """
11897: (4)             return _mupdf.ll_fz_tint_pixmap(pix, black, white)
11898: (0)         def ll_fz_tolower(c):
11899: (4)             r"""
11900: (4)             Low-level wrapper for `::fz_tolower()`.
11901: (4)             Unicode aware tolower and toupper functions.
11902: (4)             """
11903: (4)             return _mupdf.ll_fz_tolower(c)
11904: (0)         def ll_fz_toupper(c):
11905: (4)             r""" Low-level wrapper for `::fz_toupper()`."""
11906: (4)             return _mupdf.ll_fz_toupper(c)
11907: (0)         def ll_fz_transform_page(mediabox, resolution, rotate):
11908: (4)             r"""
11909: (4)             Low-level wrapper for `::fz_transform_page()`.
11910: (4)             Create transform matrix to draw page
11911: (4)             at a given resolution and rotation. Adjusts the scaling
11912: (4)             factors so that the page covers whole number of
11913: (4)             pixels and adjust the page origin to be at 0,0.
11914: (4)             """
11915: (4)             return _mupdf.ll_fz_transform_page(mediabox, resolution, rotate)
11916: (0)         def ll_fz_transform_path(path, transform):
11917: (4)             r"""
11918: (4)             Low-level wrapper for `::fz_transform_path()`.
11919: (4)             Transform a path by a given
11920: (4)             matrix.
11921: (4)             path: The path to modify (must not be a packed path).
11922: (4)             transform: The transform to apply.
11923: (4)             Throws exceptions if the path is packed, or on failure
11924: (4)             to allocate.
11925: (4)             """
11926: (4)             return _mupdf.ll_fz_transform_path(path, transform)
11927: (0)         def ll_fz_transform_point(point, m):
11928: (4)             r"""
11929: (4)             Low-level wrapper for `::fz_transform_point()`.
11930: (4)             Apply a transformation to a point.
11931: (4)             transform: Transformation matrix to apply. See fz_concat,
11932: (4)             fz_scale, fz_rotate and fz_translate for how to create a
11933: (4)             matrix.
11934: (4)             point: Pointer to point to update.
11935: (4)             Returns transform (unchanged).
11936: (4)             """
11937: (4)             return _mupdf.ll_fz_transform_point(point, m)
11938: (0)         def ll_fz_transform_point_xy(x, y, m):
11939: (4)             r""" Low-level wrapper for `::fz_transform_point_xy()`."""
11940: (4)             return _mupdf.ll_fz_transform_point_xy(x, y, m)
11941: (0)         def ll_fz_transform_quad(q, m):
11942: (4)             r"""
11943: (4)             Low-level wrapper for `::fz_transform_quad()`.
11944: (4)             Transform a quad by a matrix.
11945: (4)             """
11946: (4)             return _mupdf.ll_fz_transform_quad(q, m)
11947: (0)         def ll_fz_transform_rect(rect, m):
11948: (4)             r"""
11949: (4)             Low-level wrapper for `::fz_transform_rect()`.
11950: (4)             Apply a transform to a rectangle.
11951: (4)             After the four corner points of the axis-aligned rectangle
11952: (4)             have been transformed it may not longer be axis-aligned. So a
11953: (4)             new axis-aligned rectangle is created covering at least the
11954: (4)             area of the transformed rectangle.
11955: (4)             transform: Transformation matrix to apply. See fz_concat,
11956: (4)             fz_scale and fz_rotate for how to create a matrix.
```

```
(4)             rect: Rectangle to be transformed. The two special cases
(4)             fz_empty_rect and fz_infinite_rect, may be used but are
(4)             returned unchanged as expected.
(4)             """
(4)             return _mupdf.ll_fz_transform_rect(rect, m)
(0)         def ll_fz_transform_vector(vector, m):
(4)             r"""
(4)             Low-level wrapper for `::fz_transform_vector()`.
(4)             Apply a transformation to a vector.
(4)             transform: Transformation matrix to apply. See fz_concat,
(4)             fz_scale and fz_rotate for how to create a matrix. Any
(4)             translation will be ignored.
(4)             vector: Pointer to vector to update.
(4)             """
(4)             return _mupdf.ll_fz_transform_vector(vector, m)
(0)         def ll_fz_translate(tx, ty):
(4)             r"""
(4)             Low-level wrapper for `::fz_translate()`.
(4)             Create a translation matrix.
(4)             The returned matrix is of the form [ 1 0 0 1 tx ty ].
(4)             m: A place to store the created matrix.
(4)             tx, ty: Translation distances along the X- and Y-axes. A
(4)             translation of 0 will not cause any translation along the
(4)             relevant axis.
(4)             Returns m.
(4)             """
(4)             return _mupdf.ll_fz_translate(tx, ty)
(0)         def ll_fz_translate_irect(a, xoff, yoff):
(4)             r""" Low-level wrapper for `::fz_translate_irect()`."""
(4)             return _mupdf.ll_fz_translate_irect(a, xoff, yoff)
(0)         def ll_fz_translate_rect(a, xoff, yoff):
(4)             r"""
(4)             Low-level wrapper for `::fz_translate_rect()`.
(4)             Translate bounding box.
(4)             Translate a bbox by a given x and y offset. Allows for overflow.
(4)             """
(4)             return _mupdf.ll_fz_translate_rect(a, xoff, yoff)
(0)         def ll_fz_tree_archive_add_buffer(arch_, name, buf):
(4)             r"""
(4)             Low-level wrapper for `::fz_tree_archive_add_buffer()`.
(4)             Add a named buffer to an existing tree archive.
(4)             The tree will take a new reference to the buffer. Ownership
(4)             is not transferred.
(4)             """
(4)             return _mupdf.ll_fz_tree_archive_add_buffer(arch_, name, buf)
(0)         def ll_fz_tree_archive_add_data(arch_, name, data, size):
(4)             r"""
(4)             Low-level wrapper for `::fz_tree_archive_add_data()`.
(4)             Add a named block of data to an existing tree archive.
(4)             The data will be copied into a buffer, and so the caller
(4)             may free it as soon as this returns.
(4)             """
(4)             return _mupdf.ll_fz_tree_archive_add_data(arch_, name, data, size)
(0)         def ll_fz_tree_insert(root, key, value):
(4)             r"""
(4)             Low-level wrapper for `::fz_tree_insert()`.
(4)             Insert a new key/value pair and rebalance the tree.
(4)             Return the new root of the tree after inserting and rebalancing.
(4)             May be called with a NULL root to create a new tree.
(4)             No data is copied into the tree structure; key and value are
(4)             merely kept as pointers.
(4)             """
(4)             return _mupdf.ll_fz_tree_insert(root, key, value)
(0)         def ll_fz_tree_lookup(node, key):
(4)             r"""
(4)             Low-level wrapper for `::fz_tree_lookup()`.
(4)             Look for the value of a node in the tree with the given key.
(4)             Simple pointer equivalence is used for key.
(4)             Returns NULL for no match.
```

```
12026: (4)             """
12027: (4)             return _mupdf.ll_fz_tree_lookup(node, key)
12028: (0)         def ll_fz_trim_buffer(buf):
12029: (4)             r"""
12030: (4)             Low-level wrapper for `::fz_trim_buffer()`.
12031: (4)             Trim wasted capacity from a buffer by resizing internal memory.
12032: (4)             """
12033: (4)             return _mupdf.ll_fz_trim_buffer(buf)
12034: (0)         def ll_fz_trim_path(path):
12035: (4)             r"""
12036: (4)             Low-level wrapper for `::fz_trim_path()`.
12037: (4)             Minimise the internal storage used by a path.
12038: (4)             As paths are constructed, the internal buffers
12039: (4)             grow. To avoid repeated reallocations they
12040: (4)             grow with some spare space. Once a path has
12041: (4)             been fully constructed, this call allows the
12042: (4)             excess space to be trimmed.
12043: (4)             """
12044: (4)             return _mupdf.ll_fz_trim_path(path)
12045: (0)         def ll_fz_truncate_output(arg_1):
12046: (4)             r"""
12047: (4)             Low-level wrapper for `::fz_truncate_output()`.
12048: (4)             Truncate the output at the current position.
12049: (4)             This allows output streams which have seeked back from the end
12050: (4)             of their storage to be truncated at the current point.
12051: (4)             """
12052: (4)             return _mupdf.ll_fz_truncate_output(arg_1)
12053: (0)         def ll_fz_try_invert_matrix(inv, src):
12054: (4)             r"""
12055: (4)             Low-level wrapper for `::fz_try_invert_matrix()`.
12056: (4)             Attempt to create an inverse matrix.
12057: (4)             inverse: Place to store inverse matrix.
12058: (4)             matrix: Matrix to invert. A degenerate matrix, where the
12059: (4)             determinant is equal to zero, can not be inverted.
12060: (4)             Returns 1 if matrix is degenerate (singular), or 0 otherwise.
12061: (4)             """
12062: (4)             return _mupdf.ll_fz_try_invert_matrix(inv, src)
12063: (0)         def ll_fz_try_open_archive_entry(arch, name):
12064: (4)             r"""
12065: (4)             Low-level wrapper for `::fz_try_open_archive_entry()`.
12066: (4)             Opens an archive entry as a stream.
12067: (4)             Returns NULL if a matching entry cannot be found, otherwise
12068: (4)             behaves exactly as fz_open_archive_entry.
12069: (4)             """
12070: (4)             return _mupdf.ll_fz_try_open_archive_entry(arch, name)
12071: (0)         def ll_fz_try_open_archive_with_stream(file):
12072: (4)             r"""
12073: (4)             Low-level wrapper for `::fz_try_open_archive_with_stream()`.
12074: (4)             Open zip or tar archive stream.
12075: (4)             Does the same as fz_open_archive_with_stream, but will not throw
12076: (4)             an error in the event of failing to recognise the format. Will
12077: (4)             still throw errors in other cases though!
12078: (4)             """
12079: (4)             return _mupdf.ll_fz_try_open_archive_with_stream(file)
12080: (0)         def ll_fz_try_open_file(name):
12081: (4)             r"""
12082: (4)             Low-level wrapper for `::fz_try_open_file()`.
12083: (4)             Open the named file and wrap it in a stream.
12084: (4)             Does the same as fz_open_file, but in the event the file
12085: (4)             does not open, it will return NULL rather than throw an
12086: (4)             exception.
12087: (4)             """
12088: (4)             return _mupdf.ll_fz_try_open_file(name)
12089: (0)         def ll_fz_try_parse_xml_archive_entry(dir, filename, preserve_white):
12090: (4)             r"""
12091: (4)             Low-level wrapper for `::fz_try_parse_xml_archive_entry()`.
12092: (4)             Try and parse the contents of an archive entry into a tree of xml nodes.
12093: (4)             preserve_white: whether to keep or delete all-whitespace nodes.
12094: (4)             Will return NULL if the archive entry can't be found. Otherwise behaves
```

```
12095: (4)              the same as fz_parse_xml_archive_entry. May throw exceptions.
12096: (4)              """
12097: (4)              return _mupdf.ll_fz_try_parse_xml_archive_entry(dir, filename,
preserve_white)
12098: (0)          def ll_fz_try_read_archive_entry(arch, name):
12099: (4)              r"""
12100: (4)              Low-level wrapper for `::fz_try_read_archive_entry()`.
12101: (4)              Reads all bytes in an archive entry
12102: (4)              into a buffer.
12103: (4)              name: Entry name to look for, this must be an exact match to
12104: (4)              the entry name in the archive.
12105: (4)              Returns NULL if a matching entry cannot be found. Otherwise behaves
12106: (4)              the same as fz_read_archive_entry. Exceptions may be thrown.
12107: (4)              """
12108: (4)              return _mupdf.ll_fz_try_read_archive_entry(arch, name)
12109: (0)          def ll_fz_try_read_file(filename):
12110: (4)              r"""
12111: (4)              Low-level wrapper for `::fz_try_read_file()`.
12112: (4)              Read all the contents of a file into a buffer.
12113: (4)              Returns NULL if the file does not exist, otherwise
12114: (4)              behaves exactly as fz_read_file.
12115: (4)              """
12116: (4)              return _mupdf.ll_fz_try_read_file(filename)
12117: (0)          def ll_fz_tune_image_decode(image_decode, arg):
12118: (4)              r"""
12119: (4)              Low-level wrapper for `::fz_tune_image_decode()`.
12120: (4)              Set the tuning function to use for
12121: (4)              image decode.
12122: (4)              image_decode: Function to use.
12123: (4)              arg: Opaque argument to be passed to tuning function.
12124: (4)              """
12125: (4)              return _mupdf.ll_fz_tune_image_decode(image_decode, arg)
12126: (0)          def ll_fz_tune_image_scale(image_scale, arg):
12127: (4)              r"""
12128: (4)              Low-level wrapper for `::fz_tune_image_scale()`.
12129: (4)              Set the tuning function to use for
12130: (4)              image scaling.
12131: (4)              image_scale: Function to use.
12132: (4)              arg: Opaque argument to be passed to tuning function.
12133: (4)              """
12134: (4)              return _mupdf.ll_fz_tune_image_scale(image_scale, arg)
12135: (0)          def ll_fz_unicode_from_glyph_name(name):
12136: (4)              r""" Low-level wrapper for `::fz_unicode_from_glyph_name()`."""
12137: (4)              return _mupdf.ll_fz_unicode_from_glyph_name(name)
12138: (0)          def ll_fz_unicode_from_glyph_name_strict(name):
12139: (4)              r""" Low-level wrapper for `::fz_unicode_from_glyph_name_strict()`."""
12140: (4)              return _mupdf.ll_fz_unicode_from_glyph_name_strict(name)
12141: (0)          def ll_fz_union_rect(a, b):
12142: (4)              r"""
12143: (4)              Low-level wrapper for `::fz_union_rect()`.
12144: (4)              Compute union of two rectangles.
12145: (4)              Given two rectangles, update the first to be the smallest
12146: (4)              axis-aligned rectangle that encompasses both given rectangles.
12147: (4)              If either rectangle is infinite then the union is also infinite.
12148: (4)              If either rectangle is empty then the union is simply the
12149: (4)              non-empty rectangle. Should both rectangles be empty, then the
12150: (4)              union is also empty.
12151: (4)              """
12152: (4)              return _mupdf.ll_fz_union_rect(a, b)
12153: (0)          def ll_fz_unlock(lock):
12154: (4)              r"""
12155: (4)              Low-level wrapper for `::fz_unlock()`.
12156: (4)              Unlock one of the user supplied mutexes.
12157: (4)              """
12158: (4)              return _mupdf.ll_fz_unlock(lock)
12159: (0)          def ll_fz_unpack_stream(src, depth, w, h, n, indexed, pad, skip):
12160: (4)              r""" Low-level wrapper for `::fz_unpack_stream()`."""
12161: (4)              return _mupdf.ll_fz_unpack_stream(src, depth, w, h, n, indexed, pad, skip)
12162: (0)          def ll_fz_unread_byte(stm):
```

```
12163: (4)             r"""
12164: (4)             Low-level wrapper for `::fz_unread_byte()`.
12165: (4)             Unread the single last byte successfully
12166: (4)             read from a stream. Do not call this without having
12167: (4)             successfully read a byte.
12168: (4)             stm: The stream to operate upon.
12169: (4)             """
12170: (4)             return _mupdf.ll_fz_unread_byte(stm)
12171: (0)         def ll_fz_unshare_stroke_state(shared):
12172: (4)             r"""
12173: (4)             Low-level wrapper for `::fz_unshare_stroke_state()`.
12174: (4)             Given a reference to a (possibly) shared stroke_state structure,
12175: (4)             return a reference to an equivalent stroke_state structure
12176: (4)             that is guaranteed to be unshared (i.e. one that can
12177: (4)             safely be modified).
12178: (4)             shared: The reference to a (possibly) shared structure
12179: (4)             to unshare. Ownership of this reference is passed in
12180: (4)             to this function, even in the case of exceptions being
12181: (4)             thrown.
12182: (4)             Exceptions may be thrown in the event of failure to
12183: (4)             allocate if required.
12184: (4)             """
12185: (4)             return _mupdf.ll_fz_unshare_stroke_state(shared)
12186: (0)         def ll_fz_unshare_stroke_state_with_dash_len(shared, len):
12187: (4)             r"""
12188: (4)             Low-level wrapper for `::fz_unshare_stroke_state_with_dash_len()`.
12189: (4)             Given a reference to a (possibly) shared stroke_state structure,
12190: (4)             return a reference to a stroke_state structure (with room for a
12191: (4)             given amount of dash data) that is guaranteed to be unshared
12192: (4)             (i.e. one that can safely be modified).
12193: (4)             shared: The reference to a (possibly) shared structure
12194: (4)             to unshare. Ownership of this reference is passed in
12195: (4)             to this function, even in the case of exceptions being
12196: (4)             thrown.
12197: (4)             Exceptions may be thrown in the event of failure to
12198: (4)             allocate if required.
12199: (4)             """
12200: (4)             return _mupdf.ll_fz_unshare_stroke_state_with_dash_len(shared, len)
12201: (0)         def ll_fz_urldecode(url):
12202: (4)             r"""
12203: (4)             Low-level wrapper for `::fz_urldecode()`.
12204: (4)             Like fz_decode_uri_component but in-place.
12205: (4)             """
12206: (4)             return _mupdf.ll_fz_urldecode(url)
12207: (0)         def ll_fz_use_document_css():
12208: (4)             r"""
12209: (4)             Low-level wrapper for `::fz_use_document_css()`.
12210: (4)             Return whether to respect document styles in HTML and EPUB.
12211: (4)             """
12212: (4)             return _mupdf.ll_fz_use_document_css()
12213: (0)         def ll_fz_user_context():
12214: (4)             r"""
12215: (4)             Low-level wrapper for `::fz_user_context()`.
12216: (4)             Read the user field from the context.
12217: (4)             """
12218: (4)             return _mupdf.ll_fz_user_context()
12219: (0)         def ll_fz_user_css():
12220: (4)             r"""
12221: (4)             Low-level wrapper for `::fz_user_css()`.
12222: (4)             Get the user stylesheet source text.
12223: (4)             """
12224: (4)             return _mupdf.ll_fz_user_css()
12225: (0)         def ll_fz_utflen(s):
12226: (4)             r"""
12227: (4)             Low-level wrapper for `::fz_utflen()`.
12228: (4)             Count how many runes the UTF-8 encoded string
12229: (4)             consists of.
12230: (4)             s: The UTF-8 encoded, NUL-terminated text string.
12231: (4)             Returns the number of runes in the string.
```

```
12232: (4)             """
12233: (4)             return _mupdf.ll_fz_utflen(s)
12234: (0)         def ll_fz_var_imp(arg_0):
12235: (4)             r""" Low-level wrapper for `::fz_var_imp()`."""
12236: (4)             return _mupdf.ll_fz_var_imp(arg_0)
12237: (0)         def ll_fz_walk_path(path, walker, arg):
12238: (4)             r"""
12239: (4)             Low-level wrapper for `::fz_walk_path()`.
12240: (4)             Walk the segments of a path, calling the
12241: (4)             appropriate callback function from a given set for each
12242: (4)             segment of the path.
12243: (4)             path: The path to walk.
12244: (4)             walker: The set of callback functions to use. The first
12245: (4)             4 callback pointers in the set must be non-NULL. The
12246: (4)             subsequent ones can either be supplied, or can be left
12247: (4)             as NULL, in which case the top 4 functions will be
12248: (4)             called as appropriate to simulate them.
12249: (4)             arg: An opaque argument passed in to each callback.
12250: (4)             Exceptions will only be thrown if the underlying callback
12251: (4)             functions throw them.
12252: (4)             """
12253: (4)             return _mupdf.ll_fz_walk_path(path, walker, arg)
12254: (0)         def ll_fz_warn(*args):
12255: (4)             r""" Low-level wrapper for `::fz_warn()`."""
12256: (4)             return _mupdf.ll_fz_warn(*args)
12257: (0)         def ll_fz_warning_callback(user):
12258: (4)             r"""
12259: (5)              Low-level wrapper for `::fz_warning_callback()`.
12260: (4)             This function has out-params. Python/C# wrappers look like:
12261: (8)                 `ll_fz_warning_callback()` => `(fz_warning_cb *, void *user)`
12262: (8)                 Retrieve the currently set warning callback, or NULL if none
12263: (8)                 has been set. Optionally, if user is non-NULL, the user pointer
12264: (8)                 given when the warning callback was set is also passed back to
12265: (8)                 the caller.
12266: (4)             """
12267: (4)             return _mupdf.ll_fz_warning_callback(user)
12268: (0)         def ll_fz_warp_pixmap(src, points, width, height):
12269: (4)             r""" Low-level wrapper for `::fz_warp_pixmap()`."""
12270: (4)             return _mupdf.ll_fz_warp_pixmap(src, points, width, height)
12271: (0)         def ll_fz_windows_1250_from_unicode(u):
12272: (4)             r""" Low-level wrapper for `::fz_windows_1250_from_unicode()`."""
12273: (4)             return _mupdf.ll_fz_windows_1250_from_unicode(u)
12274: (0)         def ll_fz_windows_1251_from_unicode(u):
12275: (4)             r""" Low-level wrapper for `::fz_windows_1251_from_unicode()`."""
12276: (4)             return _mupdf.ll_fz_windows_1251_from_unicode(u)
12277: (0)         def ll_fz_windows_1252_from_unicode(u):
12278: (4)             r""" Low-level wrapper for `::fz_windows_1252_from_unicode()`."""
12279: (4)             return _mupdf.ll_fz_windows_1252_from_unicode(u)
12280: (0)         def ll_fz_write_band(writer, stride, band_height, samples):
12281: (4)             r"""
12282: (4)             Low-level wrapper for `::fz_write_band()`.
12283: (4)             Cause a band writer to write the next band
12284: (4)             of data for an image.
12285: (4)             stride: The byte offset from the first byte of the data
12286: (4)             for a pixel to the first byte of the data for the same pixel
12287: (4)             on the row below.
12288: (4)             band_height: The number of lines in this band.
12289: (4)             samples: Pointer to first byte of the data.
12290: (4)             """
12291: (4)             return _mupdf.ll_fz_write_band(writer, stride, band_height, samples)
12292: (0)         def ll_fz_write_base64(out, data, size, newline):
12293: (4)             r"""
12294: (4)             Low-level wrapper for `::fz_write_base64()`.
12295: (4)             Write a base64 encoded data block, optionally with periodic
12296: (4)             newlines.
12297: (4)             """
12298: (4)             return _mupdf.ll_fz_write_base64(out, data, size, newline)
12299: (0)         def ll_fz_write_base64_buffer(out, data, newline):
12300: (4)             r"""
```

```
12301: (4)             Low-level wrapper for `::fz_write_base64_buffer()`.
12302: (4)             Write a base64 encoded fz_buffer, optionally with periodic
12303: (4)             newlines.
12304: (4)             """
12305: (4)             return _mupdf.ll_fz_write_base64_buffer(out, data, newline)
12306: (0)         def ll_fz_write_bitmap_as_pbm(out, bitmap):
12307: (4)             r"""
12308: (4)             Low-level wrapper for `::fz_write_bitmap_as_pbm()`.
12309: (4)             Write a bitmap as a pbm.
12310: (4)             """
12311: (4)             return _mupdf.ll_fz_write_bitmap_as_pbm(out, bitmap)
12312: (0)         def ll_fz_write_bitmap_as_pcl(out, bitmap, pcl):
12313: (4)             r"""
12314: (4)             Low-level wrapper for `::fz_write_bitmap_as_pcl()`.
12315: (4)             Write a bitmap as mono PCL.
12316: (4)             """
12317: (4)             return _mupdf.ll_fz_write_bitmap_as_pcl(out, bitmap, pcl)
12318: (0)         def ll_fz_write_bitmap_as_pkm(out, bitmap):
12319: (4)             r"""
12320: (4)             Low-level wrapper for `::fz_write_bitmap_as_pkm()`.
12321: (4)             Write a CMYK bitmap as a pkm.
12322: (4)             """
12323: (4)             return _mupdf.ll_fz_write_bitmap_as_pkm(out, bitmap)
12324: (0)         def ll_fz_write_bitmap_as_pwg(out, bitmap, pwg):
12325: (4)             r"""
12326: (4)             Low-level wrapper for `::fz_write_bitmap_as_pwg()`.
12327: (4)             Write a bitmap as a PWG.
12328: (4)             """
12329: (4)             return _mupdf.ll_fz_write_bitmap_as_pwg(out, bitmap, pwg)
12330: (0)         def ll_fz_write_bitmap_as_pwg_page(out, bitmap, pwg):
12331: (4)             r"""
12332: (4)             Low-level wrapper for `::fz_write_bitmap_as_pwg_page()`.
12333: (4)             Write a bitmap as a PWG page.
12334: (4)             Caller should provide a file header by calling
12335: (4)             fz_write_pwg_file_header, but can then write several pages to
12336: (4)             the same file.
12337: (4)             """
12338: (4)             return _mupdf.ll_fz_write_bitmap_as_pwg_page(out, bitmap, pwg)
12339: (0)         def ll_fz_write_bits(out, data, num_bits):
12340: (4)             r"""
12341: (4)             Low-level wrapper for `::fz_write_bits()`.
12342: (4)             Write num_bits of data to the end of the output stream, assumed to be
packed
12343: (4)             most significant bits first.
12344: (4)             """
12345: (4)             return _mupdf.ll_fz_write_bits(out, data, num_bits)
12346: (0)         def ll_fz_write_bits_sync(out):
12347: (4)             r"""
12348: (4)             Low-level wrapper for `::fz_write_bits_sync()`.
12349: (4)             Sync to byte boundary after writing bits.
12350: (4)             """
12351: (4)             return _mupdf.ll_fz_write_bits_sync(out)
12352: (0)         def ll_fz_write_buffer(out, data):
12353: (4)             r""" Low-level wrapper for `::fz_write_buffer()`."""
12354: (4)             return _mupdf.ll_fz_write_buffer(out, data)
12355: (0)         def ll_fz_write_byte(out, x):
12356: (4)             r""" Low-level wrapper for `::fz_write_byte()`."""
12357: (4)             return _mupdf.ll_fz_write_byte(out, x)
12358: (0)         def ll_fz_write_char(out, x):
12359: (4)             r""" Low-level wrapper for `::fz_write_char()`."""
12360: (4)             return _mupdf.ll_fz_write_char(out, x)
12361: (0)         def ll_fz_write_data(out, data, size):
12362: (4)             r"""
12363: (4)             Low-level wrapper for `::fz_write_data()`.
12364: (4)             Write data to output.
12365: (4)             data: Pointer to data to write.
12366: (4)             size: Size of data to write in bytes.
12367: (4)             """
12368: (4)             return _mupdf.ll_fz_write_data(out, data, size)
```

```
12369: (0)          def ll_fz_write_document(wri, doc):
12370: (4)              r"""
12371: (4)              Low-level wrapper for `::fz_write_document()`.
12372: (4)              Convenience function to feed all the pages of a document to
12373: (4)              fz_begin_page/fz_run_page/fz_end_page.
12374: (4)              """
12375: (4)              return _mupdf.ll_fz_write_document(wri, doc)
12376: (0)          def ll_fz_write_float_be(out, f):
12377: (4)              r""" Low-level wrapper for `::fz_write_float_be()`."""
12378: (4)              return _mupdf.ll_fz_write_float_be(out, f)
12379: (0)          def ll_fz_write_float_le(out, f):
12380: (4)              r""" Low-level wrapper for `::fz_write_float_le()`."""
12381: (4)              return _mupdf.ll_fz_write_float_le(out, f)
12382: (0)          def ll_fz_write_header(writer, w, h, n, alpha, xres, yres, pagenum, cs, seps):
12383: (4)              r"""
12384: (4)              Low-level wrapper for `::fz_write_header()`.
12385: (4)              Cause a band writer to write the header for
12386: (4)              a banded image with the given properties/dimensions etc. This
12387: (4)              also configures the bandwriter for the format of the data to be
12388: (4)              passed in future calls.
12389: (4)              w, h: Width and Height of the entire page.
12390: (4)              n: Number of components (including spots and alphas).
12391: (4)              alpha: Number of alpha components.
12392: (4)              xres, yres: X and Y resolutions in dpi.
12393: (4)              cs: Colorspace (NULL for bitmaps)
12394: (4)              seps: Separation details (or NULL).
12395: (4)              """
12396: (4)              return _mupdf.ll_fz_write_header(writer, w, h, n, alpha, xres, yres,
pagenum, cs, seps)
12397: (0)          def ll_fz_write_image_as_data_uri(out, image):
12398: (4)              r"""
12399: (4)              Low-level wrapper for `::fz_write_image_as_data_uri()`.
12400: (4)              Write image as a data URI (for HTML and SVG output).
12401: (4)              """
12402: (4)              return _mupdf.ll_fz_write_image_as_data_uri(out, image)
12403: (0)          def ll_fz_write_int16_be(out, x):
12404: (4)              r""" Low-level wrapper for `::fz_write_int16_be()`."""
12405: (4)              return _mupdf.ll_fz_write_int16_be(out, x)
12406: (0)          def ll_fz_write_int16_le(out, x):
12407: (4)              r""" Low-level wrapper for `::fz_write_int16_le()`."""
12408: (4)              return _mupdf.ll_fz_write_int16_le(out, x)
12409: (0)          def ll_fz_write_int32_be(out, x):
12410: (4)              r"""
12411: (4)              Low-level wrapper for `::fz_write_int32_be()`.
12412: (4)              Write different sized data to an output stream.
12413: (4)              """
12414: (4)              return _mupdf.ll_fz_write_int32_be(out, x)
12415: (0)          def ll_fz_write_int32_le(out, x):
12416: (4)              r""" Low-level wrapper for `::fz_write_int32_le()`."""
12417: (4)              return _mupdf.ll_fz_write_int32_le(out, x)
12418: (0)          def ll_fz_write_pixmap_as_data_uri(out, pixmap):
12419: (4)              r""" Low-level wrapper for `::fz_write_pixmap_as_data_uri()`."""
12420: (4)              return _mupdf.ll_fz_write_pixmap_as_data_uri(out, pixmap)
12421: (0)          def ll_fz_write_pixmap_as_jpeg(out, pix, quality, invert_cmyk):
12422: (4)              r"""
12423: (4)              Low-level wrapper for `::fz_write_pixmap_as_jpeg()`.
12424: (4)              Write a pixmap as a JPEG.
12425: (4)              """
12426: (4)              return _mupdf.ll_fz_write_pixmap_as_jpeg(out, pix, quality, invert_cmyk)
12427: (0)          def ll_fz_write_pixmap_as_jpx(out, pix, quality):
12428: (4)              r"""
12429: (4)              Low-level wrapper for `::fz_write_pixmap_as_jpx()`.
12430: (4)              Pixmap data as JP2K with no subsampling.
12431: (4)              quality = 100 = lossless
12432: (4)              otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
12433: (4)              """
12434: (4)              return _mupdf.ll_fz_write_pixmap_as_jpx(out, pix, quality)
12435: (0)          def ll_fz_write_pixmap_as_pam(out, pixmap):
```

```
12436: (4)             r"""
12437: (4)             Low-level wrapper for `::fz_write_pixmap_as_pam()`.
12438: (4)             Write a pixmap as a pnm (greyscale, rgb or cmyk, with or without
12439: (4)             alpha).
12440: (4)             """
12441: (4)             return _mupdf.ll_fz_write_pixmap_as_pam(out, pixmap)
12442: (0)         def ll_fz_write_pixmap_as_pcl(out, pixmap, pcl):
12443: (4)             r"""
12444: (4)             Low-level wrapper for `::fz_write_pixmap_as_pcl()`.
12445: (4)             Write an (RGB) pixmap as color PCL.
12446: (4)             """
12447: (4)             return _mupdf.ll_fz_write_pixmap_as_pcl(out, pixmap, pcl)
12448: (0)         def ll_fz_write_pixmap_as_pclm(out, pixmap, options):
12449: (4)             r"""
12450: (4)             Low-level wrapper for `::fz_write_pixmap_as_pclm()`.
12451: (4)             Write a (Greyscale or RGB) pixmap as pclm.
12452: (4)             """
12453: (4)             return _mupdf.ll_fz_write_pixmap_as_pclm(out, pixmap, options)
12454: (0)         def ll_fz_write_pixmap_as_pdfocr(out, pixmap, options):
12455: (4)             r"""
12456: (4)             Low-level wrapper for `::fz_write_pixmap_as_pdfocr()`.
12457: (4)             Write a (Greyscale or RGB) pixmap as pdfocr.
12458: (4)             """
12459: (4)             return _mupdf.ll_fz_write_pixmap_as_pdfocr(out, pixmap, options)
12460: (0)         def ll_fz_write_pixmap_as_png(out, pixmap):
12461: (4)             r"""
12462: (4)             Low-level wrapper for `::fz_write_pixmap_as_png()`.
12463: (4)             Write a (Greyscale or RGB) pixmap as a png.
12464: (4)             """
12465: (4)             return _mupdf.ll_fz_write_pixmap_as_png(out, pixmap)
12466: (0)         def ll_fz_write_pixmap_as_pnm(out, pixmap):
12467: (4)             r"""
12468: (4)             Low-level wrapper for `::fz_write_pixmap_as_pnm()`.
12469: (4)             Write a pixmap as a pnm (greyscale or rgb, no alpha).
12470: (4)             """
12471: (4)             return _mupdf.ll_fz_write_pixmap_as_pnm(out, pixmap)
12472: (0)         def ll_fz_write_pixmap_as_ps(out, pixmap):
12473: (4)             r"""
12474: (4)             Low-level wrapper for `::fz_write_pixmap_as_ps()`.
12475: (4)             Write a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
12476: (4)             """
12477: (4)             return _mupdf.ll_fz_write_pixmap_as_ps(out, pixmap)
12478: (0)         def ll_fz_write_pixmap_as_psd(out, pixmap):
12479: (4)             r"""
12480: (4)             Low-level wrapper for `::fz_write_pixmap_as_psd()`.
12481: (4)             Write a pixmap as a PSD file.
12482: (4)             """
12483: (4)             return _mupdf.ll_fz_write_pixmap_as_psd(out, pixmap)
12484: (0)         def ll_fz_write_pixmap_as_pwg(out, pixmap, pwg):
12485: (4)             r"""
12486: (4)             Low-level wrapper for `::fz_write_pixmap_as_pwg()`.
12487: (4)             Write a pixmap as a PWG.
12488: (4)             """
12489: (4)             return _mupdf.ll_fz_write_pixmap_as_pwg(out, pixmap, pwg)
12490: (0)         def ll_fz_write_pixmap_as_pwg_page(out, pixmap, pwg):
12491: (4)             r"""
12492: (4)             Low-level wrapper for `::fz_write_pixmap_as_pwg_page()`.
12493: (4)             Write a pixmap as a PWG page.
12494: (4)             Caller should provide a file header by calling
12495: (4)             fz_write_pwg_file_header, but can then write several pages to
12496: (4)             the same file.
12497: (4)             """
12498: (4)             return _mupdf.ll_fz_write_pixmap_as_pwg_page(out, pixmap, pwg)
12499: (0)         def ll_fz_write_ps_file_header(out):
12500: (4)             r"""
12501: (4)             Low-level wrapper for `::fz_write_ps_file_header()`.
12502: (4)             Write the file level header for ps band writer output.
12503: (4)             """
12504: (4)             return _mupdf.ll_fz_write_ps_file_header(out)
```

```
12505: (0)          def ll_fz_write_ps_file_trailer(out, pages):
12506: (4)              r"""
12507: (4)              Low-level wrapper for `::fz_write_ps_file_trailer()`.
12508: (4)              Write the file level trailer for ps band writer output.
12509: (4)              """
12510: (4)              return _mupdf.ll_fz_write_ps_file_trailer(out, pages)
12511: (0)          def ll_fz_write_pwg_file_header(out):
12512: (4)              r"""
12513: (4)              Low-level wrapper for `::fz_write_pwg_file_header()`.
12514: (4)              Output the file header to a pwg stream, ready for pages to follow it.
12515: (4)              """
12516: (4)              return _mupdf.ll_fz_write_pwg_file_header(out)
12517: (0)          def ll_fz_write_rune(out, rune):
12518: (4)              r"""
12519: (4)              Low-level wrapper for `::fz_write_rune()`.
12520: (4)              Write a UTF-8 encoded unicode character.
12521: (4)              """
12522: (4)              return _mupdf.ll_fz_write_rune(out, rune)
12523: (0)          def ll_fz_write_stabilized_story(writer, user_css, em, contentfn,
contentfn_ref, rectfn, rectfn_ref, pagefn, pagefn_ref, dir):
12524: (4)              r""" Low-level wrapper for `::fz_write_stabilized_story()`."""
12525: (4)              return _mupdf.ll_fz_write_stabilized_story(writer, user_css, em,
contentfn, contentfn_ref, rectfn, rectfn_ref, pagefn, pagefn_ref, dir)
12526: (0)          def ll_fz_write_story(writer, story, rectfn, rectfn_ref, positionfn,
positionfn_ref, pagefn, pagefn_ref):
12527: (4)              r""" Low-level wrapper for `::fz_write_story()`."""
12528: (4)              return _mupdf.ll_fz_write_story(writer, story, rectfn, rectfn_ref,
positionfn, positionfn_ref, pagefn, pagefn_ref)
12529: (0)          def ll_fz_write_stream(out, _in):
12530: (4)              r"""
12531: (4)              Low-level wrapper for `::fz_write_stream()`.
12532: (4)              Copy the stream contents to the output.
12533: (4)              """
12534: (4)              return _mupdf.ll_fz_write_stream(out, _in)
12535: (0)          def ll_fz_write_string(out, s):
12536: (4)              r"""
12537: (4)              Low-level wrapper for `::fz_write_string()`.
12538: (4)              Write a string. Does not write zero terminator.
12539: (4)              """
12540: (4)              return _mupdf.ll_fz_write_string(out, s)
12541: (0)          def ll_fz_write_uint16_be(out, x):
12542: (4)              r""" Low-level wrapper for `::fz_write_uint16_be()`."""
12543: (4)              return _mupdf.ll_fz_write_uint16_be(out, x)
12544: (0)          def ll_fz_write_uint16_le(out, x):
12545: (4)              r""" Low-level wrapper for `::fz_write_uint16_le()`."""
12546: (4)              return _mupdf.ll_fz_write_uint16_le(out, x)
12547: (0)          def ll_fz_write_uint32_be(out, x):
12548: (4)              r""" Low-level wrapper for `::fz_write_uint32_be()`."""
12549: (4)              return _mupdf.ll_fz_write_uint32_be(out, x)
12550: (0)          def ll_fz_write_uint32_le(out, x):
12551: (4)              r""" Low-level wrapper for `::fz_write_uint32_le()`."""
12552: (4)              return _mupdf.ll_fz_write_uint32_le(out, x)
12553: (0)          def ll_fz_write_zip_entry(zip, name, buf, compress):
12554: (4)              r"""
12555: (4)              Low-level wrapper for `::fz_write_zip_entry()`.
12556: (4)              Given a buffer of data, (optionally) compress it, and add it to
12557: (4)              the zip file with the given name.
12558: (4)              """
12559: (4)              return _mupdf.ll_fz_write_zip_entry(zip, name, buf, compress)
12560: (0)          def ll_fz_xml_add_att(pool, node, key, val):
12561: (4)              r"""
12562: (4)              Low-level wrapper for `::fz_xml_add_att()`.
12563: (4)              Add an attribute to an XML node.
12564: (4)              """
12565: (4)              return _mupdf.ll_fz_xml_add_att(pool, node, key, val)
12566: (0)          def ll_fz_xml_att(item, att):
12567: (4)              r"""
12568: (4)              Low-level wrapper for `::fz_xml_att()`.
12569: (4)              Return the value of an attribute of an XML node.
```

```
12570: (4)             NULL if the attribute doesn't exist.
12571: (4)             """
12572: (4)             return _mupdf.ll_fz_xml_att(item, att)
12573: (0)         def ll_fz_xml_att_alt(item, one, two):
12574: (4)             r"""
12575: (4)             Low-level wrapper for `::fz_xml_att_alt()`.
12576: (4)             Return the value of an attribute of an XML node.
12577: (4)             If the first attribute doesn't exist, try the second.
12578: (4)             NULL if neither attribute exists.
12579: (4)             """
12580: (4)             return _mupdf.ll_fz_xml_att_alt(item, one, two)
12581: (0)         def ll_fz_xml_att_eq(item, name, match):
12582: (4)             r"""
12583: (4)             Low-level wrapper for `::fz_xml_att_eq()`.
12584: (4)             Check for a matching attribute on an XML node.
12585: (4)             If the node has the requested attribute (name), and the value
12586: (4)             matches (match) then return 1. Otherwise, 0.
12587: (4)             """
12588: (4)             return _mupdf.ll_fz_xml_att_eq(item, name, match)
12589: (0)         def ll_fz_xml_down(item):
12590: (4)             r"""
12591: (4)             Low-level wrapper for `::fz_xml_down()`.
12592: (4)             Return first child of XML node.
12593: (4)             """
12594: (4)             return _mupdf.ll_fz_xml_down(item)
12595: (0)         def ll_fz_xml_find(item, tag):
12596: (4)             r"""
12597: (4)             Low-level wrapper for `::fz_xml_find()`.
12598: (4)             Search the siblings of XML nodes starting with item looking for
12599: (4)             the first with the given tag.
12600: (4)             Return NULL if none found.
12601: (4)             """
12602: (4)             return _mupdf.ll_fz_xml_find(item, tag)
12603: (0)         def ll_fz_xml_find_dfs(item, tag, att, match):
12604: (4)             r"""
12605: (4)             Low-level wrapper for `::fz_xml_find_dfs()`.
12606: (4)             Perform a depth first search from item, returning the first
12607: (4)             child that matches the given tag (or any tag if tag is NULL),
12608: (4)             with the given attribute (if att is non NULL), that matches
12609: (4)             match (if match is non NULL).
12610: (4)             """
12611: (4)             return _mupdf.ll_fz_xml_find_dfs(item, tag, att, match)
12612: (0)         def ll_fz_xml_find_dfs_top(item, tag, att, match, top):
12613: (4)             r"""
12614: (4)             Low-level wrapper for `::fz_xml_find_dfs_top()`.
12615: (4)             Perform a depth first search from item, returning the first
12616: (4)             child that matches the given tag (or any tag if tag is NULL),
12617: (4)             with the given attribute (if att is non NULL), that matches
12618: (4)             match (if match is non NULL). The search stops if it ever
12619: (4)             reaches the top of the tree, or the declared 'top' item.
12620: (4)             """
12621: (4)             return _mupdf.ll_fz_xml_find_dfs_top(item, tag, att, match, top)
12622: (0)         def ll_fz_xml_find_down(item, tag):
12623: (4)             r"""
12624: (4)             Low-level wrapper for `::fz_xml_find_down()`.
12625: (4)             Search the siblings of XML nodes starting with the first child
12626: (4)             of item looking for the first with the given tag.
12627: (4)             Return NULL if none found.
12628: (4)             """
12629: (4)             return _mupdf.ll_fz_xml_find_down(item, tag)
12630: (0)         def ll_fz_xml_find_down_match(item, tag, att, match):
12631: (4)             r"""
12632: (4)             Low-level wrapper for `::fz_xml_find_down_match()`.
12633: (4)             Search the siblings of XML nodes starting with the first child
12634: (4)             of item looking for the first with the given tag (or any tag if
12635: (4)             tag is NULL), and with a matching attribute.
12636: (4)             Return NULL if none found.
12637: (4)             """
12638: (4)             return _mupdf.ll_fz_xml_find_down_match(item, tag, att, match)
```

```
(0)          def ll_fz_xml_find_match(item, tag, att, match):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_find_match()`.
(4)              Search the siblings of XML nodes starting with item looking for
(4)              the first with the given tag (or any tag if tag is NULL), and
(4)              with a matching attribute.
(4)              Return NULL if none found.
(4)              """
(4)              return _mupdf.ll_fz_xml_find_match(item, tag, att, match)
(0)          def ll_fz_xml_find_next(item, tag):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_find_next()`.
(4)              Search the siblings of XML nodes starting with the first sibling
(4)              of item looking for the first with the given tag.
(4)              Return NULL if none found.
(4)              """
(4)              return _mupdf.ll_fz_xml_find_next(item, tag)
(0)          def ll_fz_xml_find_next_dfs(item, tag, att, match):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_find_next_dfs()`.
(4)              Perform a depth first search onwards from item, returning the first
(4)              child that matches the given tag (or any tag if tag is NULL),
(4)              with the given attribute (if att is non NULL), that matches
(4)              match (if match is non NULL).
(4)              """
(4)              return _mupdf.ll_fz_xml_find_next_dfs(item, tag, att, match)
(0)          def ll_fz_xml_find_next_dfs_top(item, tag, att, match, top):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_find_next_dfs_top()`.
(4)              Perform a depth first search onwards from item, returning the first
(4)              child that matches the given tag (or any tag if tag is NULL),
(4)              with the given attribute (if att is non NULL), that matches
(4)              match (if match is non NULL). The search stops if it ever reaches
(4)              the top of the tree, or the declared 'top' item.
(4)              """
(4)              return _mupdf.ll_fz_xml_find_next_dfs_top(item, tag, att, match, top)
(0)          def ll_fz_xml_find_next_match(item, tag, att, match):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_find_next_match()`.
(4)              Search the siblings of XML nodes starting with the first sibling
(4)              of item looking for the first with the given tag (or any tag if tag
(4)              is NULL), and with a matching attribute.
(4)              Return NULL if none found.
(4)              """
(4)              return _mupdf.ll_fz_xml_find_next_match(item, tag, att, match)
(0)          def ll_fz_xml_is_tag(item, name):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_is_tag()`.
(4)              Return true if the tag name matches.
(4)              """
(4)              return _mupdf.ll_fz_xml_is_tag(item, name)
(0)          def ll_fz_xml_next(item):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_next()`.
(4)              Return next sibling of XML node.
(4)              """
(4)              return _mupdf.ll_fz_xml_next(item)
(0)          def ll_fz_xml_prev(item):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_prev()`.
(4)              Return previous sibling of XML node.
(4)              """
(4)              return _mupdf.ll_fz_xml_prev(item)
(0)          def ll_fz_xml_root(xml):
(4)              r"""
(4)              Low-level wrapper for `::fz_xml_root()`.
(4)              Return the topmost XML node of a document.
(4)              """
(4)              return _mupdf.ll_fz_xml_root(xml)
```

```
12708: (0)          def ll_fz_xml_tag(item):
12709: (4)              r"""
12710: (4)              Low-level wrapper for `::fz_xml_tag()`.
12711: (4)              Return tag of XML node. Return NULL for text nodes.
12712: (4)              """
12713: (4)              return _mupdf.ll_fz_xml_tag(item)
12714: (0)          def ll_fz_xml_text(item):
12715: (4)              r"""
12716: (4)              Low-level wrapper for `::fz_xml_text()`.
12717: (4)              Return the text content of an XML node.
12718: (4)              Return NULL if the node is a tag.
12719: (4)              """
12720: (4)              return _mupdf.ll_fz_xml_text(item)
12721: (0)          def ll_fz_xml_up(item):
12722: (4)              r"""
12723: (4)              Low-level wrapper for `::fz_xml_up()`.
12724: (4)              Return parent of XML node.
12725: (4)              """
12726: (4)              return _mupdf.ll_fz_xml_up(item)
12727: (0)          def ll_pdf_abandon_operation(doc):
12728: (4)              r""" Low-level wrapper for `::pdf_abandon_operation()`."""
12729: (4)              return _mupdf.ll_pdf_abandon_operation(doc)
12730: (0)          def ll_pdf_access_alert_event(evt):
12731: (4)              r""" Low-level wrapper for `::pdf_access_alert_event()`."""
12732: (4)              return _mupdf.ll_pdf_access_alert_event(evt)
12733: (0)          def ll_pdf_access_exec_menu_item_event(evt):
12734: (4)              r""" Low-level wrapper for `::pdf_access_exec_menu_item_event()`."""
12735: (4)              return _mupdf.ll_pdf_access_exec_menu_item_event(evt)
12736: (0)          def ll_pdf_access_launch_url_event(evt):
12737: (4)              r""" Low-level wrapper for `::pdf_access_launch_url_event()`."""
12738: (4)              return _mupdf.ll_pdf_access_launch_url_event(evt)
12739: (0)          def ll_pdf_access_mail_doc_event(evt):
12740: (4)              r""" Low-level wrapper for `::pdf_access_mail_doc_event()`."""
12741: (4)              return _mupdf.ll_pdf_access_mail_doc_event(evt)
12742: (0)          def ll_pdf_add_annot_border_dash_item(annot, length):
12743: (4)              r""" Low-level wrapper for `::pdf_add_annot_border_dash_item()`."""
12744: (4)              return _mupdf.ll_pdf_add_annot_border_dash_item(annot, length)
12745: (0)          def ll_pdf_add_annot_ink_list(annot, n, stroke):
12746: (4)              r""" Low-level wrapper for `::pdf_add_annot_ink_list()`."""
12747: (4)              return _mupdf.ll_pdf_add_annot_ink_list(annot, n, stroke)
12748: (0)          def ll_pdf_add_annot_ink_list_stroke(annot):
12749: (4)              r""" Low-level wrapper for `::pdf_add_annot_ink_list_stroke()`."""
12750: (4)              return _mupdf.ll_pdf_add_annot_ink_list_stroke(annot)
12751: (0)          def ll_pdf_add_annot_ink_list_stroke_vertex(annot, p):
12752: (4)              r""" Low-level wrapper for `::pdf_add_annot_ink_list_stroke_vertex()`."""
12753: (4)              return _mupdf.ll_pdf_add_annot_ink_list_stroke_vertex(annot, p)
12754: (0)          def ll_pdf_add_annot_quad_point(annot, quad):
12755: (4)              r""" Low-level wrapper for `::pdf_add_annot_quad_point()`."""
12756: (4)              return _mupdf.ll_pdf_add_annot_quad_point(annot, quad)
12757: (0)          def ll_pdf_add_annot_vertex(annot, p):
12758: (4)              r""" Low-level wrapper for `::pdf_add_annot_vertex()`."""
12759: (4)              return _mupdf.ll_pdf_add_annot_vertex(annot, p)
12760: (0)          def ll_pdf_add_cid_font(doc, font):
12761: (4)              r""" Low-level wrapper for `::pdf_add_cid_font()`."""
12762: (4)              return _mupdf.ll_pdf_add_cid_font(doc, font)
12763: (0)          def ll_pdf_add_cjk_font(doc, font, script, wmode, serif):
12764: (4)              r""" Low-level wrapper for `::pdf_add_cjk_font()`."""
12765: (4)              return _mupdf.ll_pdf_add_cjk_font(doc, font, script, wmode, serif)
12766: (0)          def ll_pdf_add_codespace(cmap, low, high, n):
12767: (4)              r""" Low-level wrapper for `::pdf_add_codespace()`."""
12768: (4)              return _mupdf.ll_pdf_add_codespace(cmap, low, high, n)
12769: (0)          def ll_pdf_add_embedded_file(doc, filename, mimetype, contents, created,
modifed, add_checksum):
12770: (4)              r""" Low-level wrapper for `::pdf_add_embedded_file()`."""
12771: (4)              return _mupdf.ll_pdf_add_embedded_file(doc, filename, mimetype, contents,
created, modifed, add_checksum)
12772: (0)          def ll_pdf_add_hmtx(font, lo, hi, w):
12773: (4)              r""" Low-level wrapper for `::pdf_add_hmtx()`."""
12774: (4)              return _mupdf.ll_pdf_add_hmtx(font, lo, hi, w)
```

```
(0)         def ll_pdf_add_image(doc, image):
(4)             r""" Low-level wrapper for `::pdf_add_image()`."""
(4)             return _mupdf.ll_pdf_add_image(doc, image)
(0)         def ll_pdf_add_journal_fragment(doc, parent, copy, copy_stream, newobj):
(4)             r""" Low-level wrapper for `::pdf_add_journal_fragment()`."""
(4)             return _mupdf.ll_pdf_add_journal_fragment(doc, parent, copy, copy_stream,
newobj)
(0)         def ll_pdf_add_new_array(doc, initial):
(4)             r""" Low-level wrapper for `::pdf_add_new_array()`."""
(4)             return _mupdf.ll_pdf_add_new_array(doc, initial)
(0)         def ll_pdf_add_new_dict(doc, initial):
(4)             r""" Low-level wrapper for `::pdf_add_new_dict()`."""
(4)             return _mupdf.ll_pdf_add_new_dict(doc, initial)
(0)         def ll_pdf_add_object(doc, obj):
(4)             r""" Low-level wrapper for `::pdf_add_object()`."""
(4)             return _mupdf.ll_pdf_add_object(doc, obj)
(0)         def ll_pdf_add_object_drop(doc, obj):
(4)             r""" Low-level wrapper for `::pdf_add_object_drop()`."""
(4)             return _mupdf.ll_pdf_add_object_drop(doc, obj)
(0)         def ll_pdf_add_page(doc, mediabox, rotate, resources, contents):
(4)             r""" Low-level wrapper for `::pdf_add_page()`."""
(4)             return _mupdf.ll_pdf_add_page(doc, mediabox, rotate, resources, contents)
(0)         def ll_pdf_add_simple_font(doc, font, encoding):
(4)             r""" Low-level wrapper for `::pdf_add_simple_font()`."""
(4)             return _mupdf.ll_pdf_add_simple_font(doc, font, encoding)
(0)         def ll_pdf_add_stream(doc, buf, obj, compressed):
(4)             r""" Low-level wrapper for `::pdf_add_stream()`."""
(4)             return _mupdf.ll_pdf_add_stream(doc, buf, obj, compressed)
(0)         def ll_pdf_add_substitute_font(doc, font):
(4)             r""" Low-level wrapper for `::pdf_add_substitute_font()`."""
(4)             return _mupdf.ll_pdf_add_substitute_font(doc, font)
(0)         def ll_pdf_add_vmtx(font, lo, hi, x, y, w):
(4)             r""" Low-level wrapper for `::pdf_add_vmtx()`."""
(4)             return _mupdf.ll_pdf_add_vmtx(font, lo, hi, x, y, w)
(0)         def ll_pdf_annot_MK_BC(annot, n, color):
(4)             r"""
(5)              Low-level wrapper for `::pdf_annot_MK_BC()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_pdf_annot_MK_BC(::pdf_annot *annot, float color[4])` => int n
(4)             """
(4)             return _mupdf.ll_pdf_annot_MK_BC(annot, n, color)
(0)         def ll_pdf_annot_MK_BC_rgb(annot, rgb):
(4)             r""" Low-level wrapper for `::pdf_annot_MK_BC_rgb()`."""
(4)             return _mupdf.ll_pdf_annot_MK_BC_rgb(annot, rgb)
(0)         def ll_pdf_annot_MK_BG(annot, n, color):
(4)             r"""
(5)              Low-level wrapper for `::pdf_annot_MK_BG()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_pdf_annot_MK_BG(::pdf_annot *annot, float color[4])` => int n
(4)             """
(4)             return _mupdf.ll_pdf_annot_MK_BG(annot, n, color)
(0)         def ll_pdf_annot_MK_BG_rgb(annot, rgb):
(4)             r""" Low-level wrapper for `::pdf_annot_MK_BG_rgb()`."""
(4)             return _mupdf.ll_pdf_annot_MK_BG_rgb(annot, rgb)
(0)         def ll_pdf_annot_active(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_active()`."""
(4)             return _mupdf.ll_pdf_annot_active(annot)
(0)         def ll_pdf_annot_ap(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_ap()`."""
(4)             return _mupdf.ll_pdf_annot_ap(annot)
(0)         def ll_pdf_annot_author(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_author()`."""
(4)             return _mupdf.ll_pdf_annot_author(annot)
(0)         def ll_pdf_annot_border(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border()`."""
(4)             return _mupdf.ll_pdf_annot_border(annot)
(0)         def ll_pdf_annot_border_dash_count(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border_dash_count()`."""
(4)             return _mupdf.ll_pdf_annot_border_dash_count(annot)
```

```
(0)         def ll_pdf_annot_border_dash_item(annot, i):
(4)             r""" Low-level wrapper for `::pdf_annot_border_dash_item()`."""
(4)             return _mupdf.ll_pdf_annot_border_dash_item(annot, i)
(0)         def ll_pdf_annot_border_effect(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border_effect()`."""
(4)             return _mupdf.ll_pdf_annot_border_effect(annot)
(0)         def ll_pdf_annot_border_effect_intensity(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border_effect_intensity()`."""
(4)             return _mupdf.ll_pdf_annot_border_effect_intensity(annot)
(0)         def ll_pdf_annot_border_style(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border_style()`."""
(4)             return _mupdf.ll_pdf_annot_border_style(annot)
(0)         def ll_pdf_annot_border_width(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_border_width()`."""
(4)             return _mupdf.ll_pdf_annot_border_width(annot)
(0)         def ll_pdf_annot_color(annot, n, color):
(4)             r"""
(5)              Low-level wrapper for `::pdf_annot_color()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_pdf_annot_color(::pdf_annot *annot, float color[4])` => int n
(4)             """
(4)             return _mupdf.ll_pdf_annot_color(annot, n, color)
(0)         def ll_pdf_annot_contents(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_contents()`."""
(4)             return _mupdf.ll_pdf_annot_contents(annot)
(0)         def ll_pdf_annot_creation_date(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_creation_date()`."""
(4)             return _mupdf.ll_pdf_annot_creation_date(annot)
(0)         def ll_pdf_annot_default_appearance(annot, font, size, n, color):
(4)             r"""
(5)              Low-level wrapper for `::pdf_annot_default_appearance()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `ll_pdf_annot_default_appearance(::pdf_annot *annot, float color[4])`
=> `(const char *font, float size, int n)`
(4)             """
(4)             return _mupdf.ll_pdf_annot_default_appearance(annot, font, size, n, color)
(0)         def ll_pdf_annot_ensure_local_xref(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_ensure_local_xref()`."""
(4)             return _mupdf.ll_pdf_annot_ensure_local_xref(annot)
(0)         def ll_pdf_annot_event_blur(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_blur()`."""
(4)             return _mupdf.ll_pdf_annot_event_blur(annot)
(0)         def ll_pdf_annot_event_down(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_down()`."""
(4)             return _mupdf.ll_pdf_annot_event_down(annot)
(0)         def ll_pdf_annot_event_enter(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_enter()`."""
(4)             return _mupdf.ll_pdf_annot_event_enter(annot)
(0)         def ll_pdf_annot_event_exit(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_exit()`."""
(4)             return _mupdf.ll_pdf_annot_event_exit(annot)
(0)         def ll_pdf_annot_event_focus(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_focus()`."""
(4)             return _mupdf.ll_pdf_annot_event_focus(annot)
(0)         def ll_pdf_annot_event_page_close(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_page_close()`."""
(4)             return _mupdf.ll_pdf_annot_event_page_close(annot)
(0)         def ll_pdf_annot_event_page_invisible(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_page_invisible()`."""
(4)             return _mupdf.ll_pdf_annot_event_page_invisible(annot)
(0)         def ll_pdf_annot_event_page_open(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_page_open()`."""
(4)             return _mupdf.ll_pdf_annot_event_page_open(annot)
(0)         def ll_pdf_annot_event_page_visible(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_page_visible()`."""
(4)             return _mupdf.ll_pdf_annot_event_page_visible(annot)
(0)         def ll_pdf_annot_event_up(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_event_up()`."""
(4)             return _mupdf.ll_pdf_annot_event_up(annot)
```

```
(0)         def ll_pdf_annot_field_event_keystroke(doc, annot, evt):
(4)             r""" Low-level wrapper for `::pdf_annot_field_event_keystroke()`."""
(4)             return _mupdf.ll_pdf_annot_field_event_keystroke(doc, annot, evt)
(0)         def ll_pdf_annot_field_flags(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_field_flags()`."""
(4)             return _mupdf.ll_pdf_annot_field_flags(annot)
(0)         def ll_pdf_annot_field_label(widget):
(4)             r""" Low-level wrapper for `::pdf_annot_field_label()`."""
(4)             return _mupdf.ll_pdf_annot_field_label(widget)
(0)         def ll_pdf_annot_field_value(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_field_value()`."""
(4)             return _mupdf.ll_pdf_annot_field_value(annot)
(0)         def ll_pdf_annot_filespec(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_filespec()`."""
(4)             return _mupdf.ll_pdf_annot_filespec(annot)
(0)         def ll_pdf_annot_flags(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_flags()`."""
(4)             return _mupdf.ll_pdf_annot_flags(annot)
(0)         def ll_pdf_annot_has_author(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_author()`."""
(4)             return _mupdf.ll_pdf_annot_has_author(annot)
(0)         def ll_pdf_annot_has_border(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_border()`."""
(4)             return _mupdf.ll_pdf_annot_has_border(annot)
(0)         def ll_pdf_annot_has_border_effect(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_border_effect()`."""
(4)             return _mupdf.ll_pdf_annot_has_border_effect(annot)
(0)         def ll_pdf_annot_has_filespec(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_filespec()`."""
(4)             return _mupdf.ll_pdf_annot_has_filespec(annot)
(0)         def ll_pdf_annot_has_icon_name(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_icon_name()`."""
(4)             return _mupdf.ll_pdf_annot_has_icon_name(annot)
(0)         def ll_pdf_annot_has_ink_list(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_ink_list()`."""
(4)             return _mupdf.ll_pdf_annot_has_ink_list(annot)
(0)         def ll_pdf_annot_has_intent(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_intent()`."""
(4)             return _mupdf.ll_pdf_annot_has_intent(annot)
(0)         def ll_pdf_annot_has_interior_color(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_interior_color()`."""
(4)             return _mupdf.ll_pdf_annot_has_interior_color(annot)
(0)         def ll_pdf_annot_has_line(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_line()`."""
(4)             return _mupdf.ll_pdf_annot_has_line(annot)
(0)         def ll_pdf_annot_has_line_ending_styles(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_line_ending_styles()`."""
(4)             return _mupdf.ll_pdf_annot_has_line_ending_styles(annot)
(0)         def ll_pdf_annot_has_open(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_open()`."""
(4)             return _mupdf.ll_pdf_annot_has_open(annot)
(0)         def ll_pdf_annot_has_quad_points(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_quad_points()`."""
(4)             return _mupdf.ll_pdf_annot_has_quad_points(annot)
(0)         def ll_pdf_annot_has_quadding(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_quadding()`."""
(4)             return _mupdf.ll_pdf_annot_has_quadding(annot)
(0)         def ll_pdf_annot_has_rect(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_rect()`."""
(4)             return _mupdf.ll_pdf_annot_has_rect(annot)
(0)         def ll_pdf_annot_has_vertices(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_has_vertices()`."""
(4)             return _mupdf.ll_pdf_annot_has_vertices(annot)
(0)         def ll_pdf_annot_hidden_for_editing(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_hidden_for_editing()`."""
(4)             return _mupdf.ll_pdf_annot_hidden_for_editing(annot)
(0)         def ll_pdf_annot_hot(annot):
(4)             r""" Low-level wrapper for `::pdf_annot_hot()`."""
(4)             return _mupdf.ll_pdf_annot_hot(annot)
```

```
(0)          def ll_pdf_annot_icon_name(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_icon_name()`."""
(4)              return _mupdf.ll_pdf_annot_icon_name(annot)
(0)          def ll_pdf_annot_ink_list_count(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_ink_list_count()`."""
(4)              return _mupdf.ll_pdf_annot_ink_list_count(annot)
(0)          def ll_pdf_annot_ink_list_stroke_count(annot, i):
(4)              r""" Low-level wrapper for `::pdf_annot_ink_list_stroke_count()`."""
(4)              return _mupdf.ll_pdf_annot_ink_list_stroke_count(annot, i)
(0)          def ll_pdf_annot_ink_list_stroke_vertex(annot, i, k):
(4)              r""" Low-level wrapper for `::pdf_annot_ink_list_stroke_vertex()`."""
(4)              return _mupdf.ll_pdf_annot_ink_list_stroke_vertex(annot, i, k)
(0)          def ll_pdf_annot_intent(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_intent()`."""
(4)              return _mupdf.ll_pdf_annot_intent(annot)
(0)          def ll_pdf_annot_interior_color(annot, n, color):
(4)              r"""
(5)               Low-level wrapper for `::pdf_annot_interior_color()`.
(4)              This function has out-params. Python/C# wrappers look like:
(8)                  `ll_pdf_annot_interior_color(::pdf_annot *annot, float color[4])` =>
int n
(4)              """
(4)              return _mupdf.ll_pdf_annot_interior_color(annot, n, color)
(0)          def ll_pdf_annot_is_open(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_is_open()`."""
(4)              return _mupdf.ll_pdf_annot_is_open(annot)
(0)          def ll_pdf_annot_is_standard_stamp(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_is_standard_stamp()`."""
(4)              return _mupdf.ll_pdf_annot_is_standard_stamp(annot)
(0)          def ll_pdf_annot_language(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_language()`."""
(4)              return _mupdf.ll_pdf_annot_language(annot)
(0)          def ll_pdf_annot_line(annot, a, b):
(4)              r""" Low-level wrapper for `::pdf_annot_line()`."""
(4)              return _mupdf.ll_pdf_annot_line(annot, a, b)
(0)          def ll_pdf_annot_line_end_style(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_line_end_style()`."""
(4)              return _mupdf.ll_pdf_annot_line_end_style(annot)
(0)          def ll_pdf_annot_line_ending_styles(annot, start_style, end_style):
(4)              r"""
(5)               Low-level wrapper for `::pdf_annot_line_ending_styles()`.
(4)              This function has out-params. Python/C# wrappers look like:
(8)                  `ll_pdf_annot_line_ending_styles(::pdf_annot *annot)` => `(enum
pdf_line_ending start_style, enum pdf_line_ending end_style)`
(4)              """
(4)              return _mupdf.ll_pdf_annot_line_ending_styles(annot, start_style,
end_style)
(0)          def ll_pdf_annot_line_start_style(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_line_start_style()`."""
(4)              return _mupdf.ll_pdf_annot_line_start_style(annot)
(0)          def ll_pdf_annot_modification_date(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_modification_date()`."""
(4)              return _mupdf.ll_pdf_annot_modification_date(annot)
(0)          def ll_pdf_annot_needs_resynthesis(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_needs_resynthesis()`."""
(4)              return _mupdf.ll_pdf_annot_needs_resynthesis(annot)
(0)          def ll_pdf_annot_obj(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_obj()`."""
(4)              return _mupdf.ll_pdf_annot_obj(annot)
(0)          def ll_pdf_annot_opacity(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_opacity()`."""
(4)              return _mupdf.ll_pdf_annot_opacity(annot)
(0)          def ll_pdf_annot_page(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_page()`."""
(4)              return _mupdf.ll_pdf_annot_page(annot)
(0)          def ll_pdf_annot_pop_and_discard_local_xref(annot):
(4)              r""" Low-level wrapper for `::pdf_annot_pop_and_discard_local_xref()`."""
(4)              return _mupdf.ll_pdf_annot_pop_and_discard_local_xref(annot)
(0)          def ll_pdf_annot_pop_local_xref(annot):
```

```
13046: (4)             r""" Low-level wrapper for `::pdf_annot_pop_local_xref()`."""
13047: (4)             return _mupdf.ll_pdf_annot_pop_local_xref(annot)
13048: (0)         def ll_pdf_annot_popup(annot):
13049: (4)             r""" Low-level wrapper for `::pdf_annot_popup()`."""
13050: (4)             return _mupdf.ll_pdf_annot_popup(annot)
13051: (0)         def ll_pdf_annot_push_local_xref(annot):
13052: (4)             r""" Low-level wrapper for `::pdf_annot_push_local_xref()`."""
13053: (4)             return _mupdf.ll_pdf_annot_push_local_xref(annot)
13054: (0)         def ll_pdf_annot_quad_point(annot, i):
13055: (4)             r""" Low-level wrapper for `::pdf_annot_quad_point()`."""
13056: (4)             return _mupdf.ll_pdf_annot_quad_point(annot, i)
13057: (0)         def ll_pdf_annot_quad_point_count(annot):
13058: (4)             r""" Low-level wrapper for `::pdf_annot_quad_point_count()`."""
13059: (4)             return _mupdf.ll_pdf_annot_quad_point_count(annot)
13060: (0)         def ll_pdf_annot_quadding(annot):
13061: (4)             r""" Low-level wrapper for `::pdf_annot_quadding()`."""
13062: (4)             return _mupdf.ll_pdf_annot_quadding(annot)
13063: (0)         def ll_pdf_annot_rect(annot):
13064: (4)             r""" Low-level wrapper for `::pdf_annot_rect()`."""
13065: (4)             return _mupdf.ll_pdf_annot_rect(annot)
13066: (0)         def ll_pdf_annot_request_resynthesis(annot):
13067: (4)             r""" Low-level wrapper for `::pdf_annot_request_resynthesis()`."""
13068: (4)             return _mupdf.ll_pdf_annot_request_resynthesis(annot)
13069: (0)         def ll_pdf_annot_request_synthesis(annot):
13070: (4)             r""" Low-level wrapper for `::pdf_annot_request_synthesis()`."""
13071: (4)             return _mupdf.ll_pdf_annot_request_synthesis(annot)
13072: (0)         def ll_pdf_annot_transform(annot):
13073: (4)             r""" Low-level wrapper for `::pdf_annot_transform()`."""
13074: (4)             return _mupdf.ll_pdf_annot_transform(annot)
13075: (0)         def ll_pdf_annot_type(annot):
13076: (4)             r""" Low-level wrapper for `::pdf_annot_type()`."""
13077: (4)             return _mupdf.ll_pdf_annot_type(annot)
13078: (0)         def ll_pdf_annot_type_from_string(subtype):
13079: (4)             r""" Low-level wrapper for `::pdf_annot_type_from_string()`."""
13080: (4)             return _mupdf.ll_pdf_annot_type_from_string(subtype)
13081: (0)         def ll_pdf_annot_vertex(annot, i):
13082: (4)             r""" Low-level wrapper for `::pdf_annot_vertex()`."""
13083: (4)             return _mupdf.ll_pdf_annot_vertex(annot, i)
13084: (0)         def ll_pdf_annot_vertex_count(annot):
13085: (4)             r""" Low-level wrapper for `::pdf_annot_vertex_count()`."""
13086: (4)             return _mupdf.ll_pdf_annot_vertex_count(annot)
13087: (0)         def ll_pdf_append_explicit_dest_to_uri(url, dest):
13088: (4)             r""" Low-level wrapper for `::pdf_append_explicit_dest_to_uri()`."""
13089: (4)             return _mupdf.ll_pdf_append_explicit_dest_to_uri(url, dest)
13090: (0)         def ll_pdf_append_named_dest_to_uri(url, name):
13091: (4)             r""" Low-level wrapper for `::pdf_append_named_dest_to_uri()`."""
13092: (4)             return _mupdf.ll_pdf_append_named_dest_to_uri(url, name)
13093: (0)         def ll_pdf_append_token(buf, tok, lex):
13094: (4)             r""" Low-level wrapper for `::pdf_append_token()`."""
13095: (4)             return _mupdf.ll_pdf_append_token(buf, tok, lex)
13096: (0)         def ll_pdf_apply_redaction(annot, opts):
13097: (4)             r""" Low-level wrapper for `::pdf_apply_redaction()`."""
13098: (4)             return _mupdf.ll_pdf_apply_redaction(annot, opts)
13099: (0)         def ll_pdf_array_contains(array, obj):
13100: (4)             r""" Low-level wrapper for `::pdf_array_contains()`."""
13101: (4)             return _mupdf.ll_pdf_array_contains(array, obj)
13102: (0)         def ll_pdf_array_delete(array, index):
13103: (4)             r""" Low-level wrapper for `::pdf_array_delete()`."""
13104: (4)             return _mupdf.ll_pdf_array_delete(array, index)
13105: (0)         def ll_pdf_array_find(array, obj):
13106: (4)             r""" Low-level wrapper for `::pdf_array_find()`."""
13107: (4)             return _mupdf.ll_pdf_array_find(array, obj)
13108: (0)         def ll_pdf_array_get(array, i):
13109: (4)             r""" Low-level wrapper for `::pdf_array_get()`."""
13110: (4)             return _mupdf.ll_pdf_array_get(array, i)
13111: (0)         def ll_pdf_array_get_bool(array, index):
13112: (4)             r""" Low-level wrapper for `::pdf_array_get_bool()`."""
13113: (4)             return _mupdf.ll_pdf_array_get_bool(array, index)
13114: (0)         def ll_pdf_array_get_int(array, index):
```

```
13115: (4)             r""" Low-level wrapper for `::pdf_array_get_int()`."""
13116: (4)             return _mupdf.ll_pdf_array_get_int(array, index)
13117: (0)         def ll_pdf_array_get_matrix(array, index):
13118: (4)             r""" Low-level wrapper for `::pdf_array_get_matrix()`."""
13119: (4)             return _mupdf.ll_pdf_array_get_matrix(array, index)
13120: (0)         def ll_pdf_array_get_name(array, index):
13121: (4)             r""" Low-level wrapper for `::pdf_array_get_name()`."""
13122: (4)             return _mupdf.ll_pdf_array_get_name(array, index)
13123: (0)         def ll_pdf_array_get_real(array, index):
13124: (4)             r""" Low-level wrapper for `::pdf_array_get_real()`."""
13125: (4)             return _mupdf.ll_pdf_array_get_real(array, index)
13126: (0)         def ll_pdf_array_get_rect(array, index):
13127: (4)             r""" Low-level wrapper for `::pdf_array_get_rect()`."""
13128: (4)             return _mupdf.ll_pdf_array_get_rect(array, index)
13129: (0)         def ll_pdf_array_get_string(array, index, sizep):
13130: (4)             r"""
13131: (5)              Low-level wrapper for `::pdf_array_get_string()`.
13132: (4)             This function has out-params. Python/C# wrappers look like:
13133: (8)                 `ll_pdf_array_get_string(::pdf_obj *array, int index)` => `(const char
*, size_t sizep)`
13134: (4)             """
13135: (4)             return _mupdf.ll_pdf_array_get_string(array, index, sizep)
13136: (0)         def ll_pdf_array_get_text_string(array, index):
13137: (4)             r""" Low-level wrapper for `::pdf_array_get_text_string()`."""
13138: (4)             return _mupdf.ll_pdf_array_get_text_string(array, index)
13139: (0)         def ll_pdf_array_insert(array, obj, index):
13140: (4)             r""" Low-level wrapper for `::pdf_array_insert()`."""
13141: (4)             return _mupdf.ll_pdf_array_insert(array, obj, index)
13142: (0)         def ll_pdf_array_insert_drop(array, obj, index):
13143: (4)             r""" Low-level wrapper for `::pdf_array_insert_drop()`."""
13144: (4)             return _mupdf.ll_pdf_array_insert_drop(array, obj, index)
13145: (0)         def ll_pdf_array_len(array):
13146: (4)             r""" Low-level wrapper for `::pdf_array_len()`."""
13147: (4)             return _mupdf.ll_pdf_array_len(array)
13148: (0)         def ll_pdf_array_push(array, obj):
13149: (4)             r""" Low-level wrapper for `::pdf_array_push()`."""
13150: (4)             return _mupdf.ll_pdf_array_push(array, obj)
13151: (0)         def ll_pdf_array_push_array(array, initial):
13152: (4)             r""" Low-level wrapper for `::pdf_array_push_array()`."""
13153: (4)             return _mupdf.ll_pdf_array_push_array(array, initial)
13154: (0)         def ll_pdf_array_push_bool(array, x):
13155: (4)             r""" Low-level wrapper for `::pdf_array_push_bool()`."""
13156: (4)             return _mupdf.ll_pdf_array_push_bool(array, x)
13157: (0)         def ll_pdf_array_push_dict(array, initial):
13158: (4)             r""" Low-level wrapper for `::pdf_array_push_dict()`."""
13159: (4)             return _mupdf.ll_pdf_array_push_dict(array, initial)
13160: (0)         def ll_pdf_array_push_drop(array, obj):
13161: (4)             r""" Low-level wrapper for `::pdf_array_push_drop()`."""
13162: (4)             return _mupdf.ll_pdf_array_push_drop(array, obj)
13163: (0)         def ll_pdf_array_push_int(array, x):
13164: (4)             r""" Low-level wrapper for `::pdf_array_push_int()`."""
13165: (4)             return _mupdf.ll_pdf_array_push_int(array, x)
13166: (0)         def ll_pdf_array_push_name(array, x):
13167: (4)             r""" Low-level wrapper for `::pdf_array_push_name()`."""
13168: (4)             return _mupdf.ll_pdf_array_push_name(array, x)
13169: (0)         def ll_pdf_array_push_real(array, x):
13170: (4)             r""" Low-level wrapper for `::pdf_array_push_real()`."""
13171: (4)             return _mupdf.ll_pdf_array_push_real(array, x)
13172: (0)         def ll_pdf_array_push_string(array, x, n):
13173: (4)             r""" Low-level wrapper for `::pdf_array_push_string()`."""
13174: (4)             return _mupdf.ll_pdf_array_push_string(array, x, n)
13175: (0)         def ll_pdf_array_push_text_string(array, x):
13176: (4)             r""" Low-level wrapper for `::pdf_array_push_text_string()`."""
13177: (4)             return _mupdf.ll_pdf_array_push_text_string(array, x)
13178: (0)         def ll_pdf_array_put(array, i, obj):
13179: (4)             r""" Low-level wrapper for `::pdf_array_put()`."""
13180: (4)             return _mupdf.ll_pdf_array_put(array, i, obj)
13181: (0)         def ll_pdf_array_put_array(array, i, initial):
13182: (4)             r""" Low-level wrapper for `::pdf_array_put_array()`."""
```

```
13183: (4)             return _mupdf.ll_pdf_array_put_array(array, i, initial)
13184: (0)         def ll_pdf_array_put_bool(array, i, x):
13185: (4)             r""" Low-level wrapper for `::pdf_array_put_bool()`."""
13186: (4)             return _mupdf.ll_pdf_array_put_bool(array, i, x)
13187: (0)         def ll_pdf_array_put_dict(array, i, initial):
13188: (4)             r""" Low-level wrapper for `::pdf_array_put_dict()`."""
13189: (4)             return _mupdf.ll_pdf_array_put_dict(array, i, initial)
13190: (0)         def ll_pdf_array_put_drop(array, i, obj):
13191: (4)             r""" Low-level wrapper for `::pdf_array_put_drop()`."""
13192: (4)             return _mupdf.ll_pdf_array_put_drop(array, i, obj)
13193: (0)         def ll_pdf_array_put_int(array, i, x):
13194: (4)             r""" Low-level wrapper for `::pdf_array_put_int()`."""
13195: (4)             return _mupdf.ll_pdf_array_put_int(array, i, x)
13196: (0)         def ll_pdf_array_put_name(array, i, x):
13197: (4)             r""" Low-level wrapper for `::pdf_array_put_name()`."""
13198: (4)             return _mupdf.ll_pdf_array_put_name(array, i, x)
13199: (0)         def ll_pdf_array_put_real(array, i, x):
13200: (4)             r""" Low-level wrapper for `::pdf_array_put_real()`."""
13201: (4)             return _mupdf.ll_pdf_array_put_real(array, i, x)
13202: (0)         def ll_pdf_array_put_string(array, i, x, n):
13203: (4)             r""" Low-level wrapper for `::pdf_array_put_string()`."""
13204: (4)             return _mupdf.ll_pdf_array_put_string(array, i, x, n)
13205: (0)         def ll_pdf_array_put_text_string(array, i, x):
13206: (4)             r""" Low-level wrapper for `::pdf_array_put_text_string()`."""
13207: (4)             return _mupdf.ll_pdf_array_put_text_string(array, i, x)
13208: (0)         def ll_pdf_authenticate_password(doc, pw):
13209: (4)             r""" Low-level wrapper for `::pdf_authenticate_password()`."""
13210: (4)             return _mupdf.ll_pdf_authenticate_password(doc, pw)
13211: (0)         def ll_pdf_bake_document(doc, bake_annots, bake_widgets):
13212: (4)             r""" Low-level wrapper for `::pdf_bake_document()`."""
13213: (4)             return _mupdf.ll_pdf_bake_document(doc, bake_annots, bake_widgets)
13214: (0)         def ll_pdf_begin_implicit_operation(doc):
13215: (4)             r""" Low-level wrapper for `::pdf_begin_implicit_operation()`."""
13216: (4)             return _mupdf.ll_pdf_begin_implicit_operation(doc)
13217: (0)         def ll_pdf_begin_operation(doc, operation):
13218: (4)             r""" Low-level wrapper for `::pdf_begin_operation()`."""
13219: (4)             return _mupdf.ll_pdf_begin_operation(doc, operation)
13220: (0)         def ll_pdf_bound_annot(annot):
13221: (4)             r""" Low-level wrapper for `::pdf_bound_annot()`."""
13222: (4)             return _mupdf.ll_pdf_bound_annot(annot)
13223: (0)         def ll_pdf_bound_page(page, box):
13224: (4)             r""" Low-level wrapper for `::pdf_bound_page()`."""
13225: (4)             return _mupdf.ll_pdf_bound_page(page, box)
13226: (0)         def ll_pdf_bound_widget(widget):
13227: (4)             r""" Low-level wrapper for `::pdf_bound_widget()`."""
13228: (4)             return _mupdf.ll_pdf_bound_widget(widget)
13229: (0)         def ll_pdf_button_field_on_state(field):
13230: (4)             r""" Low-level wrapper for `::pdf_button_field_on_state()`."""
13231: (4)             return _mupdf.ll_pdf_button_field_on_state(field)
13232: (0)         def ll_pdf_cache_object(doc, num):
13233: (4)             r"""
13234: (4)             Low-level wrapper for `::pdf_cache_object()`.
13235: (4)             Retrieve the pdf_xref_entry for a given object.
13236: (4)             This can cause xref reorganisations (solidifications etc) due to
13237: (4)             repairs, so all held pdf_xref_entries should be considered
13238: (4)             invalid after this call (other than the returned one).
13239: (4)             """
13240: (4)             return _mupdf.ll_pdf_cache_object(doc, num)
13241: (0)         def ll_pdf_calculate_form(doc):
13242: (4)             r""" Low-level wrapper for `::pdf_calculate_form()`."""
13243: (4)             return _mupdf.ll_pdf_calculate_form(doc)
13244: (0)         def ll_pdf_can_be_saved_incrementally(doc):
13245: (4)             r""" Low-level wrapper for `::pdf_can_be_saved_incrementally()`."""
13246: (4)             return _mupdf.ll_pdf_can_be_saved_incrementally(doc)
13247: (0)         def ll_pdf_can_redo(doc):
13248: (4)             r""" Low-level wrapper for `::pdf_can_redo()`."""
13249: (4)             return _mupdf.ll_pdf_can_redo(doc)
13250: (0)         def ll_pdf_can_undo(doc):
13251: (4)             r""" Low-level wrapper for `::pdf_can_undo()`."""
```

```
13252: (4)             return _mupdf.ll_pdf_can_undo(doc)
13253: (0)         def ll_pdf_check_certificate(verifier, doc, signature):
13254: (4)             r""" Low-level wrapper for `::pdf_check_certificate()`."""
13255: (4)             return _mupdf.ll_pdf_check_certificate(verifier, doc, signature)
13256: (0)         def ll_pdf_check_digest(verifier, doc, signature):
13257: (4)             r""" Low-level wrapper for `::pdf_check_digest()`."""
13258: (4)             return _mupdf.ll_pdf_check_digest(verifier, doc, signature)
13259: (0)         def ll_pdf_check_widget_certificate(verifier, widget):
13260: (4)             r""" Low-level wrapper for `::pdf_check_widget_certificate()`."""
13261: (4)             return _mupdf.ll_pdf_check_widget_certificate(verifier, widget)
13262: (0)         def ll_pdf_check_widget_digest(verifier, widget):
13263: (4)             r""" Low-level wrapper for `::pdf_check_widget_digest()`."""
13264: (4)             return _mupdf.ll_pdf_check_widget_digest(verifier, widget)
13265: (0)         def ll_pdf_choice_field_option(field, exportval, i):
13266: (4)             r""" Low-level wrapper for `::pdf_choice_field_option()`."""
13267: (4)             return _mupdf.ll_pdf_choice_field_option(field, exportval, i)
13268: (0)         def ll_pdf_choice_field_option_count(field):
13269: (4)             r""" Low-level wrapper for `::pdf_choice_field_option_count()`."""
13270: (4)             return _mupdf.ll_pdf_choice_field_option_count(field)
13271: (0)         def ll_pdf_choice_widget_is_multiselect(tw):
13272: (4)             r""" Low-level wrapper for `::pdf_choice_widget_is_multiselect()`."""
13273: (4)             return _mupdf.ll_pdf_choice_widget_is_multiselect(tw)
13274: (0)         def ll_pdf_choice_widget_options(tw, exportval, opts):
13275: (4)             r""" Low-level wrapper for `::pdf_choice_widget_options()`."""
13276: (4)             return _mupdf.ll_pdf_choice_widget_options(tw, exportval, opts)
13277: (0)         def ll_pdf_choice_widget_options2(tw, exportval):
13278: (4)             r"""
13279: (5)              Low-level wrapper for `::pdf_choice_widget_options2()`.  Swig-friendly
wrapper for pdf_choice_widget_options(), returns the
13280: (4)             options directly in a vector.
13281: (4)             """
13282: (4)             return _mupdf.ll_pdf_choice_widget_options2(tw, exportval)
13283: (0)         def ll_pdf_choice_widget_set_value(tw, n, opts):
13284: (4)             r""" Low-level wrapper for `::pdf_choice_widget_set_value()`."""
13285: (4)             return _mupdf.ll_pdf_choice_widget_set_value(tw, n, opts)
13286: (0)         def ll_pdf_choice_widget_value(tw, opts):
13287: (4)             r""" Low-level wrapper for `::pdf_choice_widget_value()`."""
13288: (4)             return _mupdf.ll_pdf_choice_widget_value(tw, opts)
13289: (0)         def ll_pdf_clean_file(infile, outfile, password, opts, retainlen):
13290: (4)             r""" Low-level wrapper for `::pdf_clean_file()`."""
13291: (4)             return _mupdf.ll_pdf_clean_file(infile, outfile, password, opts,
retainlen)
13292: (0)         def ll_pdf_clean_font_name(fontname):
13293: (4)             r""" Low-level wrapper for `::pdf_clean_font_name()`."""
13294: (4)             return _mupdf.ll_pdf_clean_font_name(fontname)
13295: (0)         def ll_pdf_clean_obj(obj):
13296: (4)             r""" Low-level wrapper for `::pdf_clean_obj()`."""
13297: (4)             return _mupdf.ll_pdf_clean_obj(obj)
13298: (0)         def ll_pdf_clear_annot_border_dash(annot):
13299: (4)             r""" Low-level wrapper for `::pdf_clear_annot_border_dash()`."""
13300: (4)             return _mupdf.ll_pdf_clear_annot_border_dash(annot)
13301: (0)         def ll_pdf_clear_annot_ink_list(annot):
13302: (4)             r""" Low-level wrapper for `::pdf_clear_annot_ink_list()`."""
13303: (4)             return _mupdf.ll_pdf_clear_annot_ink_list(annot)
13304: (0)         def ll_pdf_clear_annot_quad_points(annot):
13305: (4)             r""" Low-level wrapper for `::pdf_clear_annot_quad_points()`."""
13306: (4)             return _mupdf.ll_pdf_clear_annot_quad_points(annot)
13307: (0)         def ll_pdf_clear_annot_vertices(annot):
13308: (4)             r""" Low-level wrapper for `::pdf_clear_annot_vertices()`."""
13309: (4)             return _mupdf.ll_pdf_clear_annot_vertices(annot)
13310: (0)         def ll_pdf_clear_signature(widget):
13311: (4)             r""" Low-level wrapper for `::pdf_clear_signature()`."""
13312: (4)             return _mupdf.ll_pdf_clear_signature(widget)
13313: (0)         def ll_pdf_clear_xref(doc):
13314: (4)             r""" Low-level wrapper for `::pdf_clear_xref()`."""
13315: (4)             return _mupdf.ll_pdf_clear_xref(doc)
13316: (0)         def ll_pdf_clear_xref_to_mark(doc):
13317: (4)             r""" Low-level wrapper for `::pdf_clear_xref_to_mark()`."""
13318: (4)             return _mupdf.ll_pdf_clear_xref_to_mark(doc)
```

```
13319: (0)          def ll_pdf_close_processor(proc):
13320: (4)              r""" Low-level wrapper for `::pdf_close_processor()`."""
13321: (4)              return _mupdf.ll_pdf_close_processor(proc)
13322: (0)          def ll_pdf_cmap_size(cmap):
13323: (4)              r""" Low-level wrapper for `::pdf_cmap_size()`."""
13324: (4)              return _mupdf.ll_pdf_cmap_size(cmap)
13325: (0)          def ll_pdf_cmap_wmode(cmap):
13326: (4)              r""" Low-level wrapper for `::pdf_cmap_wmode()`."""
13327: (4)              return _mupdf.ll_pdf_cmap_wmode(cmap)
13328: (0)          def ll_pdf_copy_array(array):
13329: (4)              r""" Low-level wrapper for `::pdf_copy_array()`."""
13330: (4)              return _mupdf.ll_pdf_copy_array(array)
13331: (0)          def ll_pdf_copy_dict(dict):
13332: (4)              r""" Low-level wrapper for `::pdf_copy_dict()`."""
13333: (4)              return _mupdf.ll_pdf_copy_dict(dict)
13334: (0)          def ll_pdf_count_layer_config_ui(doc):
13335: (4)              r""" Low-level wrapper for `::pdf_count_layer_config_ui()`."""
13336: (4)              return _mupdf.ll_pdf_count_layer_config_ui(doc)
13337: (0)          def ll_pdf_count_layer_configs(doc):
13338: (4)              r""" Low-level wrapper for `::pdf_count_layer_configs()`."""
13339: (4)              return _mupdf.ll_pdf_count_layer_configs(doc)
13340: (0)          def ll_pdf_count_layers(doc):
13341: (4)              r""" Low-level wrapper for `::pdf_count_layers()`."""
13342: (4)              return _mupdf.ll_pdf_count_layers(doc)
13343: (0)          def ll_pdf_count_objects(doc):
13344: (4)              r""" Low-level wrapper for `::pdf_count_objects()`."""
13345: (4)              return _mupdf.ll_pdf_count_objects(doc)
13346: (0)          def ll_pdf_count_pages(doc):
13347: (4)              r""" Low-level wrapper for `::pdf_count_pages()`."""
13348: (4)              return _mupdf.ll_pdf_count_pages(doc)
13349: (0)          def ll_pdf_count_pages_imp(doc, chapter):
13350: (4)              r""" Low-level wrapper for `::pdf_count_pages_imp()`."""
13351: (4)              return _mupdf.ll_pdf_count_pages_imp(doc, chapter)
13352: (0)          def ll_pdf_count_q_balance(doc, res, stm, prepend, append):
13353: (4)              r"""
13354: (5)               Low-level wrapper for `::pdf_count_q_balance()`.
13355: (4)              This function has out-params. Python/C# wrappers look like:
13356: (8)                  `ll_pdf_count_q_balance(::pdf_document *doc, ::pdf_obj *res, ::pdf_obj
*stm)` => `(int prepend, int append)`
13357: (4)              """
13358: (4)              return _mupdf.ll_pdf_count_q_balance(doc, res, stm, prepend, append)
13359: (0)          def ll_pdf_count_signatures(doc):
13360: (4)              r""" Low-level wrapper for `::pdf_count_signatures()`."""
13361: (4)              return _mupdf.ll_pdf_count_signatures(doc)
13362: (0)          def ll_pdf_count_unsaved_versions(doc):
13363: (4)              r""" Low-level wrapper for `::pdf_count_unsaved_versions()`."""
13364: (4)              return _mupdf.ll_pdf_count_unsaved_versions(doc)
13365: (0)          def ll_pdf_count_versions(doc):
13366: (4)              r""" Low-level wrapper for `::pdf_count_versions()`."""
13367: (4)              return _mupdf.ll_pdf_count_versions(doc)
13368: (0)          def ll_pdf_create_annot(page, type):
13369: (4)              r""" Low-level wrapper for `::pdf_create_annot()`."""
13370: (4)              return _mupdf.ll_pdf_create_annot(page, type)
13371: (0)          def ll_pdf_create_annot_raw(page, type):
13372: (4)              r""" Low-level wrapper for `::pdf_create_annot_raw()`."""
13373: (4)              return _mupdf.ll_pdf_create_annot_raw(page, type)
13374: (0)          def ll_pdf_create_document():
13375: (4)              r""" Low-level wrapper for `::pdf_create_document()`."""
13376: (4)              return _mupdf.ll_pdf_create_document()
13377: (0)          def ll_pdf_create_field_name(doc, prefix, buf, len):
13378: (4)              r""" Low-level wrapper for `::pdf_create_field_name()`."""
13379: (4)              return _mupdf.ll_pdf_create_field_name(doc, prefix, buf, len)
13380: (0)          def ll_pdf_create_link(page, bbox, uri):
13381: (4)              r""" Low-level wrapper for `::pdf_create_link()`."""
13382: (4)              return _mupdf.ll_pdf_create_link(page, bbox, uri)
13383: (0)          def ll_pdf_create_object(doc):
13384: (4)              r""" Low-level wrapper for `::pdf_create_object()`."""
13385: (4)              return _mupdf.ll_pdf_create_object(doc)
13386: (0)          def ll_pdf_create_signature_widget(page, name):
```

```
13387: (4)             r""" Low-level wrapper for `::pdf_create_signature_widget()`."""
13388: (4)             return _mupdf.ll_pdf_create_signature_widget(page, name)
13389: (0)         def ll_pdf_crypt_encrypt_metadata(crypt):
13390: (4)             r""" Low-level wrapper for `::pdf_crypt_encrypt_metadata()`."""
13391: (4)             return _mupdf.ll_pdf_crypt_encrypt_metadata(crypt)
13392: (0)         def ll_pdf_crypt_key(crypt):
13393: (4)             r""" Low-level wrapper for `::pdf_crypt_key()`."""
13394: (4)             return _mupdf.ll_pdf_crypt_key(crypt)
13395: (0)         def ll_pdf_crypt_length(crypt):
13396: (4)             r""" Low-level wrapper for `::pdf_crypt_length()`."""
13397: (4)             return _mupdf.ll_pdf_crypt_length(crypt)
13398: (0)         def ll_pdf_crypt_method(crypt):
13399: (4)             r""" Low-level wrapper for `::pdf_crypt_method()`."""
13400: (4)             return _mupdf.ll_pdf_crypt_method(crypt)
13401: (0)         def ll_pdf_crypt_obj(crypt, obj, num, gen):
13402: (4)             r""" Low-level wrapper for `::pdf_crypt_obj()`."""
13403: (4)             return _mupdf.ll_pdf_crypt_obj(crypt, obj, num, gen)
13404: (0)         def ll_pdf_crypt_owner_encryption(crypt):
13405: (4)             r""" Low-level wrapper for `::pdf_crypt_owner_encryption()`."""
13406: (4)             return _mupdf.ll_pdf_crypt_owner_encryption(crypt)
13407: (0)         def ll_pdf_crypt_owner_password(crypt):
13408: (4)             r""" Low-level wrapper for `::pdf_crypt_owner_password()`."""
13409: (4)             return _mupdf.ll_pdf_crypt_owner_password(crypt)
13410: (0)         def ll_pdf_crypt_permissions(crypt):
13411: (4)             r""" Low-level wrapper for `::pdf_crypt_permissions()`."""
13412: (4)             return _mupdf.ll_pdf_crypt_permissions(crypt)
13413: (0)         def ll_pdf_crypt_permissions_encryption(crypt):
13414: (4)             r""" Low-level wrapper for `::pdf_crypt_permissions_encryption()`."""
13415: (4)             return _mupdf.ll_pdf_crypt_permissions_encryption(crypt)
13416: (0)         def ll_pdf_crypt_revision(crypt):
13417: (4)             r""" Low-level wrapper for `::pdf_crypt_revision()`."""
13418: (4)             return _mupdf.ll_pdf_crypt_revision(crypt)
13419: (0)         def ll_pdf_crypt_stream_method(crypt):
13420: (4)             r""" Low-level wrapper for `::pdf_crypt_stream_method()`."""
13421: (4)             return _mupdf.ll_pdf_crypt_stream_method(crypt)
13422: (0)         def ll_pdf_crypt_string_method(crypt):
13423: (4)             r""" Low-level wrapper for `::pdf_crypt_string_method()`."""
13424: (4)             return _mupdf.ll_pdf_crypt_string_method(crypt)
13425: (0)         def ll_pdf_crypt_user_encryption(crypt):
13426: (4)             r""" Low-level wrapper for `::pdf_crypt_user_encryption()`."""
13427: (4)             return _mupdf.ll_pdf_crypt_user_encryption(crypt)
13428: (0)         def ll_pdf_crypt_user_password(crypt):
13429: (4)             r""" Low-level wrapper for `::pdf_crypt_user_password()`."""
13430: (4)             return _mupdf.ll_pdf_crypt_user_password(crypt)
13431: (0)         def ll_pdf_crypt_version(crypt):
13432: (4)             r""" Low-level wrapper for `::pdf_crypt_version()`."""
13433: (4)             return _mupdf.ll_pdf_crypt_version(crypt)
13434: (0)         def ll_pdf_cycle(here, prev, obj):
13435: (4)             r""" Low-level wrapper for `::pdf_cycle()`."""
13436: (4)             return _mupdf.ll_pdf_cycle(here, prev, obj)
13437: (0)         def ll_pdf_debug_doc_changes(doc):
13438: (4)             r""" Low-level wrapper for `::pdf_debug_doc_changes()`."""
13439: (4)             return _mupdf.ll_pdf_debug_doc_changes(doc)
13440: (0)         def ll_pdf_debug_obj(obj):
13441: (4)             r""" Low-level wrapper for `::pdf_debug_obj()`."""
13442: (4)             return _mupdf.ll_pdf_debug_obj(obj)
13443: (0)         def ll_pdf_debug_ref(obj):
13444: (4)             r""" Low-level wrapper for `::pdf_debug_ref()`."""
13445: (4)             return _mupdf.ll_pdf_debug_ref(obj)
13446: (0)         def ll_pdf_decode_cmap(cmap, s, e, cpt):
13447: (4)             r"""
13448: (5)              Low-level wrapper for `::pdf_decode_cmap()`.
13449: (4)             This function has out-params. Python/C# wrappers look like:
13450: (8)                 `ll_pdf_decode_cmap(::pdf_cmap *cmap, unsigned char *s, unsigned char
*e)` => `(int, unsigned int cpt)`
13451: (4)             """
13452: (4)             return _mupdf.ll_pdf_decode_cmap(cmap, s, e, cpt)
13453: (0)         def ll_pdf_deep_copy_obj(obj):
13454: (4)             r""" Low-level wrapper for `::pdf_deep_copy_obj()`."""
```

```
13455: (4)             return _mupdf.ll_pdf_deep_copy_obj(obj)
13456: (0)         def ll_pdf_delete_annot(page, annot):
13457: (4)             r""" Low-level wrapper for `::pdf_delete_annot()`."""
13458: (4)             return _mupdf.ll_pdf_delete_annot(page, annot)
13459: (0)         def ll_pdf_delete_link(page, link):
13460: (4)             r""" Low-level wrapper for `::pdf_delete_link()`."""
13461: (4)             return _mupdf.ll_pdf_delete_link(page, link)
13462: (0)         def ll_pdf_delete_object(doc, num):
13463: (4)             r""" Low-level wrapper for `::pdf_delete_object()`."""
13464: (4)             return _mupdf.ll_pdf_delete_object(doc, num)
13465: (0)         def ll_pdf_delete_page(doc, number):
13466: (4)             r""" Low-level wrapper for `::pdf_delete_page()`."""
13467: (4)             return _mupdf.ll_pdf_delete_page(doc, number)
13468: (0)         def ll_pdf_delete_page_labels(doc, index):
13469: (4)             r""" Low-level wrapper for `::pdf_delete_page_labels()`."""
13470: (4)             return _mupdf.ll_pdf_delete_page_labels(doc, index)
13471: (0)         def ll_pdf_delete_page_range(doc, start, end):
13472: (4)             r""" Low-level wrapper for `::pdf_delete_page_range()`."""
13473: (4)             return _mupdf.ll_pdf_delete_page_range(doc, start, end)
13474: (0)         def ll_pdf_deselect_layer_config_ui(doc, ui):
13475: (4)             r""" Low-level wrapper for `::pdf_deselect_layer_config_ui()`."""
13476: (4)             return _mupdf.ll_pdf_deselect_layer_config_ui(doc, ui)
13477: (0)         def ll_pdf_deserialise_journal(doc, stm):
13478: (4)             r""" Low-level wrapper for `::pdf_deserialise_journal()`."""
13479: (4)             return _mupdf.ll_pdf_deserialise_journal(doc, stm)
13480: (0)         def ll_pdf_dict_del(dict, key):
13481: (4)             r""" Low-level wrapper for `::pdf_dict_del()`."""
13482: (4)             return _mupdf.ll_pdf_dict_del(dict, key)
13483: (0)         def ll_pdf_dict_dels(dict, key):
13484: (4)             r""" Low-level wrapper for `::pdf_dict_dels()`."""
13485: (4)             return _mupdf.ll_pdf_dict_dels(dict, key)
13486: (0)         def ll_pdf_dict_get(dict, key):
13487: (4)             r""" Low-level wrapper for `::pdf_dict_get()`."""
13488: (4)             return _mupdf.ll_pdf_dict_get(dict, key)
13489: (0)         def ll_pdf_dict_get_bool(dict, key):
13490: (4)             r""" Low-level wrapper for `::pdf_dict_get_bool()`."""
13491: (4)             return _mupdf.ll_pdf_dict_get_bool(dict, key)
13492: (0)         def ll_pdf_dict_get_bool_default(dict, key, _def):
13493: (4)             r""" Low-level wrapper for `::pdf_dict_get_bool_default()`."""
13494: (4)             return _mupdf.ll_pdf_dict_get_bool_default(dict, key, _def)
13495: (0)         def ll_pdf_dict_get_date(dict, key):
13496: (4)             r""" Low-level wrapper for `::pdf_dict_get_date()`."""
13497: (4)             return _mupdf.ll_pdf_dict_get_date(dict, key)
13498: (0)         def ll_pdf_dict_get_inheritable(dict, key):
13499: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable()`."""
13500: (4)             return _mupdf.ll_pdf_dict_get_inheritable(dict, key)
13501: (0)         def ll_pdf_dict_get_inheritable_bool(dict, key):
13502: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_bool()`."""
13503: (4)             return _mupdf.ll_pdf_dict_get_inheritable_bool(dict, key)
13504: (0)         def ll_pdf_dict_get_inheritable_date(dict, key):
13505: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_date()`."""
13506: (4)             return _mupdf.ll_pdf_dict_get_inheritable_date(dict, key)
13507: (0)         def ll_pdf_dict_get_inheritable_int(dict, key):
13508: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_int()`."""
13509: (4)             return _mupdf.ll_pdf_dict_get_inheritable_int(dict, key)
13510: (0)         def ll_pdf_dict_get_inheritable_int64(dict, key):
13511: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_int64()`."""
13512: (4)             return _mupdf.ll_pdf_dict_get_inheritable_int64(dict, key)
13513: (0)         def ll_pdf_dict_get_inheritable_matrix(dict, key):
13514: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_matrix()`."""
13515: (4)             return _mupdf.ll_pdf_dict_get_inheritable_matrix(dict, key)
13516: (0)         def ll_pdf_dict_get_inheritable_name(dict, key):
13517: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_name()`."""
13518: (4)             return _mupdf.ll_pdf_dict_get_inheritable_name(dict, key)
13519: (0)         def ll_pdf_dict_get_inheritable_real(dict, key):
13520: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_real()`."""
13521: (4)             return _mupdf.ll_pdf_dict_get_inheritable_real(dict, key)
13522: (0)         def ll_pdf_dict_get_inheritable_rect(dict, key):
13523: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_rect()`."""
```

```
13524: (4)             return _mupdf.ll_pdf_dict_get_inheritable_rect(dict, key)
13525: (0)         def ll_pdf_dict_get_inheritable_string(dict, key, sizep):
13526: (4)             r"""
13527: (5)              Low-level wrapper for `::pdf_dict_get_inheritable_string()`.
13528: (4)             This function has out-params. Python/C# wrappers look like:
13529: (8)                 `ll_pdf_dict_get_inheritable_string(::pdf_obj *dict, ::pdf_obj *key)`
=> `(const char *, size_t sizep)`
13530: (4)             """
13531: (4)             return _mupdf.ll_pdf_dict_get_inheritable_string(dict, key, sizep)
13532: (0)         def ll_pdf_dict_get_inheritable_text_string(dict, key):
13533: (4)             r""" Low-level wrapper for `::pdf_dict_get_inheritable_text_string()`."""
13534: (4)             return _mupdf.ll_pdf_dict_get_inheritable_text_string(dict, key)
13535: (0)         def ll_pdf_dict_get_int(dict, key):
13536: (4)             r""" Low-level wrapper for `::pdf_dict_get_int()`."""
13537: (4)             return _mupdf.ll_pdf_dict_get_int(dict, key)
13538: (0)         def ll_pdf_dict_get_int64(dict, key):
13539: (4)             r""" Low-level wrapper for `::pdf_dict_get_int64()`."""
13540: (4)             return _mupdf.ll_pdf_dict_get_int64(dict, key)
13541: (0)         def ll_pdf_dict_get_int_default(dict, key, _def):
13542: (4)             r""" Low-level wrapper for `::pdf_dict_get_int_default()`."""
13543: (4)             return _mupdf.ll_pdf_dict_get_int_default(dict, key, _def)
13544: (0)         def ll_pdf_dict_get_key(dict, idx):
13545: (4)             r""" Low-level wrapper for `::pdf_dict_get_key()`."""
13546: (4)             return _mupdf.ll_pdf_dict_get_key(dict, idx)
13547: (0)         def ll_pdf_dict_get_matrix(dict, key):
13548: (4)             r""" Low-level wrapper for `::pdf_dict_get_matrix()`."""
13549: (4)             return _mupdf.ll_pdf_dict_get_matrix(dict, key)
13550: (0)         def ll_pdf_dict_get_name(dict, key):
13551: (4)             r""" Low-level wrapper for `::pdf_dict_get_name()`."""
13552: (4)             return _mupdf.ll_pdf_dict_get_name(dict, key)
13553: (0)         def ll_pdf_dict_get_put_drop(dict, key, val, old_val):
13554: (4)             r"""
13555: (5)              Low-level wrapper for `::pdf_dict_get_put_drop()`.
13556: (4)             This function has out-params. Python/C# wrappers look like:
13557: (8)                 `ll_pdf_dict_get_put_drop(::pdf_obj *dict, ::pdf_obj *key, ::pdf_obj
*val, ::pdf_obj **old_val)` =>
13558: (4)             """
13559: (4)             return _mupdf.ll_pdf_dict_get_put_drop(dict, key, val, old_val)
13560: (0)         def ll_pdf_dict_get_real(dict, key):
13561: (4)             r""" Low-level wrapper for `::pdf_dict_get_real()`."""
13562: (4)             return _mupdf.ll_pdf_dict_get_real(dict, key)
13563: (0)         def ll_pdf_dict_get_real_default(dict, key, _def):
13564: (4)             r""" Low-level wrapper for `::pdf_dict_get_real_default()`."""
13565: (4)             return _mupdf.ll_pdf_dict_get_real_default(dict, key, _def)
13566: (0)         def ll_pdf_dict_get_rect(dict, key):
13567: (4)             r""" Low-level wrapper for `::pdf_dict_get_rect()`."""
13568: (4)             return _mupdf.ll_pdf_dict_get_rect(dict, key)
13569: (0)         def ll_pdf_dict_get_string(dict, key, sizep):
13570: (4)             r"""
13571: (5)              Low-level wrapper for `::pdf_dict_get_string()`.
13572: (4)             This function has out-params. Python/C# wrappers look like:
13573: (8)                 `ll_pdf_dict_get_string(::pdf_obj *dict, ::pdf_obj *key)` => `(const
char *, size_t sizep)`
13574: (4)             """
13575: (4)             return _mupdf.ll_pdf_dict_get_string(dict, key, sizep)
13576: (0)         def ll_pdf_dict_get_text_string(dict, key):
13577: (4)             r""" Low-level wrapper for `::pdf_dict_get_text_string()`."""
13578: (4)             return _mupdf.ll_pdf_dict_get_text_string(dict, key)
13579: (0)         def ll_pdf_dict_get_val(dict, idx):
13580: (4)             r""" Low-level wrapper for `::pdf_dict_get_val()`."""
13581: (4)             return _mupdf.ll_pdf_dict_get_val(dict, idx)
13582: (0)         def ll_pdf_dict_geta(dict, key, abbrev):
13583: (4)             r""" Low-level wrapper for `::pdf_dict_geta()`."""
13584: (4)             return _mupdf.ll_pdf_dict_geta(dict, key, abbrev)
13585: (0)         def ll_pdf_dict_getp(dict, path):
13586: (4)             r""" Low-level wrapper for `::pdf_dict_getp()`."""
13587: (4)             return _mupdf.ll_pdf_dict_getp(dict, path)
13588: (0)         def ll_pdf_dict_getp_inheritable(dict, path):
13589: (4)             r""" Low-level wrapper for `::pdf_dict_getp_inheritable()`."""
```

```
13590: (4)             return _mupdf.ll_pdf_dict_getp_inheritable(dict, path)
13591: (0)         def ll_pdf_dict_gets(dict, key):
13592: (4)             r""" Low-level wrapper for `::pdf_dict_gets()`."""
13593: (4)             return _mupdf.ll_pdf_dict_gets(dict, key)
13594: (0)         def ll_pdf_dict_gets_inheritable(dict, key):
13595: (4)             r""" Low-level wrapper for `::pdf_dict_gets_inheritable()`."""
13596: (4)             return _mupdf.ll_pdf_dict_gets_inheritable(dict, key)
13597: (0)         def ll_pdf_dict_getsa(dict, key, abbrev):
13598: (4)             r""" Low-level wrapper for `::pdf_dict_getsa()`."""
13599: (4)             return _mupdf.ll_pdf_dict_getsa(dict, key, abbrev)
13600: (0)         def ll_pdf_dict_len(dict):
13601: (4)             r""" Low-level wrapper for `::pdf_dict_len()`."""
13602: (4)             return _mupdf.ll_pdf_dict_len(dict)
13603: (0)         def ll_pdf_dict_put(dict, key, val):
13604: (4)             r""" Low-level wrapper for `::pdf_dict_put()`."""
13605: (4)             return _mupdf.ll_pdf_dict_put(dict, key, val)
13606: (0)         def ll_pdf_dict_put_array(dict, key, initial):
13607: (4)             r""" Low-level wrapper for `::pdf_dict_put_array()`."""
13608: (4)             return _mupdf.ll_pdf_dict_put_array(dict, key, initial)
13609: (0)         def ll_pdf_dict_put_bool(dict, key, x):
13610: (4)             r""" Low-level wrapper for `::pdf_dict_put_bool()`."""
13611: (4)             return _mupdf.ll_pdf_dict_put_bool(dict, key, x)
13612: (0)         def ll_pdf_dict_put_date(dict, key, time):
13613: (4)             r""" Low-level wrapper for `::pdf_dict_put_date()`."""
13614: (4)             return _mupdf.ll_pdf_dict_put_date(dict, key, time)
13615: (0)         def ll_pdf_dict_put_dict(dict, key, initial):
13616: (4)             r""" Low-level wrapper for `::pdf_dict_put_dict()`."""
13617: (4)             return _mupdf.ll_pdf_dict_put_dict(dict, key, initial)
13618: (0)         def ll_pdf_dict_put_drop(dict, key, val):
13619: (4)             r""" Low-level wrapper for `::pdf_dict_put_drop()`."""
13620: (4)             return _mupdf.ll_pdf_dict_put_drop(dict, key, val)
13621: (0)         def ll_pdf_dict_put_int(dict, key, x):
13622: (4)             r""" Low-level wrapper for `::pdf_dict_put_int()`."""
13623: (4)             return _mupdf.ll_pdf_dict_put_int(dict, key, x)
13624: (0)         def ll_pdf_dict_put_matrix(dict, key, x):
13625: (4)             r""" Low-level wrapper for `::pdf_dict_put_matrix()`."""
13626: (4)             return _mupdf.ll_pdf_dict_put_matrix(dict, key, x)
13627: (0)         def ll_pdf_dict_put_name(dict, key, x):
13628: (4)             r""" Low-level wrapper for `::pdf_dict_put_name()`."""
13629: (4)             return _mupdf.ll_pdf_dict_put_name(dict, key, x)
13630: (0)         def ll_pdf_dict_put_real(dict, key, x):
13631: (4)             r""" Low-level wrapper for `::pdf_dict_put_real()`."""
13632: (4)             return _mupdf.ll_pdf_dict_put_real(dict, key, x)
13633: (0)         def ll_pdf_dict_put_rect(dict, key, x):
13634: (4)             r""" Low-level wrapper for `::pdf_dict_put_rect()`."""
13635: (4)             return _mupdf.ll_pdf_dict_put_rect(dict, key, x)
13636: (0)         def ll_pdf_dict_put_string(dict, key, x, n):
13637: (4)             r""" Low-level wrapper for `::pdf_dict_put_string()`."""
13638: (4)             return _mupdf.ll_pdf_dict_put_string(dict, key, x, n)
13639: (0)         def ll_pdf_dict_put_text_string(dict, key, x):
13640: (4)             r""" Low-level wrapper for `::pdf_dict_put_text_string()`."""
13641: (4)             return _mupdf.ll_pdf_dict_put_text_string(dict, key, x)
13642: (0)         def ll_pdf_dict_put_val_null(obj, idx):
13643: (4)             r""" Low-level wrapper for `::pdf_dict_put_val_null()`."""
13644: (4)             return _mupdf.ll_pdf_dict_put_val_null(obj, idx)
13645: (0)         def ll_pdf_dict_putp(dict, path, val):
13646: (4)             r""" Low-level wrapper for `::pdf_dict_putp()`."""
13647: (4)             return _mupdf.ll_pdf_dict_putp(dict, path, val)
13648: (0)         def ll_pdf_dict_putp_drop(dict, path, val):
13649: (4)             r""" Low-level wrapper for `::pdf_dict_putp_drop()`."""
13650: (4)             return _mupdf.ll_pdf_dict_putp_drop(dict, path, val)
13651: (0)         def ll_pdf_dict_puts(dict, key, val):
13652: (4)             r""" Low-level wrapper for `::pdf_dict_puts()`."""
13653: (4)             return _mupdf.ll_pdf_dict_puts(dict, key, val)
13654: (0)         def ll_pdf_dict_puts_dict(dict, key, initial):
13655: (4)             r""" Low-level wrapper for `::pdf_dict_puts_dict()`."""
13656: (4)             return _mupdf.ll_pdf_dict_puts_dict(dict, key, initial)
13657: (0)         def ll_pdf_dict_puts_drop(dict, key, val):
13658: (4)             r""" Low-level wrapper for `::pdf_dict_puts_drop()`."""
```

```
13659: (4)             return _mupdf.ll_pdf_dict_puts_drop(dict, key, val)
13660: (0)         def ll_pdf_dirty_annot(annot):
13661: (4)             r""" Low-level wrapper for `::pdf_dirty_annot()`."""
13662: (4)             return _mupdf.ll_pdf_dirty_annot(annot)
13663: (0)         def ll_pdf_dirty_obj(obj):
13664: (4)             r""" Low-level wrapper for `::pdf_dirty_obj()`."""
13665: (4)             return _mupdf.ll_pdf_dirty_obj(obj)
13666: (0)         def ll_pdf_disable_js(doc):
13667: (4)             r""" Low-level wrapper for `::pdf_disable_js()`."""
13668: (4)             return _mupdf.ll_pdf_disable_js(doc)
13669: (0)         def ll_pdf_discard_journal(journal):
13670: (4)             r""" Low-level wrapper for `::pdf_discard_journal()`."""
13671: (4)             return _mupdf.ll_pdf_discard_journal(journal)
13672: (0)         def ll_pdf_doc_was_linearized(doc):
13673: (4)             r""" Low-level wrapper for `::pdf_doc_was_linearized()`."""
13674: (4)             return _mupdf.ll_pdf_doc_was_linearized(doc)
13675: (0)         def ll_pdf_document_event_did_print(doc):
13676: (4)             r""" Low-level wrapper for `::pdf_document_event_did_print()`."""
13677: (4)             return _mupdf.ll_pdf_document_event_did_print(doc)
13678: (0)         def ll_pdf_document_event_did_save(doc):
13679: (4)             r""" Low-level wrapper for `::pdf_document_event_did_save()`."""
13680: (4)             return _mupdf.ll_pdf_document_event_did_save(doc)
13681: (0)         def ll_pdf_document_event_will_close(doc):
13682: (4)             r""" Low-level wrapper for `::pdf_document_event_will_close()`."""
13683: (4)             return _mupdf.ll_pdf_document_event_will_close(doc)
13684: (0)         def ll_pdf_document_event_will_print(doc):
13685: (4)             r""" Low-level wrapper for `::pdf_document_event_will_print()`."""
13686: (4)             return _mupdf.ll_pdf_document_event_will_print(doc)
13687: (0)         def ll_pdf_document_event_will_save(doc):
13688: (4)             r""" Low-level wrapper for `::pdf_document_event_will_save()`."""
13689: (4)             return _mupdf.ll_pdf_document_event_will_save(doc)
13690: (0)         def ll_pdf_document_from_fz_document(ptr):
13691: (4)             r""" Low-level wrapper for `::pdf_document_from_fz_document()`."""
13692: (4)             return _mupdf.ll_pdf_document_from_fz_document(ptr)
13693: (0)         def ll_pdf_document_language(doc):
13694: (4)             r""" Low-level wrapper for `::pdf_document_language()`."""
13695: (4)             return _mupdf.ll_pdf_document_language(doc)
13696: (0)         def ll_pdf_document_output_intent(doc):
13697: (4)             r""" Low-level wrapper for `::pdf_document_output_intent()`."""
13698: (4)             return _mupdf.ll_pdf_document_output_intent(doc)
13699: (0)         def ll_pdf_document_permissions(doc):
13700: (4)             r""" Low-level wrapper for `::pdf_document_permissions()`."""
13701: (4)             return _mupdf.ll_pdf_document_permissions(doc)
13702: (0)         def ll_pdf_drop_annot(annot):
13703: (4)             r""" Low-level wrapper for `::pdf_drop_annot()`."""
13704: (4)             return _mupdf.ll_pdf_drop_annot(annot)
13705: (0)         def ll_pdf_drop_cmap(cmap):
13706: (4)             r""" Low-level wrapper for `::pdf_drop_cmap()`."""
13707: (4)             return _mupdf.ll_pdf_drop_cmap(cmap)
13708: (0)         def ll_pdf_drop_cmap_imp(cmap):
13709: (4)             r""" Low-level wrapper for `::pdf_drop_cmap_imp()`."""
13710: (4)             return _mupdf.ll_pdf_drop_cmap_imp(cmap)
13711: (0)         def ll_pdf_drop_crypt(crypt):
13712: (4)             r""" Low-level wrapper for `::pdf_drop_crypt()`."""
13713: (4)             return _mupdf.ll_pdf_drop_crypt(crypt)
13714: (0)         def ll_pdf_drop_document(doc):
13715: (4)             r""" Low-level wrapper for `::pdf_drop_document()`."""
13716: (4)             return _mupdf.ll_pdf_drop_document(doc)
13717: (0)         def ll_pdf_drop_font(font):
13718: (4)             r""" Low-level wrapper for `::pdf_drop_font()`."""
13719: (4)             return _mupdf.ll_pdf_drop_font(font)
13720: (0)         def ll_pdf_drop_function(func):
13721: (4)             r""" Low-level wrapper for `::pdf_drop_function()`."""
13722: (4)             return _mupdf.ll_pdf_drop_function(func)
13723: (0)         def ll_pdf_drop_graft_map(map):
13724: (4)             r""" Low-level wrapper for `::pdf_drop_graft_map()`."""
13725: (4)             return _mupdf.ll_pdf_drop_graft_map(map)
13726: (0)         def ll_pdf_drop_journal(journal):
13727: (4)             r""" Low-level wrapper for `::pdf_drop_journal()`."""
```

```
13728: (4)             return _mupdf.ll_pdf_drop_journal(journal)
13729: (0)         def ll_pdf_drop_js(js):
13730: (4)             r""" Low-level wrapper for `::pdf_drop_js()`."""
13731: (4)             return _mupdf.ll_pdf_drop_js(js)
13732: (0)         def ll_pdf_drop_local_xref(xref):
13733: (4)             r""" Low-level wrapper for `::pdf_drop_local_xref()`."""
13734: (4)             return _mupdf.ll_pdf_drop_local_xref(xref)
13735: (0)         def ll_pdf_drop_local_xref_and_resources(doc):
13736: (4)             r""" Low-level wrapper for `::pdf_drop_local_xref_and_resources()`."""
13737: (4)             return _mupdf.ll_pdf_drop_local_xref_and_resources(doc)
13738: (0)         def ll_pdf_drop_locked_fields(locked):
13739: (4)             r""" Low-level wrapper for `::pdf_drop_locked_fields()`."""
13740: (4)             return _mupdf.ll_pdf_drop_locked_fields(locked)
13741: (0)         def ll_pdf_drop_mark_bits(marks):
13742: (4)             r""" Low-level wrapper for `::pdf_drop_mark_bits()`."""
13743: (4)             return _mupdf.ll_pdf_drop_mark_bits(marks)
13744: (0)         def ll_pdf_drop_obj(obj):
13745: (4)             r""" Low-level wrapper for `::pdf_drop_obj()`."""
13746: (4)             return _mupdf.ll_pdf_drop_obj(obj)
13747: (0)         def ll_pdf_drop_ocg(doc):
13748: (4)             r""" Low-level wrapper for `::pdf_drop_ocg()`."""
13749: (4)             return _mupdf.ll_pdf_drop_ocg(doc)
13750: (0)         def ll_pdf_drop_page(page):
13751: (4)             r""" Low-level wrapper for `::pdf_drop_page()`."""
13752: (4)             return _mupdf.ll_pdf_drop_page(page)
13753: (0)         def ll_pdf_drop_page_tree(doc):
13754: (4)             r""" Low-level wrapper for `::pdf_drop_page_tree()`."""
13755: (4)             return _mupdf.ll_pdf_drop_page_tree(doc)
13756: (0)         def ll_pdf_drop_page_tree_internal(doc):
13757: (4)             r""" Low-level wrapper for `::pdf_drop_page_tree_internal()`."""
13758: (4)             return _mupdf.ll_pdf_drop_page_tree_internal(doc)
13759: (0)         def ll_pdf_drop_pattern(pat):
13760: (4)             r""" Low-level wrapper for `::pdf_drop_pattern()`."""
13761: (4)             return _mupdf.ll_pdf_drop_pattern(pat)
13762: (0)         def ll_pdf_drop_processor(proc):
13763: (4)             r""" Low-level wrapper for `::pdf_drop_processor()`."""
13764: (4)             return _mupdf.ll_pdf_drop_processor(proc)
13765: (0)         def ll_pdf_drop_resource_tables(doc):
13766: (4)             r""" Low-level wrapper for `::pdf_drop_resource_tables()`."""
13767: (4)             return _mupdf.ll_pdf_drop_resource_tables(doc)
13768: (0)         def ll_pdf_drop_signer(signer):
13769: (4)             r""" Low-level wrapper for `::pdf_drop_signer()`."""
13770: (4)             return _mupdf.ll_pdf_drop_signer(signer)
13771: (0)         def ll_pdf_drop_singleton_obj(obj):
13772: (4)             r""" Low-level wrapper for `::pdf_drop_singleton_obj()`."""
13773: (4)             return _mupdf.ll_pdf_drop_singleton_obj(obj)
13774: (0)         def ll_pdf_drop_verifier(verifier):
13775: (4)             r""" Low-level wrapper for `::pdf_drop_verifier()`."""
13776: (4)             return _mupdf.ll_pdf_drop_verifier(verifier)
13777: (0)         def ll_pdf_drop_widget(widget):
13778: (4)             r""" Low-level wrapper for `::pdf_drop_widget()`."""
13779: (4)             return _mupdf.ll_pdf_drop_widget(widget)
13780: (0)         def ll_pdf_edit_text_field_value(widget, value, change, selStart, selEnd,
newvalue):
13781: (4)             r"""
13782: (5)              Low-level wrapper for `::pdf_edit_text_field_value()`.
13783: (4)             This function has out-params. Python/C# wrappers look like:
13784: (8)                 `ll_pdf_edit_text_field_value(::pdf_annot *widget, const char *value,
const char *change)` => `(int, int selStart, int selEnd, char *newvalue)`
13785: (4)             """
13786: (4)             return _mupdf.ll_pdf_edit_text_field_value(widget, value, change,
selStart, selEnd, newvalue)
13787: (0)         def ll_pdf_empty_store(doc):
13788: (4)             r""" Low-level wrapper for `::pdf_empty_store()`."""
13789: (4)             return _mupdf.ll_pdf_empty_store(doc)
13790: (0)         def ll_pdf_enable_journal(doc):
13791: (4)             r""" Low-level wrapper for `::pdf_enable_journal()`."""
13792: (4)             return _mupdf.ll_pdf_enable_journal(doc)
13793: (0)         def ll_pdf_enable_js(doc):
```

```
13794: (4)             r""" Low-level wrapper for `::pdf_enable_js()`."""
13795: (4)             return _mupdf.ll_pdf_enable_js(doc)
13796: (0)         def ll_pdf_enable_layer(doc, layer, enabled):
13797: (4)             r""" Low-level wrapper for `::pdf_enable_layer()`."""
13798: (4)             return _mupdf.ll_pdf_enable_layer(doc, layer, enabled)
13799: (0)         def ll_pdf_encrypt_data(crypt, num, gen, fmt_str_out, arg, s, n):
13800: (4)             r""" Low-level wrapper for `::pdf_encrypt_data()`."""
13801: (4)             return _mupdf.ll_pdf_encrypt_data(crypt, num, gen, fmt_str_out, arg, s, n)
13802: (0)         def ll_pdf_encrypted_len(crypt, num, gen, len):
13803: (4)             r""" Low-level wrapper for `::pdf_encrypted_len()`."""
13804: (4)             return _mupdf.ll_pdf_encrypted_len(crypt, num, gen, len)
13805: (0)         def ll_pdf_end_hmtx(font):
13806: (4)             r""" Low-level wrapper for `::pdf_end_hmtx()`."""
13807: (4)             return _mupdf.ll_pdf_end_hmtx(font)
13808: (0)         def ll_pdf_end_operation(doc):
13809: (4)             r""" Low-level wrapper for `::pdf_end_operation()`."""
13810: (4)             return _mupdf.ll_pdf_end_operation(doc)
13811: (0)         def ll_pdf_end_vmtx(font):
13812: (4)             r""" Low-level wrapper for `::pdf_end_vmtx()`."""
13813: (4)             return _mupdf.ll_pdf_end_vmtx(font)
13814: (0)         def ll_pdf_ensure_solid_xref(doc, num):
13815: (4)             r""" Low-level wrapper for `::pdf_ensure_solid_xref()`."""
13816: (4)             return _mupdf.ll_pdf_ensure_solid_xref(doc, num)
13817: (0)         def ll_pdf_eval_function(func, _in, inlen, out, outlen):
13818: (4)             r"""
13819: (5)              Low-level wrapper for `::pdf_eval_function()`.
13820: (4)             This function has out-params. Python/C# wrappers look like:
13821: (8)                 `ll_pdf_eval_function(::pdf_function *func, const float *in, int
inlen, int outlen)` => float out
13822: (4)             """
13823: (4)             return _mupdf.ll_pdf_eval_function(func, _in, inlen, out, outlen)
13824: (0)         def ll_pdf_event_issue_alert(doc, evt):
13825: (4)             r""" Low-level wrapper for `::pdf_event_issue_alert()`."""
13826: (4)             return _mupdf.ll_pdf_event_issue_alert(doc, evt)
13827: (0)         def ll_pdf_event_issue_exec_menu_item(doc, item):
13828: (4)             r""" Low-level wrapper for `::pdf_event_issue_exec_menu_item()`."""
13829: (4)             return _mupdf.ll_pdf_event_issue_exec_menu_item(doc, item)
13830: (0)         def ll_pdf_event_issue_launch_url(doc, url, new_frame):
13831: (4)             r""" Low-level wrapper for `::pdf_event_issue_launch_url()`."""
13832: (4)             return _mupdf.ll_pdf_event_issue_launch_url(doc, url, new_frame)
13833: (0)         def ll_pdf_event_issue_mail_doc(doc, evt):
13834: (4)             r""" Low-level wrapper for `::pdf_event_issue_mail_doc()`."""
13835: (4)             return _mupdf.ll_pdf_event_issue_mail_doc(doc, evt)
13836: (0)         def ll_pdf_event_issue_print(doc):
13837: (4)             r""" Low-level wrapper for `::pdf_event_issue_print()`."""
13838: (4)             return _mupdf.ll_pdf_event_issue_print(doc)
13839: (0)         def ll_pdf_field_border_style(field):
13840: (4)             r""" Low-level wrapper for `::pdf_field_border_style()`."""
13841: (4)             return _mupdf.ll_pdf_field_border_style(field)
13842: (0)         def ll_pdf_field_display(field):
13843: (4)             r""" Low-level wrapper for `::pdf_field_display()`."""
13844: (4)             return _mupdf.ll_pdf_field_display(field)
13845: (0)         def ll_pdf_field_event_calculate(doc, field):
13846: (4)             r""" Low-level wrapper for `::pdf_field_event_calculate()`."""
13847: (4)             return _mupdf.ll_pdf_field_event_calculate(doc, field)
13848: (0)         def ll_pdf_field_event_format(doc, field):
13849: (4)             r""" Low-level wrapper for `::pdf_field_event_format()`."""
13850: (4)             return _mupdf.ll_pdf_field_event_format(doc, field)
13851: (0)         def ll_pdf_field_event_keystroke(doc, field, evt):
13852: (4)             r""" Low-level wrapper for `::pdf_field_event_keystroke()`."""
13853: (4)             return _mupdf.ll_pdf_field_event_keystroke(doc, field, evt)
13854: (0)         def ll_pdf_field_event_validate(doc, field, value, newvalue):
13855: (4)             r"""
13856: (5)              Low-level wrapper for `::pdf_field_event_validate()`.
13857: (4)             This function has out-params. Python/C# wrappers look like:
13858: (8)                 `ll_pdf_field_event_validate(::pdf_document *doc, ::pdf_obj *field,
const char *value)` => `(int, char *newvalue)`
13859: (4)             """
13860: (4)             return _mupdf.ll_pdf_field_event_validate(doc, field, value, newvalue)
```

```
13861: (0)          def ll_pdf_field_flags(field):
13862: (4)              r""" Low-level wrapper for `::pdf_field_flags()`."""
13863: (4)              return _mupdf.ll_pdf_field_flags(field)
13864: (0)          def ll_pdf_field_label(field):
13865: (4)              r""" Low-level wrapper for `::pdf_field_label()`."""
13866: (4)              return _mupdf.ll_pdf_field_label(field)
13867: (0)          def ll_pdf_field_reset(doc, field):
13868: (4)              r""" Low-level wrapper for `::pdf_field_reset()`."""
13869: (4)              return _mupdf.ll_pdf_field_reset(doc, field)
13870: (0)          def ll_pdf_field_set_border_style(field, text):
13871: (4)              r""" Low-level wrapper for `::pdf_field_set_border_style()`."""
13872: (4)              return _mupdf.ll_pdf_field_set_border_style(field, text)
13873: (0)          def ll_pdf_field_set_button_caption(field, text):
13874: (4)              r""" Low-level wrapper for `::pdf_field_set_button_caption()`."""
13875: (4)              return _mupdf.ll_pdf_field_set_button_caption(field, text)
13876: (0)          def ll_pdf_field_set_display(field, d):
13877: (4)              r""" Low-level wrapper for `::pdf_field_set_display()`."""
13878: (4)              return _mupdf.ll_pdf_field_set_display(field, d)
13879: (0)          def ll_pdf_field_set_fill_color(field, col):
13880: (4)              r""" Low-level wrapper for `::pdf_field_set_fill_color()`."""
13881: (4)              return _mupdf.ll_pdf_field_set_fill_color(field, col)
13882: (0)          def ll_pdf_field_set_text_color(field, col):
13883: (4)              r""" Low-level wrapper for `::pdf_field_set_text_color()`."""
13884: (4)              return _mupdf.ll_pdf_field_set_text_color(field, col)
13885: (0)          def ll_pdf_field_type(field):
13886: (4)              r""" Low-level wrapper for `::pdf_field_type()`."""
13887: (4)              return _mupdf.ll_pdf_field_type(field)
13888: (0)          def ll_pdf_field_type_string(field):
13889: (4)              r""" Low-level wrapper for `::pdf_field_type_string()`."""
13890: (4)              return _mupdf.ll_pdf_field_type_string(field)
13891: (0)          def ll_pdf_field_value(field):
13892: (4)              r""" Low-level wrapper for `::pdf_field_value()`."""
13893: (4)              return _mupdf.ll_pdf_field_value(field)
13894: (0)          def ll_pdf_filter_annot_contents(doc, annot, options):
13895: (4)              r""" Low-level wrapper for `::pdf_filter_annot_contents()`."""
13896: (4)              return _mupdf.ll_pdf_filter_annot_contents(doc, annot, options)
13897: (0)          def ll_pdf_filter_page_contents(doc, page, options):
13898: (4)              r""" Low-level wrapper for `::pdf_filter_page_contents()`."""
13899: (4)              return _mupdf.ll_pdf_filter_page_contents(doc, page, options)
13900: (0)          def ll_pdf_filter_xobject_instance(old_xobj, page_res, ctm, options,
cycle_up):
13901: (4)              r""" Low-level wrapper for `::pdf_filter_xobject_instance()`."""
13902: (4)              return _mupdf.ll_pdf_filter_xobject_instance(old_xobj, page_res, ctm,
options, cycle_up)
13903: (0)          def ll_pdf_find_font_resource(doc, type, encoding, item, key):
13904: (4)              r""" Low-level wrapper for `::pdf_find_font_resource()`."""
13905: (4)              return _mupdf.ll_pdf_find_font_resource(doc, type, encoding, item, key)
13906: (0)          def ll_pdf_find_item(drop, key):
13907: (4)              r""" Low-level wrapper for `::pdf_find_item()`."""
13908: (4)              return _mupdf.ll_pdf_find_item(drop, key)
13909: (0)          def ll_pdf_find_locked_fields(doc, version):
13910: (4)              r""" Low-level wrapper for `::pdf_find_locked_fields()`."""
13911: (4)              return _mupdf.ll_pdf_find_locked_fields(doc, version)
13912: (0)          def ll_pdf_find_locked_fields_for_sig(doc, sig):
13913: (4)              r""" Low-level wrapper for `::pdf_find_locked_fields_for_sig()`."""
13914: (4)              return _mupdf.ll_pdf_find_locked_fields_for_sig(doc, sig)
13915: (0)          def ll_pdf_find_version_for_obj(doc, obj):
13916: (4)              r""" Low-level wrapper for `::pdf_find_version_for_obj()`."""
13917: (4)              return _mupdf.ll_pdf_find_version_for_obj(doc, obj)
13918: (0)          def ll_pdf_first_annot(page):
13919: (4)              r""" Low-level wrapper for `::pdf_first_annot()`."""
13920: (4)              return _mupdf.ll_pdf_first_annot(page)
13921: (0)          def ll_pdf_first_widget(page):
13922: (4)              r""" Low-level wrapper for `::pdf_first_widget()`."""
13923: (4)              return _mupdf.ll_pdf_first_widget(page)
13924: (0)          def ll_pdf_flatten_inheritable_page_items(page):
13925: (4)              r""" Low-level wrapper for `::pdf_flatten_inheritable_page_items()`."""
13926: (4)              return _mupdf.ll_pdf_flatten_inheritable_page_items(page)
13927: (0)          def ll_pdf_font_cid_to_gid(fontdesc, cid):
```

```
13928: (4)             r""" Low-level wrapper for `::pdf_font_cid_to_gid()`."""
13929: (4)             return _mupdf.ll_pdf_font_cid_to_gid(fontdesc, cid)
13930: (0)         def ll_pdf_font_writing_supported(font):
13931: (4)             r""" Low-level wrapper for `::pdf_font_writing_supported()`."""
13932: (4)             return _mupdf.ll_pdf_font_writing_supported(font)
13933: (0)         def ll_pdf_forget_xref(doc):
13934: (4)             r""" Low-level wrapper for `::pdf_forget_xref()`."""
13935: (4)             return _mupdf.ll_pdf_forget_xref(doc)
13936: (0)         def ll_pdf_format_date(time, s, n):
13937: (4)             r""" Low-level wrapper for `::pdf_format_date()`."""
13938: (4)             return _mupdf.ll_pdf_format_date(time, s, n)
13939: (0)         def ll_pdf_format_write_options(buffer, buffer_len, opts):
13940: (4)             r""" Low-level wrapper for `::pdf_format_write_options()`."""
13941: (4)             return _mupdf.ll_pdf_format_write_options(buffer, buffer_len, opts)
13942: (0)         def ll_pdf_function_size(func):
13943: (4)             r""" Low-level wrapper for `::pdf_function_size()`."""
13944: (4)             return _mupdf.ll_pdf_function_size(func)
13945: (0)         def ll_pdf_get_bound_document(obj):
13946: (4)             r""" Low-level wrapper for `::pdf_get_bound_document()`."""
13947: (4)             return _mupdf.ll_pdf_get_bound_document(obj)
13948: (0)         def ll_pdf_get_doc_event_callback_data(doc):
13949: (4)             r""" Low-level wrapper for `::pdf_get_doc_event_callback_data()`."""
13950: (4)             return _mupdf.ll_pdf_get_doc_event_callback_data(doc)
13951: (0)         def ll_pdf_get_embedded_file_params(fs, out):
13952: (4)             r""" Low-level wrapper for `::pdf_get_embedded_file_params()`."""
13953: (4)             return _mupdf.ll_pdf_get_embedded_file_params(fs, out)
13954: (0)         def ll_pdf_get_incremental_xref_entry(doc, i):
13955: (4)             r""" Low-level wrapper for `::pdf_get_incremental_xref_entry()`."""
13956: (4)             return _mupdf.ll_pdf_get_incremental_xref_entry(doc, i)
13957: (0)         def ll_pdf_get_indirect_document(obj):
13958: (4)             r""" Low-level wrapper for `::pdf_get_indirect_document()`."""
13959: (4)             return _mupdf.ll_pdf_get_indirect_document(obj)
13960: (0)         def ll_pdf_get_populating_xref_entry(doc, i):
13961: (4)             r""" Low-level wrapper for `::pdf_get_populating_xref_entry()`."""
13962: (4)             return _mupdf.ll_pdf_get_populating_xref_entry(doc, i)
13963: (0)         def ll_pdf_get_widget_editing_state(widget):
13964: (4)             r""" Low-level wrapper for `::pdf_get_widget_editing_state()`."""
13965: (4)             return _mupdf.ll_pdf_get_widget_editing_state(widget)
13966: (0)         def ll_pdf_get_xref_entry(doc, i):
13967: (4)             r""" Low-level wrapper for `::pdf_get_xref_entry()`."""
13968: (4)             return _mupdf.ll_pdf_get_xref_entry(doc, i)
13969: (0)         def ll_pdf_get_xref_entry_no_change(doc, i):
13970: (4)             r""" Low-level wrapper for `::pdf_get_xref_entry_no_change()`."""
13971: (4)             return _mupdf.ll_pdf_get_xref_entry_no_change(doc, i)
13972: (0)         def ll_pdf_get_xref_entry_no_null(doc, i):
13973: (4)             r""" Low-level wrapper for `::pdf_get_xref_entry_no_null()`."""
13974: (4)             return _mupdf.ll_pdf_get_xref_entry_no_null(doc, i)
13975: (0)         def ll_pdf_graft_mapped_object(map, obj):
13976: (4)             r""" Low-level wrapper for `::pdf_graft_mapped_object()`."""
13977: (4)             return _mupdf.ll_pdf_graft_mapped_object(map, obj)
13978: (0)         def ll_pdf_graft_mapped_page(map, page_to, src, page_from):
13979: (4)             r""" Low-level wrapper for `::pdf_graft_mapped_page()`."""
13980: (4)             return _mupdf.ll_pdf_graft_mapped_page(map, page_to, src, page_from)
13981: (0)         def ll_pdf_graft_object(dst, obj):
13982: (4)             r""" Low-level wrapper for `::pdf_graft_object()`."""
13983: (4)             return _mupdf.ll_pdf_graft_object(dst, obj)
13984: (0)         def ll_pdf_graft_page(dst, page_to, src, page_from):
13985: (4)             r""" Low-level wrapper for `::pdf_graft_page()`."""
13986: (4)             return _mupdf.ll_pdf_graft_page(dst, page_to, src, page_from)
13987: (0)         def ll_pdf_has_permission(doc, p):
13988: (4)             r""" Low-level wrapper for `::pdf_has_permission()`."""
13989: (4)             return _mupdf.ll_pdf_has_permission(doc, p)
13990: (0)         def ll_pdf_has_unsaved_changes(doc):
13991: (4)             r""" Low-level wrapper for `::pdf_has_unsaved_changes()`."""
13992: (4)             return _mupdf.ll_pdf_has_unsaved_changes(doc)
13993: (0)         def ll_pdf_has_unsaved_sigs(doc):
13994: (4)             r""" Low-level wrapper for `::pdf_has_unsaved_sigs()`."""
13995: (4)             return _mupdf.ll_pdf_has_unsaved_sigs(doc)
13996: (0)         def ll_pdf_insert_font_resource(doc, key, obj):
```

```
13997: (4)             r""" Low-level wrapper for `::pdf_insert_font_resource()`."""
13998: (4)             return _mupdf.ll_pdf_insert_font_resource(doc, key, obj)
13999: (0)         def ll_pdf_insert_page(doc, at, page):
14000: (4)             r""" Low-level wrapper for `::pdf_insert_page()`."""
14001: (4)             return _mupdf.ll_pdf_insert_page(doc, at, page)
14002: (0)         def ll_pdf_intent_from_name(obj):
14003: (4)             r""" Low-level wrapper for `::pdf_intent_from_name()`."""
14004: (4)             return _mupdf.ll_pdf_intent_from_name(obj)
14005: (0)         def ll_pdf_intent_from_string(str):
14006: (4)             r""" Low-level wrapper for `::pdf_intent_from_string()`."""
14007: (4)             return _mupdf.ll_pdf_intent_from_string(str)
14008: (0)         def ll_pdf_invalidate_xfa(doc):
14009: (4)             r""" Low-level wrapper for `::pdf_invalidate_xfa()`."""
14010: (4)             return _mupdf.ll_pdf_invalidate_xfa(doc)
14011: (0)         def ll_pdf_is_array(obj):
14012: (4)             r""" Low-level wrapper for `::pdf_is_array()`."""
14013: (4)             return _mupdf.ll_pdf_is_array(obj)
14014: (0)         def ll_pdf_is_bool(obj):
14015: (4)             r""" Low-level wrapper for `::pdf_is_bool()`."""
14016: (4)             return _mupdf.ll_pdf_is_bool(obj)
14017: (0)         def ll_pdf_is_dict(obj):
14018: (4)             r""" Low-level wrapper for `::pdf_is_dict()`."""
14019: (4)             return _mupdf.ll_pdf_is_dict(obj)
14020: (0)         def ll_pdf_is_embedded_file(fs):
14021: (4)             r""" Low-level wrapper for `::pdf_is_embedded_file()`."""
14022: (4)             return _mupdf.ll_pdf_is_embedded_file(fs)
14023: (0)         def ll_pdf_is_field_locked(locked, name):
14024: (4)             r""" Low-level wrapper for `::pdf_is_field_locked()`."""
14025: (4)             return _mupdf.ll_pdf_is_field_locked(locked, name)
14026: (0)         def ll_pdf_is_indirect(obj):
14027: (4)             r""" Low-level wrapper for `::pdf_is_indirect()`."""
14028: (4)             return _mupdf.ll_pdf_is_indirect(obj)
14029: (0)         def ll_pdf_is_int(obj):
14030: (4)             r""" Low-level wrapper for `::pdf_is_int()`."""
14031: (4)             return _mupdf.ll_pdf_is_int(obj)
14032: (0)         def ll_pdf_is_jpx_image(dict):
14033: (4)             r""" Low-level wrapper for `::pdf_is_jpx_image()`."""
14034: (4)             return _mupdf.ll_pdf_is_jpx_image(dict)
14035: (0)         def ll_pdf_is_local_object(doc, obj):
14036: (4)             r""" Low-level wrapper for `::pdf_is_local_object()`."""
14037: (4)             return _mupdf.ll_pdf_is_local_object(doc, obj)
14038: (0)         def ll_pdf_is_name(obj):
14039: (4)             r""" Low-level wrapper for `::pdf_is_name()`."""
14040: (4)             return _mupdf.ll_pdf_is_name(obj)
14041: (0)         def ll_pdf_is_null(obj):
14042: (4)             r""" Low-level wrapper for `::pdf_is_null()`."""
14043: (4)             return _mupdf.ll_pdf_is_null(obj)
14044: (0)         def ll_pdf_is_number(obj):
14045: (4)             r""" Low-level wrapper for `::pdf_is_number()`."""
14046: (4)             return _mupdf.ll_pdf_is_number(obj)
14047: (0)         def ll_pdf_is_ocg_hidden(doc, rdb, usage, ocg):
14048: (4)             r""" Low-level wrapper for `::pdf_is_ocg_hidden()`."""
14049: (4)             return _mupdf.ll_pdf_is_ocg_hidden(doc, rdb, usage, ocg)
14050: (0)         def ll_pdf_is_real(obj):
14051: (4)             r""" Low-level wrapper for `::pdf_is_real()`."""
14052: (4)             return _mupdf.ll_pdf_is_real(obj)
14053: (0)         def ll_pdf_is_stream(obj):
14054: (4)             r""" Low-level wrapper for `::pdf_is_stream()`."""
14055: (4)             return _mupdf.ll_pdf_is_stream(obj)
14056: (0)         def ll_pdf_is_string(obj):
14057: (4)             r""" Low-level wrapper for `::pdf_is_string()`."""
14058: (4)             return _mupdf.ll_pdf_is_string(obj)
14059: (0)         def ll_pdf_is_tint_colorspace(cs):
14060: (4)             r""" Low-level wrapper for `::pdf_is_tint_colorspace()`."""
14061: (4)             return _mupdf.ll_pdf_is_tint_colorspace(cs)
14062: (0)         def ll_pdf_js_event_init(js, target, value, willCommit):
14063: (4)             r""" Low-level wrapper for `::pdf_js_event_init()`."""
14064: (4)             return _mupdf.ll_pdf_js_event_init(js, target, value, willCommit)
14065: (0)         def ll_pdf_js_event_init_keystroke(js, target, evt):
```

```
14066: (4)             r""" Low-level wrapper for `::pdf_js_event_init_keystroke()`."""
14067: (4)             return _mupdf.ll_pdf_js_event_init_keystroke(js, target, evt)
14068: (0)         def ll_pdf_js_event_result(js):
14069: (4)             r""" Low-level wrapper for `::pdf_js_event_result()`."""
14070: (4)             return _mupdf.ll_pdf_js_event_result(js)
14071: (0)         def ll_pdf_js_event_result_keystroke(js, evt):
14072: (4)             r""" Low-level wrapper for `::pdf_js_event_result_keystroke()`."""
14073: (4)             return _mupdf.ll_pdf_js_event_result_keystroke(js, evt)
14074: (0)         def ll_pdf_js_event_result_validate(js, newvalue):
14075: (4)             r"""
14076: (5)              Low-level wrapper for `::pdf_js_event_result_validate()`.
14077: (4)             This function has out-params. Python/C# wrappers look like:
14078: (8)                 `ll_pdf_js_event_result_validate(::pdf_js *js)` => `(int, char
*newvalue)`
14079: (4)             """
14080: (4)             return _mupdf.ll_pdf_js_event_result_validate(js, newvalue)
14081: (0)         def ll_pdf_js_event_value(js):
14082: (4)             r""" Low-level wrapper for `::pdf_js_event_value()`."""
14083: (4)             return _mupdf.ll_pdf_js_event_value(js)
14084: (0)         def ll_pdf_js_execute(js, name, code, result):
14085: (4)             r"""
14086: (5)              Low-level wrapper for `::pdf_js_execute()`.
14087: (4)             This function has out-params. Python/C# wrappers look like:
14088: (8)                 `ll_pdf_js_execute(::pdf_js *js, const char *name, const char *code)`
=> char *result
14089: (4)             """
14090: (4)             return _mupdf.ll_pdf_js_execute(js, name, code, result)
14091: (0)         def ll_pdf_js_get_console(doc):
14092: (4)             r""" Low-level wrapper for `::pdf_js_get_console()`."""
14093: (4)             return _mupdf.ll_pdf_js_get_console(doc)
14094: (0)         def ll_pdf_js_set_console(doc, console, user):
14095: (4)             r""" Low-level wrapper for `::pdf_js_set_console()`."""
14096: (4)             return _mupdf.ll_pdf_js_set_console(doc, console, user)
14097: (0)         def ll_pdf_js_supported(doc):
14098: (4)             r""" Low-level wrapper for `::pdf_js_supported()`."""
14099: (4)             return _mupdf.ll_pdf_js_supported(doc)
14100: (0)         def ll_pdf_keep_annot(annot):
14101: (4)             r""" Low-level wrapper for `::pdf_keep_annot()`."""
14102: (4)             return _mupdf.ll_pdf_keep_annot(annot)
14103: (0)         def ll_pdf_keep_cmap(cmap):
14104: (4)             r""" Low-level wrapper for `::pdf_keep_cmap()`."""
14105: (4)             return _mupdf.ll_pdf_keep_cmap(cmap)
14106: (0)         def ll_pdf_keep_document(doc):
14107: (4)             r""" Low-level wrapper for `::pdf_keep_document()`."""
14108: (4)             return _mupdf.ll_pdf_keep_document(doc)
14109: (0)         def ll_pdf_keep_font(fontdesc):
14110: (4)             r""" Low-level wrapper for `::pdf_keep_font()`."""
14111: (4)             return _mupdf.ll_pdf_keep_font(fontdesc)
14112: (0)         def ll_pdf_keep_function(func):
14113: (4)             r""" Low-level wrapper for `::pdf_keep_function()`."""
14114: (4)             return _mupdf.ll_pdf_keep_function(func)
14115: (0)         def ll_pdf_keep_graft_map(map):
14116: (4)             r""" Low-level wrapper for `::pdf_keep_graft_map()`."""
14117: (4)             return _mupdf.ll_pdf_keep_graft_map(map)
14118: (0)         def ll_pdf_keep_obj(obj):
14119: (4)             r""" Low-level wrapper for `::pdf_keep_obj()`."""
14120: (4)             return _mupdf.ll_pdf_keep_obj(obj)
14121: (0)         def ll_pdf_keep_page(page):
14122: (4)             r""" Low-level wrapper for `::pdf_keep_page()`."""
14123: (4)             return _mupdf.ll_pdf_keep_page(page)
14124: (0)         def ll_pdf_keep_pattern(pat):
14125: (4)             r""" Low-level wrapper for `::pdf_keep_pattern()`."""
14126: (4)             return _mupdf.ll_pdf_keep_pattern(pat)
14127: (0)         def ll_pdf_keep_processor(proc):
14128: (4)             r""" Low-level wrapper for `::pdf_keep_processor()`."""
14129: (4)             return _mupdf.ll_pdf_keep_processor(proc)
14130: (0)         def ll_pdf_keep_widget(widget):
14131: (4)             r""" Low-level wrapper for `::pdf_keep_widget()`."""
14132: (4)             return _mupdf.ll_pdf_keep_widget(widget)
```

```
14133: (0)          def ll_pdf_layer_config_info(doc, config_num, info):
14134: (4)              r""" Low-level wrapper for `::pdf_layer_config_info()`."""
14135: (4)              return _mupdf.ll_pdf_layer_config_info(doc, config_num, info)
14136: (0)          def ll_pdf_layer_config_ui_info(doc, ui, info):
14137: (4)              r""" Low-level wrapper for `::pdf_layer_config_ui_info()`."""
14138: (4)              return _mupdf.ll_pdf_layer_config_ui_info(doc, ui, info)
14139: (0)          def ll_pdf_layer_is_enabled(doc, layer):
14140: (4)              r""" Low-level wrapper for `::pdf_layer_is_enabled()`."""
14141: (4)              return _mupdf.ll_pdf_layer_is_enabled(doc, layer)
14142: (0)          def ll_pdf_layer_name(doc, layer):
14143: (4)              r""" Low-level wrapper for `::pdf_layer_name()`."""
14144: (4)              return _mupdf.ll_pdf_layer_name(doc, layer)
14145: (0)          def ll_pdf_layout_fit_text(font, lang, str, bounds):
14146: (4)              r""" Low-level wrapper for `::pdf_layout_fit_text()`."""
14147: (4)              return _mupdf.ll_pdf_layout_fit_text(font, lang, str, bounds)
14148: (0)          def ll_pdf_layout_text_widget(annot):
14149: (4)              r""" Low-level wrapper for `::pdf_layout_text_widget()`."""
14150: (4)              return _mupdf.ll_pdf_layout_text_widget(annot)
14151: (0)          def ll_pdf_lex(f, lexbuf):
14152: (4)              r""" Low-level wrapper for `::pdf_lex()`."""
14153: (4)              return _mupdf.ll_pdf_lex(f, lexbuf)
14154: (0)          def ll_pdf_lex_no_string(f, lexbuf):
14155: (4)              r""" Low-level wrapper for `::pdf_lex_no_string()`."""
14156: (4)              return _mupdf.ll_pdf_lex_no_string(f, lexbuf)
14157: (0)          def ll_pdf_lexbuf_fin(lexbuf):
14158: (4)              r""" Low-level wrapper for `::pdf_lexbuf_fin()`."""
14159: (4)              return _mupdf.ll_pdf_lexbuf_fin(lexbuf)
14160: (0)          def ll_pdf_lexbuf_grow(lexbuf):
14161: (4)              r""" Low-level wrapper for `::pdf_lexbuf_grow()`."""
14162: (4)              return _mupdf.ll_pdf_lexbuf_grow(lexbuf)
14163: (0)          def ll_pdf_lexbuf_init(lexbuf, size):
14164: (4)              r""" Low-level wrapper for `::pdf_lexbuf_init()`."""
14165: (4)              return _mupdf.ll_pdf_lexbuf_init(lexbuf, size)
14166: (0)          def ll_pdf_line_ending_from_name(end):
14167: (4)              r""" Low-level wrapper for `::pdf_line_ending_from_name()`."""
14168: (4)              return _mupdf.ll_pdf_line_ending_from_name(end)
14169: (0)          def ll_pdf_line_ending_from_string(end):
14170: (4)              r""" Low-level wrapper for `::pdf_line_ending_from_string()`."""
14171: (4)              return _mupdf.ll_pdf_line_ending_from_string(end)
14172: (0)          def ll_pdf_load_builtin_cmap(name):
14173: (4)              r""" Low-level wrapper for `::pdf_load_builtin_cmap()`."""
14174: (4)              return _mupdf.ll_pdf_load_builtin_cmap(name)
14175: (0)          def ll_pdf_load_cmap(file):
14176: (4)              r""" Low-level wrapper for `::pdf_load_cmap()`."""
14177: (4)              return _mupdf.ll_pdf_load_cmap(file)
14178: (0)          def ll_pdf_load_colorspace(obj):
14179: (4)              r""" Low-level wrapper for `::pdf_load_colorspace()`."""
14180: (4)              return _mupdf.ll_pdf_load_colorspace(obj)
14181: (0)          def ll_pdf_load_compressed_inline_image(doc, dict, length, cstm, indexed,
image):
14182: (4)              r""" Low-level wrapper for `::pdf_load_compressed_inline_image()`."""
14183: (4)              return _mupdf.ll_pdf_load_compressed_inline_image(doc, dict, length, cstm,
indexed, image)
14184: (0)          def ll_pdf_load_compressed_stream(doc, num, worst_case):
14185: (4)              r""" Low-level wrapper for `::pdf_load_compressed_stream()`."""
14186: (4)              return _mupdf.ll_pdf_load_compressed_stream(doc, num, worst_case)
14187: (0)          def ll_pdf_load_default_colorspaces(doc, page):
14188: (4)              r""" Low-level wrapper for `::pdf_load_default_colorspaces()`."""
14189: (4)              return _mupdf.ll_pdf_load_default_colorspaces(doc, page)
14190: (0)          def ll_pdf_load_embedded_cmap(doc, ref):
14191: (4)              r""" Low-level wrapper for `::pdf_load_embedded_cmap()`."""
14192: (4)              return _mupdf.ll_pdf_load_embedded_cmap(doc, ref)
14193: (0)          def ll_pdf_load_embedded_file_contents(fs):
14194: (4)              r""" Low-level wrapper for `::pdf_load_embedded_file_contents()`."""
14195: (4)              return _mupdf.ll_pdf_load_embedded_file_contents(fs)
14196: (0)          def ll_pdf_load_encoding(estrings, encoding):
14197: (4)              r"""
14198: (5)               Low-level wrapper for `::pdf_load_encoding()`.
14199: (4)              This function has out-params. Python/C# wrappers look like:
```

```
14200: (8)                `ll_pdf_load_encoding(const char *encoding)` => const char *estrings
14201: (4)            """
14202: (4)            return _mupdf.ll_pdf_load_encoding(estrings, encoding)
14203: (0)        def ll_pdf_load_field_name(field):
14204: (4)            r""" Low-level wrapper for `::pdf_load_field_name()`."""
14205: (4)            return _mupdf.ll_pdf_load_field_name(field)
14206: (0)        def ll_pdf_load_field_name2(field):
14207: (4)            r""" Alternative to `ll_pdf_load_field_name()` that returns a
std::string."""
14208: (4)            return _mupdf.ll_pdf_load_field_name2(field)
14209: (0)        def ll_pdf_load_font(doc, rdb, obj):
14210: (4)            r""" Low-level wrapper for `::pdf_load_font()`."""
14211: (4)            return _mupdf.ll_pdf_load_font(doc, rdb, obj)
14212: (0)        def ll_pdf_load_function(ref, _in, out):
14213: (4)            r""" Low-level wrapper for `::pdf_load_function()`."""
14214: (4)            return _mupdf.ll_pdf_load_function(ref, _in, out)
14215: (0)        def ll_pdf_load_hail_mary_font(doc):
14216: (4)            r""" Low-level wrapper for `::pdf_load_hail_mary_font()`."""
14217: (4)            return _mupdf.ll_pdf_load_hail_mary_font(doc)
14218: (0)        def ll_pdf_load_image(doc, obj):
14219: (4)            r""" Low-level wrapper for `::pdf_load_image()`."""
14220: (4)            return _mupdf.ll_pdf_load_image(doc, obj)
14221: (0)        def ll_pdf_load_inline_image(doc, rdb, dict, file):
14222: (4)            r""" Low-level wrapper for `::pdf_load_inline_image()`."""
14223: (4)            return _mupdf.ll_pdf_load_inline_image(doc, rdb, dict, file)
14224: (0)        def ll_pdf_load_journal(doc, filename):
14225: (4)            r""" Low-level wrapper for `::pdf_load_journal()`."""
14226: (4)            return _mupdf.ll_pdf_load_journal(doc, filename)
14227: (0)        def ll_pdf_load_link_annots(arg_1, arg_2, annots, pagenum, page_ctm):
14228: (4)            r""" Low-level wrapper for `::pdf_load_link_annots()`."""
14229: (4)            return _mupdf.ll_pdf_load_link_annots(arg_1, arg_2, annots, pagenum,
page_ctm)
14230: (0)        def ll_pdf_load_links(page):
14231: (4)            r""" Low-level wrapper for `::pdf_load_links()`."""
14232: (4)            return _mupdf.ll_pdf_load_links(page)
14233: (0)        def ll_pdf_load_name_tree(doc, which):
14234: (4)            r""" Low-level wrapper for `::pdf_load_name_tree()`."""
14235: (4)            return _mupdf.ll_pdf_load_name_tree(doc, which)
14236: (0)        def ll_pdf_load_object(doc, num):
14237: (4)            r"""
14238: (4)            Low-level wrapper for `::pdf_load_object()`.
14239: (4)            Load a given object.
14240: (4)            This can cause xref reorganisations (solidifications etc) due to
14241: (4)            repairs, so all held pdf_xref_entries should be considered
14242: (4)            invalid after this call (other than the returned one).
14243: (4)            """
14244: (4)            return _mupdf.ll_pdf_load_object(doc, num)
14245: (0)        def ll_pdf_load_outline(doc):
14246: (4)            r""" Low-level wrapper for `::pdf_load_outline()`."""
14247: (4)            return _mupdf.ll_pdf_load_outline(doc)
14248: (0)        def ll_pdf_load_page(doc, number):
14249: (4)            r""" Low-level wrapper for `::pdf_load_page()`."""
14250: (4)            return _mupdf.ll_pdf_load_page(doc, number)
14251: (0)        def ll_pdf_load_page_imp(doc, chapter, number):
14252: (4)            r""" Low-level wrapper for `::pdf_load_page_imp()`."""
14253: (4)            return _mupdf.ll_pdf_load_page_imp(doc, chapter, number)
14254: (0)        def ll_pdf_load_page_tree(doc):
14255: (4)            r""" Low-level wrapper for `::pdf_load_page_tree()`."""
14256: (4)            return _mupdf.ll_pdf_load_page_tree(doc)
14257: (0)        def ll_pdf_load_pattern(doc, obj):
14258: (4)            r""" Low-level wrapper for `::pdf_load_pattern()`."""
14259: (4)            return _mupdf.ll_pdf_load_pattern(doc, obj)
14260: (0)        def ll_pdf_load_raw_stream(ref):
14261: (4)            r""" Low-level wrapper for `::pdf_load_raw_stream()`."""
14262: (4)            return _mupdf.ll_pdf_load_raw_stream(ref)
14263: (0)        def ll_pdf_load_raw_stream_number(doc, num):
14264: (4)            r""" Low-level wrapper for `::pdf_load_raw_stream_number()`."""
14265: (4)            return _mupdf.ll_pdf_load_raw_stream_number(doc, num)
14266: (0)        def ll_pdf_load_shading(doc, obj):
```

```
14267: (4)             r""" Low-level wrapper for `::pdf_load_shading()`."""
14268: (4)             return _mupdf.ll_pdf_load_shading(doc, obj)
14269: (0)         def ll_pdf_load_stream(ref):
14270: (4)             r""" Low-level wrapper for `::pdf_load_stream()`."""
14271: (4)             return _mupdf.ll_pdf_load_stream(ref)
14272: (0)         def ll_pdf_load_stream_number(doc, num):
14273: (4)             r""" Low-level wrapper for `::pdf_load_stream_number()`."""
14274: (4)             return _mupdf.ll_pdf_load_stream_number(doc, num)
14275: (0)         def ll_pdf_load_stream_or_string_as_utf8(src):
14276: (4)             r""" Low-level wrapper for `::pdf_load_stream_or_string_as_utf8()`."""
14277: (4)             return _mupdf.ll_pdf_load_stream_or_string_as_utf8(src)
14278: (0)         def ll_pdf_load_system_cmap(name):
14279: (4)             r""" Low-level wrapper for `::pdf_load_system_cmap()`."""
14280: (4)             return _mupdf.ll_pdf_load_system_cmap(name)
14281: (0)         def ll_pdf_load_to_unicode(doc, font, strings, collection, cmapstm):
14282: (4)             r"""
14283: (5)              Low-level wrapper for `::pdf_load_to_unicode()`.
14284: (4)             This function has out-params. Python/C# wrappers look like:
14285: (8)                 `ll_pdf_load_to_unicode(::pdf_document *doc, ::pdf_font_desc *font,
char *collection, ::pdf_obj *cmapstm)` => const char *strings
14286: (4)             """
14287: (4)             return _mupdf.ll_pdf_load_to_unicode(doc, font, strings, collection,
cmapstm)
14288: (0)         def ll_pdf_load_type3_font(doc, rdb, obj):
14289: (4)             r""" Low-level wrapper for `::pdf_load_type3_font()`."""
14290: (4)             return _mupdf.ll_pdf_load_type3_font(doc, rdb, obj)
14291: (0)         def ll_pdf_load_type3_glyphs(doc, fontdesc):
14292: (4)             r""" Low-level wrapper for `::pdf_load_type3_glyphs()`."""
14293: (4)             return _mupdf.ll_pdf_load_type3_glyphs(doc, fontdesc)
14294: (0)         def ll_pdf_load_unencrypted_object(doc, num):
14295: (4)             r""" Low-level wrapper for `::pdf_load_unencrypted_object()`."""
14296: (4)             return _mupdf.ll_pdf_load_unencrypted_object(doc, num)
14297: (0)         def ll_pdf_lookup_cmap(cmap, cpt):
14298: (4)             r""" Low-level wrapper for `::pdf_lookup_cmap()`."""
14299: (4)             return _mupdf.ll_pdf_lookup_cmap(cmap, cpt)
14300: (0)         def ll_pdf_lookup_cmap_full(cmap, cpt, out):
14301: (4)             r"""
14302: (5)              Low-level wrapper for `::pdf_lookup_cmap_full()`.
14303: (4)             This function has out-params. Python/C# wrappers look like:
14304: (8)                 `ll_pdf_lookup_cmap_full(::pdf_cmap *cmap, unsigned int cpt)` =>
`(int, int out)`
14305: (4)             """
14306: (4)             return _mupdf.ll_pdf_lookup_cmap_full(cmap, cpt, out)
14307: (0)         def ll_pdf_lookup_dest(doc, needle):
14308: (4)             r""" Low-level wrapper for `::pdf_lookup_dest()`."""
14309: (4)             return _mupdf.ll_pdf_lookup_dest(doc, needle)
14310: (0)         def ll_pdf_lookup_field(form, name):
14311: (4)             r""" Low-level wrapper for `::pdf_lookup_field()`."""
14312: (4)             return _mupdf.ll_pdf_lookup_field(form, name)
14313: (0)         def ll_pdf_lookup_hmtx(font, cid):
14314: (4)             r""" Low-level wrapper for `::pdf_lookup_hmtx()`."""
14315: (4)             return _mupdf.ll_pdf_lookup_hmtx(font, cid)
14316: (0)         def ll_pdf_lookup_metadata(doc, key, ptr, size):
14317: (4)             r""" Low-level wrapper for `::pdf_lookup_metadata()`."""
14318: (4)             return _mupdf.ll_pdf_lookup_metadata(doc, key, ptr, size)
14319: (0)         def ll_pdf_lookup_metadata2(doc, key):
14320: (4)             r"""
14321: (5)              Low-level wrapper for `::pdf_lookup_metadata2()`.
14322: (4)             C++ alternative to `pdf_lookup_metadata()` that returns a `std::string`
14323: (4)             or calls `fz_throw()` if not found.
14324: (4)             """
14325: (4)             return _mupdf.ll_pdf_lookup_metadata2(doc, key)
14326: (0)         def ll_pdf_lookup_name(doc, which, needle):
14327: (4)             r""" Low-level wrapper for `::pdf_lookup_name()`."""
14328: (4)             return _mupdf.ll_pdf_lookup_name(doc, which, needle)
14329: (0)         def ll_pdf_lookup_number(root, needle):
14330: (4)             r""" Low-level wrapper for `::pdf_lookup_number()`."""
14331: (4)             return _mupdf.ll_pdf_lookup_number(root, needle)
14332: (0)         def ll_pdf_lookup_page_loc(doc, needle, parentp, indexp):
```

```
14333: (4)             r"""
14334: (5)              Low-level wrapper for `::pdf_lookup_page_loc()`.
14335: (4)             This function has out-params. Python/C# wrappers look like:
14336: (8)                 `ll_pdf_lookup_page_loc(::pdf_document *doc, int needle, ::pdf_obj
**parentp)` => `(pdf_obj *, int indexp)`
14337: (4)             """
14338: (4)             return _mupdf.ll_pdf_lookup_page_loc(doc, needle, parentp, indexp)
14339: (0)         def ll_pdf_lookup_page_number(doc, pageobj):
14340: (4)             r""" Low-level wrapper for `::pdf_lookup_page_number()`."""
14341: (4)             return _mupdf.ll_pdf_lookup_page_number(doc, pageobj)
14342: (0)         def ll_pdf_lookup_page_obj(doc, needle):
14343: (4)             r""" Low-level wrapper for `::pdf_lookup_page_obj()`."""
14344: (4)             return _mupdf.ll_pdf_lookup_page_obj(doc, needle)
14345: (0)         def ll_pdf_lookup_substitute_font(mono, serif, bold, italic, len):
14346: (4)             r"""
14347: (5)              Low-level wrapper for `::pdf_lookup_substitute_font()`.
14348: (4)             This function has out-params. Python/C# wrappers look like:
14349: (8)                 `ll_pdf_lookup_substitute_font(int mono, int serif, int bold, int
italic)` => `(const unsigned char *, int len)`
14350: (4)             """
14351: (4)             return _mupdf.ll_pdf_lookup_substitute_font(mono, serif, bold, italic,
len)
14352: (0)         def ll_pdf_lookup_vmtx(font, cid):
14353: (4)             r""" Low-level wrapper for `::pdf_lookup_vmtx()`."""
14354: (4)             return _mupdf.ll_pdf_lookup_vmtx(font, cid)
14355: (0)         def ll_pdf_map_one_to_many(cmap, one, many, len):
14356: (4)             r"""
14357: (5)              Low-level wrapper for `::pdf_map_one_to_many()`.
14358: (4)             This function has out-params. Python/C# wrappers look like:
14359: (8)                 `ll_pdf_map_one_to_many(::pdf_cmap *cmap, unsigned int one, size_t
len)` => int many
14360: (4)             """
14361: (4)             return _mupdf.ll_pdf_map_one_to_many(cmap, one, many, len)
14362: (0)         def ll_pdf_map_range_to_range(cmap, srclo, srchi, dstlo):
14363: (4)             r""" Low-level wrapper for `::pdf_map_range_to_range()`."""
14364: (4)             return _mupdf.ll_pdf_map_range_to_range(cmap, srclo, srchi, dstlo)
14365: (0)         def ll_pdf_mark_bits_reset(marks):
14366: (4)             r""" Low-level wrapper for `::pdf_mark_bits_reset()`."""
14367: (4)             return _mupdf.ll_pdf_mark_bits_reset(marks)
14368: (0)         def ll_pdf_mark_bits_set(marks, obj):
14369: (4)             r""" Low-level wrapper for `::pdf_mark_bits_set()`."""
14370: (4)             return _mupdf.ll_pdf_mark_bits_set(marks, obj)
14371: (0)         def ll_pdf_mark_list_check(list, obj):
14372: (4)             r""" Low-level wrapper for `::pdf_mark_list_check()`."""
14373: (4)             return _mupdf.ll_pdf_mark_list_check(list, obj)
14374: (0)         def ll_pdf_mark_list_free(list):
14375: (4)             r""" Low-level wrapper for `::pdf_mark_list_free()`."""
14376: (4)             return _mupdf.ll_pdf_mark_list_free(list)
14377: (0)         def ll_pdf_mark_list_init(list):
14378: (4)             r""" Low-level wrapper for `::pdf_mark_list_init()`."""
14379: (4)             return _mupdf.ll_pdf_mark_list_init(list)
14380: (0)         def ll_pdf_mark_list_pop(list):
14381: (4)             r""" Low-level wrapper for `::pdf_mark_list_pop()`."""
14382: (4)             return _mupdf.ll_pdf_mark_list_pop(list)
14383: (0)         def ll_pdf_mark_list_push(list, obj):
14384: (4)             r""" Low-level wrapper for `::pdf_mark_list_push()`."""
14385: (4)             return _mupdf.ll_pdf_mark_list_push(list, obj)
14386: (0)         def ll_pdf_mark_obj(obj):
14387: (4)             r""" Low-level wrapper for `::pdf_mark_obj()`."""
14388: (4)             return _mupdf.ll_pdf_mark_obj(obj)
14389: (0)         def ll_pdf_mark_xref(doc):
14390: (4)             r""" Low-level wrapper for `::pdf_mark_xref()`."""
14391: (4)             return _mupdf.ll_pdf_mark_xref(doc)
14392: (0)         def ll_pdf_metadata(doc):
14393: (4)             r""" Low-level wrapper for `::pdf_metadata()`."""
14394: (4)             return _mupdf.ll_pdf_metadata(doc)
14395: (0)         def ll_pdf_minimize_document(doc):
14396: (4)             r""" Low-level wrapper for `::pdf_minimize_document()`."""
14397: (4)             return _mupdf.ll_pdf_minimize_document(doc)
```

```
14398: (0)          def ll_pdf_name_eq(a, b):
14399: (4)              r""" Low-level wrapper for `::pdf_name_eq()`."""
14400: (4)              return _mupdf.ll_pdf_name_eq(a, b)
14401: (0)          def ll_pdf_name_from_intent(intent):
14402: (4)              r""" Low-level wrapper for `::pdf_name_from_intent()`."""
14403: (4)              return _mupdf.ll_pdf_name_from_intent(intent)
14404: (0)          def ll_pdf_name_from_line_ending(end):
14405: (4)              r""" Low-level wrapper for `::pdf_name_from_line_ending()`."""
14406: (4)              return _mupdf.ll_pdf_name_from_line_ending(end)
14407: (0)          def ll_pdf_needs_password(doc):
14408: (4)              r""" Low-level wrapper for `::pdf_needs_password()`."""
14409: (4)              return _mupdf.ll_pdf_needs_password(doc)
14410: (0)          def ll_pdf_new_action_from_link(doc, uri):
14411: (4)              r""" Low-level wrapper for `::pdf_new_action_from_link()`."""
14412: (4)              return _mupdf.ll_pdf_new_action_from_link(doc, uri)
14413: (0)          def ll_pdf_new_array(doc, initialcap):
14414: (4)              r""" Low-level wrapper for `::pdf_new_array()`."""
14415: (4)              return _mupdf.ll_pdf_new_array(doc, initialcap)
14416: (0)          def ll_pdf_new_buffer_processor(buffer, ahxencode, newlines):
14417: (4)              r""" Low-level wrapper for `::pdf_new_buffer_processor()`."""
14418: (4)              return _mupdf.ll_pdf_new_buffer_processor(buffer, ahxencode, newlines)
14419: (0)          def ll_pdf_new_cmap():
14420: (4)              r""" Low-level wrapper for `::pdf_new_cmap()`."""
14421: (4)              return _mupdf.ll_pdf_new_cmap()
14422: (0)          def ll_pdf_new_color_filter(doc, chain, struct_parents, transform, options,
copts):
14423: (4)              r""" Low-level wrapper for `::pdf_new_color_filter()`."""
14424: (4)              return _mupdf.ll_pdf_new_color_filter(doc, chain, struct_parents,
transform, options, copts)
14425: (0)          def ll_pdf_new_crypt(enc, id):
14426: (4)              r""" Low-level wrapper for `::pdf_new_crypt()`."""
14427: (4)              return _mupdf.ll_pdf_new_crypt(enc, id)
14428: (0)          def ll_pdf_new_date(doc, time):
14429: (4)              r""" Low-level wrapper for `::pdf_new_date()`."""
14430: (4)              return _mupdf.ll_pdf_new_date(doc, time)
14431: (0)          def ll_pdf_new_dest_from_link(doc, uri, is_remote):
14432: (4)              r""" Low-level wrapper for `::pdf_new_dest_from_link()`."""
14433: (4)              return _mupdf.ll_pdf_new_dest_from_link(doc, uri, is_remote)
14434: (0)          def ll_pdf_new_dict(doc, initialcap):
14435: (4)              r""" Low-level wrapper for `::pdf_new_dict()`."""
14436: (4)              return _mupdf.ll_pdf_new_dict(doc, initialcap)
14437: (0)          def ll_pdf_new_display_list_from_annot(annot):
14438: (4)              r""" Low-level wrapper for `::pdf_new_display_list_from_annot()`."""
14439: (4)              return _mupdf.ll_pdf_new_display_list_from_annot(annot)
14440: (0)          def ll_pdf_new_encrypt(opwd_utf8, upwd_utf8, id, permissions, algorithm):
14441: (4)              r""" Low-level wrapper for `::pdf_new_encrypt()`."""
14442: (4)              return _mupdf.ll_pdf_new_encrypt(opwd_utf8, upwd_utf8, id, permissions,
algorithm)
14443: (0)          def ll_pdf_new_font_desc():
14444: (4)              r""" Low-level wrapper for `::pdf_new_font_desc()`."""
14445: (4)              return _mupdf.ll_pdf_new_font_desc()
14446: (0)          def ll_pdf_new_graft_map(dst):
14447: (4)              r""" Low-level wrapper for `::pdf_new_graft_map()`."""
14448: (4)              return _mupdf.ll_pdf_new_graft_map(dst)
14449: (0)          def ll_pdf_new_identity_cmap(wmode, bytes):
14450: (4)              r""" Low-level wrapper for `::pdf_new_identity_cmap()`."""
14451: (4)              return _mupdf.ll_pdf_new_identity_cmap(wmode, bytes)
14452: (0)          def ll_pdf_new_indirect(doc, num, gen):
14453: (4)              r""" Low-level wrapper for `::pdf_new_indirect()`."""
14454: (4)              return _mupdf.ll_pdf_new_indirect(doc, num, gen)
14455: (0)          def ll_pdf_new_int(i):
14456: (4)              r""" Low-level wrapper for `::pdf_new_int()`."""
14457: (4)              return _mupdf.ll_pdf_new_int(i)
14458: (0)          def ll_pdf_new_link(page, rect, uri, obj):
14459: (4)              r""" Low-level wrapper for `::pdf_new_link()`."""
14460: (4)              return _mupdf.ll_pdf_new_link(page, rect, uri, obj)
14461: (0)          def ll_pdf_new_local_xref(doc):
14462: (4)              r""" Low-level wrapper for `::pdf_new_local_xref()`."""
14463: (4)              return _mupdf.ll_pdf_new_local_xref(doc)
```

```
14464: (0)          def ll_pdf_new_mark_bits(doc):
14465: (4)              r""" Low-level wrapper for `::pdf_new_mark_bits()`."""
14466: (4)              return _mupdf.ll_pdf_new_mark_bits(doc)
14467: (0)          def ll_pdf_new_matrix(doc, mtx):
14468: (4)              r""" Low-level wrapper for `::pdf_new_matrix()`."""
14469: (4)              return _mupdf.ll_pdf_new_matrix(doc, mtx)
14470: (0)          def ll_pdf_new_name(str):
14471: (4)              r""" Low-level wrapper for `::pdf_new_name()`."""
14472: (4)              return _mupdf.ll_pdf_new_name(str)
14473: (0)          def ll_pdf_new_outline_iterator(doc):
14474: (4)              r""" Low-level wrapper for `::pdf_new_outline_iterator()`."""
14475: (4)              return _mupdf.ll_pdf_new_outline_iterator(doc)
14476: (0)          def ll_pdf_new_output_processor(out, ahxencode, newlines):
14477: (4)              r""" Low-level wrapper for `::pdf_new_output_processor()`."""
14478: (4)              return _mupdf.ll_pdf_new_output_processor(out, ahxencode, newlines)
14479: (0)          def ll_pdf_new_pdf_device(doc, topctm, resources, contents):
14480: (4)              r""" Low-level wrapper for `::pdf_new_pdf_device()`."""
14481: (4)              return _mupdf.ll_pdf_new_pdf_device(doc, topctm, resources, contents)
14482: (0)          def ll_pdf_new_pixmap_from_annot(annot, ctm, cs, seps, alpha):
14483: (4)              r""" Low-level wrapper for `::pdf_new_pixmap_from_annot()`."""
14484: (4)              return _mupdf.ll_pdf_new_pixmap_from_annot(annot, ctm, cs, seps, alpha)
14485: (0)          def ll_pdf_new_pixmap_from_page_contents_with_separations_and_usage(page, ctm,
cs, seps, alpha, usage, box):
14486: (4)              r""" Low-level wrapper for
`::pdf_new_pixmap_from_page_contents_with_separations_and_usage()`."""
14487: (4)              return
_mupdf.ll_pdf_new_pixmap_from_page_contents_with_separations_and_usage(page, ctm, cs, seps, alpha,
usage, box)
14488: (0)          def ll_pdf_new_pixmap_from_page_contents_with_usage(page, ctm, cs, alpha,
usage, box):
14489: (4)              r""" Low-level wrapper for
`::pdf_new_pixmap_from_page_contents_with_usage()`."""
14490: (4)              return _mupdf.ll_pdf_new_pixmap_from_page_contents_with_usage(page, ctm,
cs, alpha, usage, box)
14491: (0)          def ll_pdf_new_pixmap_from_page_with_separations_and_usage(page, ctm, cs,
seps, alpha, usage, box):
14492: (4)              r""" Low-level wrapper for
`::pdf_new_pixmap_from_page_with_separations_and_usage()`."""
14493: (4)              return _mupdf.ll_pdf_new_pixmap_from_page_with_separations_and_usage(page,
ctm, cs, seps, alpha, usage, box)
14494: (0)          def ll_pdf_new_pixmap_from_page_with_usage(page, ctm, cs, alpha, usage, box):
14495: (4)              r""" Low-level wrapper for `::pdf_new_pixmap_from_page_with_usage()`."""
14496: (4)              return _mupdf.ll_pdf_new_pixmap_from_page_with_usage(page, ctm, cs, alpha,
usage, box)
14497: (0)          def ll_pdf_new_processor(size):
14498: (4)              r""" Low-level wrapper for `::pdf_new_processor()`."""
14499: (4)              return _mupdf.ll_pdf_new_processor(size)
14500: (0)          def ll_pdf_new_real(f):
14501: (4)              r""" Low-level wrapper for `::pdf_new_real()`."""
14502: (4)              return _mupdf.ll_pdf_new_real(f)
14503: (0)          def ll_pdf_new_rect(doc, rect):
14504: (4)              r""" Low-level wrapper for `::pdf_new_rect()`."""
14505: (4)              return _mupdf.ll_pdf_new_rect(doc, rect)
14506: (0)          def ll_pdf_new_run_processor(doc, dev, ctm, struct_parent, usage, gstate,
default_cs, cookie):
14507: (4)              r""" Low-level wrapper for `::pdf_new_run_processor()`."""
14508: (4)              return _mupdf.ll_pdf_new_run_processor(doc, dev, ctm, struct_parent,
usage, gstate, default_cs, cookie)
14509: (0)          def ll_pdf_new_sanitize_filter(doc, chain, struct_parents, transform, options,
sopts):
14510: (4)              r""" Low-level wrapper for `::pdf_new_sanitize_filter()`."""
14511: (4)              return _mupdf.ll_pdf_new_sanitize_filter(doc, chain, struct_parents,
transform, options, sopts)
14512: (0)          def ll_pdf_new_stext_page_from_annot(annot, options):
14513: (4)              r""" Low-level wrapper for `::pdf_new_stext_page_from_annot()`."""
14514: (4)              return _mupdf.ll_pdf_new_stext_page_from_annot(annot, options)
14515: (0)          def ll_pdf_new_string(str, len):
14516: (4)              r""" Low-level wrapper for `::pdf_new_string()`."""
14517: (4)              return _mupdf.ll_pdf_new_string(str, len)
```

```
14518: (0)          def ll_pdf_new_text_string(s):
14519: (4)              r""" Low-level wrapper for `::pdf_new_text_string()`."""
14520: (4)              return _mupdf.ll_pdf_new_text_string(s)
14521: (0)          def ll_pdf_new_uri_from_explicit_dest(dest):
14522: (4)              r""" Low-level wrapper for `::pdf_new_uri_from_explicit_dest()`."""
14523: (4)              return _mupdf.ll_pdf_new_uri_from_explicit_dest(dest)
14524: (0)          def ll_pdf_new_uri_from_path_and_explicit_dest(path, dest):
14525: (4)              r""" Low-level wrapper for
`::pdf_new_uri_from_path_and_explicit_dest()`."""
14526: (4)              return _mupdf.ll_pdf_new_uri_from_path_and_explicit_dest(path, dest)
14527: (0)          def ll_pdf_new_uri_from_path_and_named_dest(path, name):
14528: (4)              r""" Low-level wrapper for `::pdf_new_uri_from_path_and_named_dest()`."""
14529: (4)              return _mupdf.ll_pdf_new_uri_from_path_and_named_dest(path, name)
14530: (0)          def ll_pdf_new_utf8_from_pdf_stream_obj(src):
14531: (4)              r""" Low-level wrapper for `::pdf_new_utf8_from_pdf_stream_obj()`."""
14532: (4)              return _mupdf.ll_pdf_new_utf8_from_pdf_stream_obj(src)
14533: (0)          def ll_pdf_new_utf8_from_pdf_string(srcptr, srclen):
14534: (4)              r""" Low-level wrapper for `::pdf_new_utf8_from_pdf_string()`."""
14535: (4)              return _mupdf.ll_pdf_new_utf8_from_pdf_string(srcptr, srclen)
14536: (0)          def ll_pdf_new_utf8_from_pdf_string_obj(src):
14537: (4)              r""" Low-level wrapper for `::pdf_new_utf8_from_pdf_string_obj()`."""
14538: (4)              return _mupdf.ll_pdf_new_utf8_from_pdf_string_obj(src)
14539: (0)          def ll_pdf_new_xobject(doc, bbox, matrix, res, buffer):
14540: (4)              r""" Low-level wrapper for `::pdf_new_xobject()`."""
14541: (4)              return _mupdf.ll_pdf_new_xobject(doc, bbox, matrix, res, buffer)
14542: (0)          def ll_pdf_next_annot(annot):
14543: (4)              r""" Low-level wrapper for `::pdf_next_annot()`."""
14544: (4)              return _mupdf.ll_pdf_next_annot(annot)
14545: (0)          def ll_pdf_next_widget(previous):
14546: (4)              r""" Low-level wrapper for `::pdf_next_widget()`."""
14547: (4)              return _mupdf.ll_pdf_next_widget(previous)
14548: (0)          def ll_pdf_obj_is_dirty(obj):
14549: (4)              r""" Low-level wrapper for `::pdf_obj_is_dirty()`."""
14550: (4)              return _mupdf.ll_pdf_obj_is_dirty(obj)
14551: (0)          def ll_pdf_obj_is_incremental(obj):
14552: (4)              r""" Low-level wrapper for `::pdf_obj_is_incremental()`."""
14553: (4)              return _mupdf.ll_pdf_obj_is_incremental(obj)
14554: (0)          def ll_pdf_obj_marked(obj):
14555: (4)              r""" Low-level wrapper for `::pdf_obj_marked()`."""
14556: (4)              return _mupdf.ll_pdf_obj_marked(obj)
14557: (0)          def ll_pdf_obj_memo(obj, bit, memo):
14558: (4)              r"""
14559: (5)               Low-level wrapper for `::pdf_obj_memo()`.
14560: (4)              This function has out-params. Python/C# wrappers look like:
14561: (8)                  `ll_pdf_obj_memo(::pdf_obj *obj, int bit)` => `(int, int memo)`
14562: (4)              """
14563: (4)              return _mupdf.ll_pdf_obj_memo(obj, bit, memo)
14564: (0)          def ll_pdf_obj_num_is_stream(doc, num):
14565: (4)              r""" Low-level wrapper for `::pdf_obj_num_is_stream()`."""
14566: (4)              return _mupdf.ll_pdf_obj_num_is_stream(doc, num)
14567: (0)          def ll_pdf_obj_parent_num(obj):
14568: (4)              r""" Low-level wrapper for `::pdf_obj_parent_num()`."""
14569: (4)              return _mupdf.ll_pdf_obj_parent_num(obj)
14570: (0)          def ll_pdf_obj_refs(ref):
14571: (4)              r""" Low-level wrapper for `::pdf_obj_refs()`."""
14572: (4)              return _mupdf.ll_pdf_obj_refs(ref)
14573: (0)          def ll_pdf_objcmp(a, b):
14574: (4)              r""" Low-level wrapper for `::pdf_objcmp()`."""
14575: (4)              return _mupdf.ll_pdf_objcmp(a, b)
14576: (0)          def ll_pdf_objcmp_deep(a, b):
14577: (4)              r""" Low-level wrapper for `::pdf_objcmp_deep()`."""
14578: (4)              return _mupdf.ll_pdf_objcmp_deep(a, b)
14579: (0)          def ll_pdf_objcmp_resolve(a, b):
14580: (4)              r""" Low-level wrapper for `::pdf_objcmp_resolve()`."""
14581: (4)              return _mupdf.ll_pdf_objcmp_resolve(a, b)
14582: (0)          def ll_pdf_open_contents_stream(doc, obj):
14583: (4)              r""" Low-level wrapper for `::pdf_open_contents_stream()`."""
14584: (4)              return _mupdf.ll_pdf_open_contents_stream(doc, obj)
14585: (0)          def ll_pdf_open_crypt(chain, crypt, num, gen):
```

```
14586: (4)             r""" Low-level wrapper for `::pdf_open_crypt()`."""
14587: (4)             return _mupdf.ll_pdf_open_crypt(chain, crypt, num, gen)
14588: (0)         def ll_pdf_open_crypt_with_filter(chain, crypt, name, num, gen):
14589: (4)             r""" Low-level wrapper for `::pdf_open_crypt_with_filter()`."""
14590: (4)             return _mupdf.ll_pdf_open_crypt_with_filter(chain, crypt, name, num, gen)
14591: (0)         def ll_pdf_open_document(filename):
14592: (4)             r""" Low-level wrapper for `::pdf_open_document()`."""
14593: (4)             return _mupdf.ll_pdf_open_document(filename)
14594: (0)         def ll_pdf_open_document_with_stream(file):
14595: (4)             r""" Low-level wrapper for `::pdf_open_document_with_stream()`."""
14596: (4)             return _mupdf.ll_pdf_open_document_with_stream(file)
14597: (0)         def ll_pdf_open_inline_stream(doc, stmobj, length, chain, params):
14598: (4)             r""" Low-level wrapper for `::pdf_open_inline_stream()`."""
14599: (4)             return _mupdf.ll_pdf_open_inline_stream(doc, stmobj, length, chain,
params)
14600: (0)         def ll_pdf_open_raw_stream(ref):
14601: (4)             r""" Low-level wrapper for `::pdf_open_raw_stream()`."""
14602: (4)             return _mupdf.ll_pdf_open_raw_stream(ref)
14603: (0)         def ll_pdf_open_raw_stream_number(doc, num):
14604: (4)             r""" Low-level wrapper for `::pdf_open_raw_stream_number()`."""
14605: (4)             return _mupdf.ll_pdf_open_raw_stream_number(doc, num)
14606: (0)         def ll_pdf_open_stream(ref):
14607: (4)             r""" Low-level wrapper for `::pdf_open_stream()`."""
14608: (4)             return _mupdf.ll_pdf_open_stream(ref)
14609: (0)         def ll_pdf_open_stream_number(doc, num):
14610: (4)             r""" Low-level wrapper for `::pdf_open_stream_number()`."""
14611: (4)             return _mupdf.ll_pdf_open_stream_number(doc, num)
14612: (0)         def ll_pdf_open_stream_with_offset(doc, num, dict, stm_ofs):
14613: (4)             r""" Low-level wrapper for `::pdf_open_stream_with_offset()`."""
14614: (4)             return _mupdf.ll_pdf_open_stream_with_offset(doc, num, dict, stm_ofs)
14615: (0)         def ll_pdf_page_contents(page):
14616: (4)             r""" Low-level wrapper for `::pdf_page_contents()`."""
14617: (4)             return _mupdf.ll_pdf_page_contents(page)
14618: (0)         def ll_pdf_page_event_close(page):
14619: (4)             r""" Low-level wrapper for `::pdf_page_event_close()`."""
14620: (4)             return _mupdf.ll_pdf_page_event_close(page)
14621: (0)         def ll_pdf_page_event_open(page):
14622: (4)             r""" Low-level wrapper for `::pdf_page_event_open()`."""
14623: (4)             return _mupdf.ll_pdf_page_event_open(page)
14624: (0)         def ll_pdf_page_from_fz_page(ptr):
14625: (4)             r""" Low-level wrapper for `::pdf_page_from_fz_page()`."""
14626: (4)             return _mupdf.ll_pdf_page_from_fz_page(ptr)
14627: (0)         def ll_pdf_page_group(page):
14628: (4)             r""" Low-level wrapper for `::pdf_page_group()`."""
14629: (4)             return _mupdf.ll_pdf_page_group(page)
14630: (0)         def ll_pdf_page_has_transparency(page):
14631: (4)             r""" Low-level wrapper for `::pdf_page_has_transparency()`."""
14632: (4)             return _mupdf.ll_pdf_page_has_transparency(page)
14633: (0)         def ll_pdf_page_label(doc, page, buf, size):
14634: (4)             r""" Low-level wrapper for `::pdf_page_label()`."""
14635: (4)             return _mupdf.ll_pdf_page_label(doc, page, buf, size)
14636: (0)         def ll_pdf_page_label_imp(doc, chapter, page, buf, size):
14637: (4)             r""" Low-level wrapper for `::pdf_page_label_imp()`."""
14638: (4)             return _mupdf.ll_pdf_page_label_imp(doc, chapter, page, buf, size)
14639: (0)         def ll_pdf_page_obj_transform(pageobj, outbox, outctm):
14640: (4)             r""" Low-level wrapper for `::pdf_page_obj_transform()`."""
14641: (4)             return _mupdf.ll_pdf_page_obj_transform(pageobj, outbox, outctm)
14642: (0)         def ll_pdf_page_obj_transform_box(pageobj, outbox, out, box):
14643: (4)             r""" Low-level wrapper for `::pdf_page_obj_transform_box()`."""
14644: (4)             return _mupdf.ll_pdf_page_obj_transform_box(pageobj, outbox, out, box)
14645: (0)         def ll_pdf_page_presentation(page, transition, duration):
14646: (4)             r"""
14647: (5)              Low-level wrapper for `::pdf_page_presentation()`.
14648: (4)             This function has out-params. Python/C# wrappers look like:
14649: (8)                 `ll_pdf_page_presentation(::pdf_page *page, ::fz_transition
*transition)` => `(fz_transition *, float duration)`
14650: (4)             """
14651: (4)             return _mupdf.ll_pdf_page_presentation(page, transition, duration)
14652: (0)         def ll_pdf_page_resources(page):
```

```
14653: (4)              r""" Low-level wrapper for `::pdf_page_resources()`."""
14654: (4)              return _mupdf.ll_pdf_page_resources(page)
14655: (0)          def ll_pdf_page_separations(page):
14656: (4)              r""" Low-level wrapper for `::pdf_page_separations()`."""
14657: (4)              return _mupdf.ll_pdf_page_separations(page)
14658: (0)          def ll_pdf_page_transform(page, mediabox, ctm):
14659: (4)              r""" Low-level wrapper for `::pdf_page_transform()`."""
14660: (4)              return _mupdf.ll_pdf_page_transform(page, mediabox, ctm)
14661: (0)          def ll_pdf_page_transform_box(page, mediabox, ctm, box):
14662: (4)              r""" Low-level wrapper for `::pdf_page_transform_box()`."""
14663: (4)              return _mupdf.ll_pdf_page_transform_box(page, mediabox, ctm, box)
14664: (0)          def ll_pdf_page_write(doc, mediabox, presources, pcontents):
14665: (4)              r"""
14666: (5)               Low-level wrapper for `::pdf_page_write()`.
14667: (4)              This function has out-params. Python/C# wrappers look like:
14668: (8)                  `ll_pdf_page_write(::pdf_document *doc, ::fz_rect mediabox, ::pdf_obj
**presources, ::fz_buffer **pcontents)` => `(fz_device *)`
14669: (4)              """
14670: (4)              return _mupdf.ll_pdf_page_write(doc, mediabox, presources, pcontents)
14671: (0)          def ll_pdf_parse_array(doc, f, buf):
14672: (4)              r""" Low-level wrapper for `::pdf_parse_array()`."""
14673: (4)              return _mupdf.ll_pdf_parse_array(doc, f, buf)
14674: (0)          def ll_pdf_parse_date(s):
14675: (4)              r""" Low-level wrapper for `::pdf_parse_date()`."""
14676: (4)              return _mupdf.ll_pdf_parse_date(s)
14677: (0)          def ll_pdf_parse_default_appearance(da, font, size, n, color):
14678: (4)              r"""
14679: (5)               Low-level wrapper for `::pdf_parse_default_appearance()`.
14680: (4)              This function has out-params. Python/C# wrappers look like:
14681: (8)                  `ll_pdf_parse_default_appearance(const char *da, float color[4])` =>
`(const char *font, float size, int n)`
14682: (4)              """
14683: (4)              return _mupdf.ll_pdf_parse_default_appearance(da, font, size, n, color)
14684: (0)          def ll_pdf_parse_dict(doc, f, buf):
14685: (4)              r""" Low-level wrapper for `::pdf_parse_dict()`."""
14686: (4)              return _mupdf.ll_pdf_parse_dict(doc, f, buf)
14687: (0)          def ll_pdf_parse_ind_obj(doc, f, num, gen, stm_ofs, try_repair):
14688: (4)              r"""
14689: (5)               Low-level wrapper for `::pdf_parse_ind_obj()`.
14690: (4)              This function has out-params. Python/C# wrappers look like:
14691: (8)                  `ll_pdf_parse_ind_obj(::pdf_document *doc, ::fz_stream *f)` =>
`(pdf_obj *, int num, int gen, int64_t stm_ofs, int try_repair)`
14692: (4)              """
14693: (4)              return _mupdf.ll_pdf_parse_ind_obj(doc, f, num, gen, stm_ofs, try_repair)
14694: (0)          def ll_pdf_parse_journal_obj(doc, stm, onum, ostm, newobj):
14695: (4)              r"""
14696: (5)               Low-level wrapper for `::pdf_parse_journal_obj()`.
14697: (4)              This function has out-params. Python/C# wrappers look like:
14698: (8)                  `ll_pdf_parse_journal_obj(::pdf_document *doc, ::fz_stream *stm,
::fz_buffer **ostm)` => `(pdf_obj *, int onum, int newobj)`
14699: (4)              """
14700: (4)              return _mupdf.ll_pdf_parse_journal_obj(doc, stm, onum, ostm, newobj)
14701: (0)          def ll_pdf_parse_stm_obj(doc, f, buf):
14702: (4)              r""" Low-level wrapper for `::pdf_parse_stm_obj()`."""
14703: (4)              return _mupdf.ll_pdf_parse_stm_obj(doc, f, buf)
14704: (0)          def ll_pdf_parse_write_options(opts, args):
14705: (4)              r""" Low-level wrapper for `::pdf_parse_write_options()`."""
14706: (4)              return _mupdf.ll_pdf_parse_write_options(opts, args)
14707: (0)          def ll_pdf_pin_document(obj):
14708: (4)              r""" Low-level wrapper for `::pdf_pin_document()`."""
14709: (4)              return _mupdf.ll_pdf_pin_document(obj)
14710: (0)          def ll_pdf_preview_signature_as_display_list(w, h, lang, signer,
appearance_flags, graphic, reason, location):
14711: (4)              r""" Low-level wrapper for `::pdf_preview_signature_as_display_list()`."""
14712: (4)              return _mupdf.ll_pdf_preview_signature_as_display_list(w, h, lang, signer,
appearance_flags, graphic, reason, location)
14713: (0)          def ll_pdf_preview_signature_as_pixmap(w, h, lang, signer, appearance_flags,
graphic, reason, location):
14714: (4)              r""" Low-level wrapper for `::pdf_preview_signature_as_pixmap()`."""
```

```
14715: (4)              return _mupdf.ll_pdf_preview_signature_as_pixmap(w, h, lang, signer,
appearance_flags, graphic, reason, location)
14716: (0)          def ll_pdf_print_crypt(out, crypt):
14717: (4)              r""" Low-level wrapper for `::pdf_print_crypt()`."""
14718: (4)              return _mupdf.ll_pdf_print_crypt(out, crypt)
14719: (0)          def ll_pdf_print_default_appearance(buf, nbuf, font, size, n, color):
14720: (4)              r""" Low-level wrapper for `::pdf_print_default_appearance()`."""
14721: (4)              return _mupdf.ll_pdf_print_default_appearance(buf, nbuf, font, size, n,
color)
14722: (0)          def ll_pdf_print_encrypted_obj(out, obj, tight, ascii, crypt, num, gen, sep):
14723: (4)              r"""
14724: (5)               Low-level wrapper for `::pdf_print_encrypted_obj()`.
14725: (4)              This function has out-params. Python/C# wrappers look like:
14726: (8)                  `ll_pdf_print_encrypted_obj(::fz_output *out, ::pdf_obj *obj, int
tight, int ascii, ::pdf_crypt *crypt, int num, int gen)` => int sep
14727: (4)              """
14728: (4)              return _mupdf.ll_pdf_print_encrypted_obj(out, obj, tight, ascii, crypt,
num, gen, sep)
14729: (0)          def ll_pdf_print_font(out, fontdesc):
14730: (4)              r""" Low-level wrapper for `::pdf_print_font()`."""
14731: (4)              return _mupdf.ll_pdf_print_font(out, fontdesc)
14732: (0)          def ll_pdf_print_obj(out, obj, tight, ascii):
14733: (4)              r""" Low-level wrapper for `::pdf_print_obj()`."""
14734: (4)              return _mupdf.ll_pdf_print_obj(out, obj, tight, ascii)
14735: (0)          def ll_pdf_process_annot(proc, annot, cookie):
14736: (4)              r""" Low-level wrapper for `::pdf_process_annot()`."""
14737: (4)              return _mupdf.ll_pdf_process_annot(proc, annot, cookie)
14738: (0)          def ll_pdf_process_contents(proc, doc, res, stm, cookie, out_res):
14739: (4)              r"""
14740: (5)               Low-level wrapper for `::pdf_process_contents()`.
14741: (4)              This function has out-params. Python/C# wrappers look like:
14742: (8)                  `ll_pdf_process_contents(::pdf_processor *proc, ::pdf_document *doc,
::pdf_obj *res, ::pdf_obj *stm, ::fz_cookie *cookie, ::pdf_obj **out_res)` =>
14743: (4)              """
14744: (4)              return _mupdf.ll_pdf_process_contents(proc, doc, res, stm, cookie,
out_res)
14745: (0)          def ll_pdf_process_glyph(proc, doc, resources, contents):
14746: (4)              r""" Low-level wrapper for `::pdf_process_glyph()`."""
14747: (4)              return _mupdf.ll_pdf_process_glyph(proc, doc, resources, contents)
14748: (0)          def ll_pdf_process_raw_contents(proc, doc, rdb, stmobj, cookie):
14749: (4)              r""" Low-level wrapper for `::pdf_process_raw_contents()`."""
14750: (4)              return _mupdf.ll_pdf_process_raw_contents(proc, doc, rdb, stmobj, cookie)
14751: (0)          def ll_pdf_processor_pop_resources(proc):
14752: (4)              r""" Low-level wrapper for `::pdf_processor_pop_resources()`."""
14753: (4)              return _mupdf.ll_pdf_processor_pop_resources(proc)
14754: (0)          def ll_pdf_processor_push_resources(proc, res):
14755: (4)              r""" Low-level wrapper for `::pdf_processor_push_resources()`."""
14756: (4)              return _mupdf.ll_pdf_processor_push_resources(proc, res)
14757: (0)          def ll_pdf_progressive_advance(doc, pagenum):
14758: (4)              r""" Low-level wrapper for `::pdf_progressive_advance()`."""
14759: (4)              return _mupdf.ll_pdf_progressive_advance(doc, pagenum)
14760: (0)          def ll_pdf_purge_local_font_resources(doc):
14761: (4)              r""" Low-level wrapper for `::pdf_purge_local_font_resources()`."""
14762: (4)              return _mupdf.ll_pdf_purge_local_font_resources(doc)
14763: (0)          def ll_pdf_purge_locals_from_store(doc):
14764: (4)              r""" Low-level wrapper for `::pdf_purge_locals_from_store()`."""
14765: (4)              return _mupdf.ll_pdf_purge_locals_from_store(doc)
14766: (0)          def ll_pdf_read_journal(doc, stm):
14767: (4)              r""" Low-level wrapper for `::pdf_read_journal()`."""
14768: (4)              return _mupdf.ll_pdf_read_journal(doc, stm)
14769: (0)          def ll_pdf_read_ocg(doc):
14770: (4)              r""" Low-level wrapper for `::pdf_read_ocg()`."""
14771: (4)              return _mupdf.ll_pdf_read_ocg(doc)
14772: (0)          def ll_pdf_rearrange_pages(doc, count, pages):
14773: (4)              r""" Low-level wrapper for `::pdf_rearrange_pages()`."""
14774: (4)              return _mupdf.ll_pdf_rearrange_pages(doc, count, pages)
14775: (0)          def ll_pdf_rearrange_pages2(doc, pages):
14776: (4)              r""" Low-level wrapper for `::pdf_rearrange_pages2()`.  Swig-friendly
wrapper for pdf_rearrange_pages()."""
```

```
14777: (4)              return _mupdf.ll_pdf_rearrange_pages2(doc, pages)
14778: (0)          def ll_pdf_recolor_shade(shade, reshade, opaque):
14779: (4)              r"""
14780: (4)              Low-level wrapper for `::pdf_recolor_shade()`.
14781: (4)              Recolor a shade.
14782: (4)              """
14783: (4)              return _mupdf.ll_pdf_recolor_shade(shade, reshade, opaque)
14784: (0)          def ll_pdf_redact_page(doc, page, opts):
14785: (4)              r""" Low-level wrapper for `::pdf_redact_page()`."""
14786: (4)              return _mupdf.ll_pdf_redact_page(doc, page, opts)
14787: (0)          def ll_pdf_redo(doc):
14788: (4)              r""" Low-level wrapper for `::pdf_redo()`."""
14789: (4)              return _mupdf.ll_pdf_redo(doc)
14790: (0)          def ll_pdf_remove_item(drop, key):
14791: (4)              r""" Low-level wrapper for `::pdf_remove_item()`."""
14792: (4)              return _mupdf.ll_pdf_remove_item(drop, key)
14793: (0)          def ll_pdf_repair_obj(doc, buf, stmofsp, stmlenp, encrypt, id, page, tmpofs,
root):
14794: (4)              r"""
14795: (5)               Low-level wrapper for `::pdf_repair_obj()`.
14796: (4)              This function has out-params. Python/C# wrappers look like:
14797: (8)                  `ll_pdf_repair_obj(::pdf_document *doc, ::pdf_lexbuf *buf, ::pdf_obj
**encrypt, ::pdf_obj **id, ::pdf_obj **page, ::pdf_obj **root)` => `(int, int64_t stmofsp, int64_t
stmlenp, int64_t tmpofs)`
14798: (4)              """
14799: (4)              return _mupdf.ll_pdf_repair_obj(doc, buf, stmofsp, stmlenp, encrypt, id,
page, tmpofs, root)
14800: (0)          def ll_pdf_repair_obj_stms(doc):
14801: (4)              r""" Low-level wrapper for `::pdf_repair_obj_stms()`."""
14802: (4)              return _mupdf.ll_pdf_repair_obj_stms(doc)
14803: (0)          def ll_pdf_repair_trailer(doc):
14804: (4)              r""" Low-level wrapper for `::pdf_repair_trailer()`."""
14805: (4)              return _mupdf.ll_pdf_repair_trailer(doc)
14806: (0)          def ll_pdf_repair_xref(doc):
14807: (4)              r""" Low-level wrapper for `::pdf_repair_xref()`."""
14808: (4)              return _mupdf.ll_pdf_repair_xref(doc)
14809: (0)          def ll_pdf_replace_xref(doc, entries, n):
14810: (4)              r""" Low-level wrapper for `::pdf_replace_xref()`."""
14811: (4)              return _mupdf.ll_pdf_replace_xref(doc, entries, n)
14812: (0)          def ll_pdf_reset_form(doc, fields, exclude):
14813: (4)              r""" Low-level wrapper for `::pdf_reset_form()`."""
14814: (4)              return _mupdf.ll_pdf_reset_form(doc, fields, exclude)
14815: (0)          def ll_pdf_reset_processor(proc):
14816: (4)              r""" Low-level wrapper for `::pdf_reset_processor()`."""
14817: (4)              return _mupdf.ll_pdf_reset_processor(proc)
14818: (0)          def ll_pdf_resolve_indirect(ref):
14819: (4)              r"""
14820: (4)              Low-level wrapper for `::pdf_resolve_indirect()`.
14821: (4)              Resolve an indirect object (or chain of objects).
14822: (4)              This can cause xref reorganisations (solidifications etc) due to
14823: (4)              repairs, so all held pdf_xref_entries should be considered
14824: (4)              invalid after this call (other than the returned one).
14825: (4)              """
14826: (4)              return _mupdf.ll_pdf_resolve_indirect(ref)
14827: (0)          def ll_pdf_resolve_indirect_chain(ref):
14828: (4)              r""" Low-level wrapper for `::pdf_resolve_indirect_chain()`."""
14829: (4)              return _mupdf.ll_pdf_resolve_indirect_chain(ref)
14830: (0)          def ll_pdf_resolve_link(doc, uri, xp, yp):
14831: (4)              r"""
14832: (5)               Low-level wrapper for `::pdf_resolve_link()`.
14833: (4)              This function has out-params. Python/C# wrappers look like:
14834: (8)                  `ll_pdf_resolve_link(::pdf_document *doc, const char *uri)` => `(int,
float xp, float yp)`
14835: (4)              """
14836: (4)              return _mupdf.ll_pdf_resolve_link(doc, uri, xp, yp)
14837: (0)          def ll_pdf_resolve_link_dest(doc, uri):
14838: (4)              r""" Low-level wrapper for `::pdf_resolve_link_dest()`."""
14839: (4)              return _mupdf.ll_pdf_resolve_link_dest(doc, uri)
14840: (0)          def ll_pdf_rewrite_images(doc, opts):
```

```
14841: (4)             r""" Low-level wrapper for `::pdf_rewrite_images()`."""
14842: (4)             return _mupdf.ll_pdf_rewrite_images(doc, opts)
14843: (0)         def ll_pdf_run_annot(annot, dev, ctm, cookie):
14844: (4)             r""" Low-level wrapper for `::pdf_run_annot()`."""
14845: (4)             return _mupdf.ll_pdf_run_annot(annot, dev, ctm, cookie)
14846: (0)         def ll_pdf_run_document_structure(doc, dev, cookie):
14847: (4)             r""" Low-level wrapper for `::pdf_run_document_structure()`."""
14848: (4)             return _mupdf.ll_pdf_run_document_structure(doc, dev, cookie)
14849: (0)         def ll_pdf_run_glyph(doc, resources, contents, dev, ctm, gstate, default_cs):
14850: (4)             r""" Low-level wrapper for `::pdf_run_glyph()`."""
14851: (4)             return _mupdf.ll_pdf_run_glyph(doc, resources, contents, dev, ctm, gstate,
default_cs)
14852: (0)         def ll_pdf_run_page(page, dev, ctm, cookie):
14853: (4)             r""" Low-level wrapper for `::pdf_run_page()`."""
14854: (4)             return _mupdf.ll_pdf_run_page(page, dev, ctm, cookie)
14855: (0)         def ll_pdf_run_page_annots(page, dev, ctm, cookie):
14856: (4)             r""" Low-level wrapper for `::pdf_run_page_annots()`."""
14857: (4)             return _mupdf.ll_pdf_run_page_annots(page, dev, ctm, cookie)
14858: (0)         def ll_pdf_run_page_annots_with_usage(page, dev, ctm, usage, cookie):
14859: (4)             r""" Low-level wrapper for `::pdf_run_page_annots_with_usage()`."""
14860: (4)             return _mupdf.ll_pdf_run_page_annots_with_usage(page, dev, ctm, usage,
cookie)
14861: (0)         def ll_pdf_run_page_contents(page, dev, ctm, cookie):
14862: (4)             r""" Low-level wrapper for `::pdf_run_page_contents()`."""
14863: (4)             return _mupdf.ll_pdf_run_page_contents(page, dev, ctm, cookie)
14864: (0)         def ll_pdf_run_page_contents_with_usage(page, dev, ctm, usage, cookie):
14865: (4)             r""" Low-level wrapper for `::pdf_run_page_contents_with_usage()`."""
14866: (4)             return _mupdf.ll_pdf_run_page_contents_with_usage(page, dev, ctm, usage,
cookie)
14867: (0)         def ll_pdf_run_page_widgets(page, dev, ctm, cookie):
14868: (4)             r""" Low-level wrapper for `::pdf_run_page_widgets()`."""
14869: (4)             return _mupdf.ll_pdf_run_page_widgets(page, dev, ctm, cookie)
14870: (0)         def ll_pdf_run_page_widgets_with_usage(page, dev, ctm, usage, cookie):
14871: (4)             r""" Low-level wrapper for `::pdf_run_page_widgets_with_usage()`."""
14872: (4)             return _mupdf.ll_pdf_run_page_widgets_with_usage(page, dev, ctm, usage,
cookie)
14873: (0)         def ll_pdf_run_page_with_usage(page, dev, ctm, usage, cookie):
14874: (4)             r""" Low-level wrapper for `::pdf_run_page_with_usage()`."""
14875: (4)             return _mupdf.ll_pdf_run_page_with_usage(page, dev, ctm, usage, cookie)
14876: (0)         def ll_pdf_sample_shade_function(samples, n, funcs, func, t0, t1):
14877: (4)             r"""
14878: (5)              Low-level wrapper for `::pdf_sample_shade_function()`.
14879: (4)             This function has out-params. Python/C# wrappers look like:
14880: (8)                 `ll_pdf_sample_shade_function(int n, int funcs, ::pdf_function **func,
float t0, float t1)` => `(float samples)`
14881: (4)             """
14882: (4)             return _mupdf.ll_pdf_sample_shade_function(samples, n, funcs, func, t0,
t1)
14883: (0)         def ll_pdf_save_document(doc, filename, opts):
14884: (4)             r""" Low-level wrapper for `::pdf_save_document()`."""
14885: (4)             return _mupdf.ll_pdf_save_document(doc, filename, opts)
14886: (0)         def ll_pdf_save_journal(doc, filename):
14887: (4)             r""" Low-level wrapper for `::pdf_save_journal()`."""
14888: (4)             return _mupdf.ll_pdf_save_journal(doc, filename)
14889: (0)         def ll_pdf_save_snapshot(doc, filename):
14890: (4)             r""" Low-level wrapper for `::pdf_save_snapshot()`."""
14891: (4)             return _mupdf.ll_pdf_save_snapshot(doc, filename)
14892: (0)         def ll_pdf_select_layer_config(doc, config_num):
14893: (4)             r""" Low-level wrapper for `::pdf_select_layer_config()`."""
14894: (4)             return _mupdf.ll_pdf_select_layer_config(doc, config_num)
14895: (0)         def ll_pdf_select_layer_config_ui(doc, ui):
14896: (4)             r""" Low-level wrapper for `::pdf_select_layer_config_ui()`."""
14897: (4)             return _mupdf.ll_pdf_select_layer_config_ui(doc, ui)
14898: (0)         def ll_pdf_serialise_journal(doc, out):
14899: (4)             r""" Low-level wrapper for `::pdf_serialise_journal()`."""
14900: (4)             return _mupdf.ll_pdf_serialise_journal(doc, out)
14901: (0)         def ll_pdf_set_annot_active(annot, active):
14902: (4)             r""" Low-level wrapper for `::pdf_set_annot_active()`."""
14903: (4)             return _mupdf.ll_pdf_set_annot_active(annot, active)
```

```
14904: (0)          def ll_pdf_set_annot_appearance(annot, appearance, state, ctm, bbox, res,
contents):
14905: (4)              r""" Low-level wrapper for `::pdf_set_annot_appearance()`."""
14906: (4)              return _mupdf.ll_pdf_set_annot_appearance(annot, appearance, state, ctm,
bbox, res, contents)
14907: (0)          def ll_pdf_set_annot_appearance_from_display_list(annot, appearance, state,
ctm, list):
14908: (4)              r""" Low-level wrapper for
`::pdf_set_annot_appearance_from_display_list()`."""
14909: (4)              return _mupdf.ll_pdf_set_annot_appearance_from_display_list(annot,
appearance, state, ctm, list)
14910: (0)          def ll_pdf_set_annot_author(annot, author):
14911: (4)              r""" Low-level wrapper for `::pdf_set_annot_author()`."""
14912: (4)              return _mupdf.ll_pdf_set_annot_author(annot, author)
14913: (0)          def ll_pdf_set_annot_border(annot, width):
14914: (4)              r""" Low-level wrapper for `::pdf_set_annot_border()`."""
14915: (4)              return _mupdf.ll_pdf_set_annot_border(annot, width)
14916: (0)          def ll_pdf_set_annot_border_effect(annot, effect):
14917: (4)              r""" Low-level wrapper for `::pdf_set_annot_border_effect()`."""
14918: (4)              return _mupdf.ll_pdf_set_annot_border_effect(annot, effect)
14919: (0)          def ll_pdf_set_annot_border_effect_intensity(annot, intensity):
14920: (4)              r""" Low-level wrapper for `::pdf_set_annot_border_effect_intensity()`."""
14921: (4)              return _mupdf.ll_pdf_set_annot_border_effect_intensity(annot, intensity)
14922: (0)          def ll_pdf_set_annot_border_style(annot, style):
14923: (4)              r""" Low-level wrapper for `::pdf_set_annot_border_style()`."""
14924: (4)              return _mupdf.ll_pdf_set_annot_border_style(annot, style)
14925: (0)          def ll_pdf_set_annot_border_width(annot, width):
14926: (4)              r""" Low-level wrapper for `::pdf_set_annot_border_width()`."""
14927: (4)              return _mupdf.ll_pdf_set_annot_border_width(annot, width)
14928: (0)          def ll_pdf_set_annot_color(annot, n, color):
14929: (4)              r""" Low-level wrapper for `::pdf_set_annot_color()`."""
14930: (4)              return _mupdf.ll_pdf_set_annot_color(annot, n, color)
14931: (0)          def ll_pdf_set_annot_contents(annot, text):
14932: (4)              r""" Low-level wrapper for `::pdf_set_annot_contents()`."""
14933: (4)              return _mupdf.ll_pdf_set_annot_contents(annot, text)
14934: (0)          def ll_pdf_set_annot_creation_date(annot, time):
14935: (4)              r""" Low-level wrapper for `::pdf_set_annot_creation_date()`."""
14936: (4)              return _mupdf.ll_pdf_set_annot_creation_date(annot, time)
14937: (0)          def ll_pdf_set_annot_default_appearance(annot, font, size, n, color):
14938: (4)              r""" Low-level wrapper for `::pdf_set_annot_default_appearance()`."""
14939: (4)              return _mupdf.ll_pdf_set_annot_default_appearance(annot, font, size, n,
color)
14940: (0)          def ll_pdf_set_annot_field_value(doc, widget, text, ignore_trigger_events):
14941: (4)              r""" Low-level wrapper for `::pdf_set_annot_field_value()`."""
14942: (4)              return _mupdf.ll_pdf_set_annot_field_value(doc, widget, text,
ignore_trigger_events)
14943: (0)          def ll_pdf_set_annot_filespec(annot, obj):
14944: (4)              r""" Low-level wrapper for `::pdf_set_annot_filespec()`."""
14945: (4)              return _mupdf.ll_pdf_set_annot_filespec(annot, obj)
14946: (0)          def ll_pdf_set_annot_flags(annot, flags):
14947: (4)              r""" Low-level wrapper for `::pdf_set_annot_flags()`."""
14948: (4)              return _mupdf.ll_pdf_set_annot_flags(annot, flags)
14949: (0)          def ll_pdf_set_annot_hidden_for_editing(annot, hidden):
14950: (4)              r""" Low-level wrapper for `::pdf_set_annot_hidden_for_editing()`."""
14951: (4)              return _mupdf.ll_pdf_set_annot_hidden_for_editing(annot, hidden)
14952: (0)          def ll_pdf_set_annot_hot(annot, hot):
14953: (4)              r""" Low-level wrapper for `::pdf_set_annot_hot()`."""
14954: (4)              return _mupdf.ll_pdf_set_annot_hot(annot, hot)
14955: (0)          def ll_pdf_set_annot_icon_name(annot, name):
14956: (4)              r""" Low-level wrapper for `::pdf_set_annot_icon_name()`."""
14957: (4)              return _mupdf.ll_pdf_set_annot_icon_name(annot, name)
14958: (0)          def ll_pdf_set_annot_ink_list(annot, n, count, v):
14959: (4)              r""" Low-level wrapper for `::pdf_set_annot_ink_list()`."""
14960: (4)              return _mupdf.ll_pdf_set_annot_ink_list(annot, n, count, v)
14961: (0)          def ll_pdf_set_annot_intent(annot, it):
14962: (4)              r""" Low-level wrapper for `::pdf_set_annot_intent()`."""
14963: (4)              return _mupdf.ll_pdf_set_annot_intent(annot, it)
14964: (0)          def ll_pdf_set_annot_interior_color(annot, n, color):
14965: (4)              r""" Low-level wrapper for `::pdf_set_annot_interior_color()`."""
```

```
14966: (4)             return _mupdf.ll_pdf_set_annot_interior_color(annot, n, color)
14967: (0)         def ll_pdf_set_annot_is_open(annot, is_open):
14968: (4)             r""" Low-level wrapper for `::pdf_set_annot_is_open()`."""
14969: (4)             return _mupdf.ll_pdf_set_annot_is_open(annot, is_open)
14970: (0)         def ll_pdf_set_annot_language(annot, lang):
14971: (4)             r""" Low-level wrapper for `::pdf_set_annot_language()`."""
14972: (4)             return _mupdf.ll_pdf_set_annot_language(annot, lang)
14973: (0)         def ll_pdf_set_annot_line(annot, a, b):
14974: (4)             r""" Low-level wrapper for `::pdf_set_annot_line()`."""
14975: (4)             return _mupdf.ll_pdf_set_annot_line(annot, a, b)
14976: (0)         def ll_pdf_set_annot_line_end_style(annot, e):
14977: (4)             r""" Low-level wrapper for `::pdf_set_annot_line_end_style()`."""
14978: (4)             return _mupdf.ll_pdf_set_annot_line_end_style(annot, e)
14979: (0)         def ll_pdf_set_annot_line_ending_styles(annot, start_style, end_style):
14980: (4)             r""" Low-level wrapper for `::pdf_set_annot_line_ending_styles()`."""
14981: (4)             return _mupdf.ll_pdf_set_annot_line_ending_styles(annot, start_style,
end_style)
14982: (0)         def ll_pdf_set_annot_line_start_style(annot, s):
14983: (4)             r""" Low-level wrapper for `::pdf_set_annot_line_start_style()`."""
14984: (4)             return _mupdf.ll_pdf_set_annot_line_start_style(annot, s)
14985: (0)         def ll_pdf_set_annot_modification_date(annot, time):
14986: (4)             r""" Low-level wrapper for `::pdf_set_annot_modification_date()`."""
14987: (4)             return _mupdf.ll_pdf_set_annot_modification_date(annot, time)
14988: (0)         def ll_pdf_set_annot_opacity(annot, opacity):
14989: (4)             r""" Low-level wrapper for `::pdf_set_annot_opacity()`."""
14990: (4)             return _mupdf.ll_pdf_set_annot_opacity(annot, opacity)
14991: (0)         def ll_pdf_set_annot_popup(annot, rect):
14992: (4)             r""" Low-level wrapper for `::pdf_set_annot_popup()`."""
14993: (4)             return _mupdf.ll_pdf_set_annot_popup(annot, rect)
14994: (0)         def ll_pdf_set_annot_quad_points(annot, n, qv):
14995: (4)             r""" Low-level wrapper for `::pdf_set_annot_quad_points()`."""
14996: (4)             return _mupdf.ll_pdf_set_annot_quad_points(annot, n, qv)
14997: (0)         def ll_pdf_set_annot_quadding(annot, q):
14998: (4)             r""" Low-level wrapper for `::pdf_set_annot_quadding()`."""
14999: (4)             return _mupdf.ll_pdf_set_annot_quadding(annot, q)
15000: (0)         def ll_pdf_set_annot_rect(annot, rect):
15001: (4)             r""" Low-level wrapper for `::pdf_set_annot_rect()`."""
15002: (4)             return _mupdf.ll_pdf_set_annot_rect(annot, rect)
15003: (0)         def ll_pdf_set_annot_resynthesised(annot):
15004: (4)             r""" Low-level wrapper for `::pdf_set_annot_resynthesised()`."""
15005: (4)             return _mupdf.ll_pdf_set_annot_resynthesised(annot)
15006: (0)         def ll_pdf_set_annot_stamp_image(annot, image):
15007: (4)             r""" Low-level wrapper for `::pdf_set_annot_stamp_image()`."""
15008: (4)             return _mupdf.ll_pdf_set_annot_stamp_image(annot, image)
15009: (0)         def ll_pdf_set_annot_vertex(annot, i, p):
15010: (4)             r""" Low-level wrapper for `::pdf_set_annot_vertex()`."""
15011: (4)             return _mupdf.ll_pdf_set_annot_vertex(annot, i, p)
15012: (0)         def ll_pdf_set_annot_vertices(annot, n, v):
15013: (4)             r""" Low-level wrapper for `::pdf_set_annot_vertices()`."""
15014: (4)             return _mupdf.ll_pdf_set_annot_vertices(annot, n, v)
15015: (0)         def ll_pdf_set_choice_field_value(widget, value):
15016: (4)             r""" Low-level wrapper for `::pdf_set_choice_field_value()`."""
15017: (4)             return _mupdf.ll_pdf_set_choice_field_value(widget, value)
15018: (0)         def ll_pdf_set_cmap_wmode(cmap, wmode):
15019: (4)             r""" Low-level wrapper for `::pdf_set_cmap_wmode()`."""
15020: (4)             return _mupdf.ll_pdf_set_cmap_wmode(cmap, wmode)
15021: (0)         def ll_pdf_set_default_hmtx(font, w):
15022: (4)             r""" Low-level wrapper for `::pdf_set_default_hmtx()`."""
15023: (4)             return _mupdf.ll_pdf_set_default_hmtx(font, w)
15024: (0)         def ll_pdf_set_default_vmtx(font, y, w):
15025: (4)             r""" Low-level wrapper for `::pdf_set_default_vmtx()`."""
15026: (4)             return _mupdf.ll_pdf_set_default_vmtx(font, y, w)
15027: (0)         def ll_pdf_set_doc_event_callback(doc, event_cb, free_event_data_cb, data):
15028: (4)             r""" Low-level wrapper for `::pdf_set_doc_event_callback()`."""
15029: (4)             return _mupdf.ll_pdf_set_doc_event_callback(doc, event_cb,
free_event_data_cb, data)
15030: (0)         def ll_pdf_set_document_language(doc, lang):
15031: (4)             r""" Low-level wrapper for `::pdf_set_document_language()`."""
15032: (4)             return _mupdf.ll_pdf_set_document_language(doc, lang)
```

```
15033: (0)          def ll_pdf_set_field_value(doc, field, text, ignore_trigger_events):
15034: (4)              r""" Low-level wrapper for `::pdf_set_field_value()`."""
15035: (4)              return _mupdf.ll_pdf_set_field_value(doc, field, text,
ignore_trigger_events)
15036: (0)          def ll_pdf_set_font_wmode(font, wmode):
15037: (4)              r""" Low-level wrapper for `::pdf_set_font_wmode()`."""
15038: (4)              return _mupdf.ll_pdf_set_font_wmode(font, wmode)
15039: (0)          def ll_pdf_set_int(obj, i):
15040: (4)              r""" Low-level wrapper for `::pdf_set_int()`."""
15041: (4)              return _mupdf.ll_pdf_set_int(obj, i)
15042: (0)          def ll_pdf_set_layer_config_as_default(doc):
15043: (4)              r""" Low-level wrapper for `::pdf_set_layer_config_as_default()`."""
15044: (4)              return _mupdf.ll_pdf_set_layer_config_as_default(doc)
15045: (0)          def ll_pdf_set_obj_memo(obj, bit, memo):
15046: (4)              r""" Low-level wrapper for `::pdf_set_obj_memo()`."""
15047: (4)              return _mupdf.ll_pdf_set_obj_memo(obj, bit, memo)
15048: (0)          def ll_pdf_set_obj_parent(obj, num):
15049: (4)              r""" Low-level wrapper for `::pdf_set_obj_parent()`."""
15050: (4)              return _mupdf.ll_pdf_set_obj_parent(obj, num)
15051: (0)          def ll_pdf_set_page_box(page, box, rect):
15052: (4)              r""" Low-level wrapper for `::pdf_set_page_box()`."""
15053: (4)              return _mupdf.ll_pdf_set_page_box(page, box, rect)
15054: (0)          def ll_pdf_set_page_labels(doc, index, style, prefix, start):
15055: (4)              r""" Low-level wrapper for `::pdf_set_page_labels()`."""
15056: (4)              return _mupdf.ll_pdf_set_page_labels(doc, index, style, prefix, start)
15057: (0)          def ll_pdf_set_populating_xref_trailer(doc, trailer):
15058: (4)              r""" Low-level wrapper for `::pdf_set_populating_xref_trailer()`."""
15059: (4)              return _mupdf.ll_pdf_set_populating_xref_trailer(doc, trailer)
15060: (0)          def ll_pdf_set_str_len(obj, newlen):
15061: (4)              r""" Low-level wrapper for `::pdf_set_str_len()`."""
15062: (4)              return _mupdf.ll_pdf_set_str_len(obj, newlen)
15063: (0)          def ll_pdf_set_text_field_value(widget, value):
15064: (4)              r""" Low-level wrapper for `::pdf_set_text_field_value()`."""
15065: (4)              return _mupdf.ll_pdf_set_text_field_value(widget, value)
15066: (0)          def ll_pdf_set_usecmap(cmap, usecmap):
15067: (4)              r""" Low-level wrapper for `::pdf_set_usecmap()`."""
15068: (4)              return _mupdf.ll_pdf_set_usecmap(cmap, usecmap)
15069: (0)          def ll_pdf_set_widget_editing_state(widget, editing):
15070: (4)              r""" Low-level wrapper for `::pdf_set_widget_editing_state()`."""
15071: (4)              return _mupdf.ll_pdf_set_widget_editing_state(widget, editing)
15072: (0)          def ll_pdf_sign_signature(widget, signer, appearance_flags, graphic, reason,
location):
15073: (4)              r""" Low-level wrapper for `::pdf_sign_signature()`."""
15074: (4)              return _mupdf.ll_pdf_sign_signature(widget, signer, appearance_flags,
graphic, reason, location)
15075: (0)          def ll_pdf_sign_signature_with_appearance(widget, signer, date, disp_list):
15076: (4)              r""" Low-level wrapper for `::pdf_sign_signature_with_appearance()`."""
15077: (4)              return _mupdf.ll_pdf_sign_signature_with_appearance(widget, signer, date,
disp_list)
15078: (0)          def ll_pdf_signature_appearance_signed(rect, lang, img, left_text, right_text,
include_logo):
15079: (4)              r""" Low-level wrapper for `::pdf_signature_appearance_signed()`."""
15080: (4)              return _mupdf.ll_pdf_signature_appearance_signed(rect, lang, img,
left_text, right_text, include_logo)
15081: (0)          def ll_pdf_signature_appearance_unsigned(rect, lang):
15082: (4)              r""" Low-level wrapper for `::pdf_signature_appearance_unsigned()`."""
15083: (4)              return _mupdf.ll_pdf_signature_appearance_unsigned(rect, lang)
15084: (0)          def ll_pdf_signature_byte_range(doc, signature, byte_range):
15085: (4)              r""" Low-level wrapper for `::pdf_signature_byte_range()`."""
15086: (4)              return _mupdf.ll_pdf_signature_byte_range(doc, signature, byte_range)
15087: (0)          def ll_pdf_signature_contents(doc, signature, contents):
15088: (4)              r"""
15089: (5)               Low-level wrapper for `::pdf_signature_contents()`.
15090: (4)              This function has out-params. Python/C# wrappers look like:
15091: (8)                  `ll_pdf_signature_contents(::pdf_document *doc, ::pdf_obj *signature)`
=> `(size_t, char *contents)`
15092: (4)              """
15093: (4)              return _mupdf.ll_pdf_signature_contents(doc, signature, contents)
15094: (0)          def ll_pdf_signature_drop_distinguished_name(name):
```

```
15095: (4)              r""" Low-level wrapper for `::pdf_signature_drop_distinguished_name()`."""
15096: (4)              return _mupdf.ll_pdf_signature_drop_distinguished_name(name)
15097: (0)          def ll_pdf_signature_error_description(err):
15098: (4)              r""" Low-level wrapper for `::pdf_signature_error_description()`."""
15099: (4)              return _mupdf.ll_pdf_signature_error_description(err)
15100: (0)          def ll_pdf_signature_format_distinguished_name(name):
15101: (4)              r""" Low-level wrapper for
`::pdf_signature_format_distinguished_name()`."""
15102: (4)              return _mupdf.ll_pdf_signature_format_distinguished_name(name)
15103: (0)          def ll_pdf_signature_get_signatory(verifier, doc, signature):
15104: (4)              r""" Low-level wrapper for `::pdf_signature_get_signatory()`."""
15105: (4)              return _mupdf.ll_pdf_signature_get_signatory(verifier, doc, signature)
15106: (0)          def ll_pdf_signature_get_widget_signatory(verifier, widget):
15107: (4)              r""" Low-level wrapper for `::pdf_signature_get_widget_signatory()`."""
15108: (4)              return _mupdf.ll_pdf_signature_get_widget_signatory(verifier, widget)
15109: (0)          def ll_pdf_signature_hash_bytes(doc, signature):
15110: (4)              r""" Low-level wrapper for `::pdf_signature_hash_bytes()`."""
15111: (4)              return _mupdf.ll_pdf_signature_hash_bytes(doc, signature)
15112: (0)          def ll_pdf_signature_incremental_change_since_signing(doc, signature):
15113: (4)              r""" Low-level wrapper for
`::pdf_signature_incremental_change_since_signing()`."""
15114: (4)              return _mupdf.ll_pdf_signature_incremental_change_since_signing(doc,
signature)
15115: (0)          def ll_pdf_signature_info(name, dn, reason, location, date, include_labels):
15116: (4)              r""" Low-level wrapper for `::pdf_signature_info()`."""
15117: (4)              return _mupdf.ll_pdf_signature_info(name, dn, reason, location, date,
include_labels)
15118: (0)          def ll_pdf_signature_is_signed(doc, field):
15119: (4)              r""" Low-level wrapper for `::pdf_signature_is_signed()`."""
15120: (4)              return _mupdf.ll_pdf_signature_is_signed(doc, field)
15121: (0)          def ll_pdf_signature_set_value(doc, field, signer, stime):
15122: (4)              r""" Low-level wrapper for `::pdf_signature_set_value()`."""
15123: (4)              return _mupdf.ll_pdf_signature_set_value(doc, field, signer, stime)
15124: (0)          def ll_pdf_sort_cmap(cmap):
15125: (4)              r""" Low-level wrapper for `::pdf_sort_cmap()`."""
15126: (4)              return _mupdf.ll_pdf_sort_cmap(cmap)
15127: (0)          def ll_pdf_sort_dict(dict):
15128: (4)              r""" Low-level wrapper for `::pdf_sort_dict()`."""
15129: (4)              return _mupdf.ll_pdf_sort_dict(dict)
15130: (0)          def ll_pdf_specifics(doc):
15131: (4)              r""" Low-level wrapper for `::pdf_specifics()`."""
15132: (4)              return _mupdf.ll_pdf_specifics(doc)
15133: (0)          def ll_pdf_sprint_obj(buf, cap, len, obj, tight, ascii):
15134: (4)              r"""
15135: (5)               Low-level wrapper for `::pdf_sprint_obj()`.
15136: (4)              This function has out-params. Python/C# wrappers look like:
15137: (8)                  `ll_pdf_sprint_obj(char *buf, size_t cap, ::pdf_obj *obj, int tight,
int ascii)` => `(char *, size_t len)`
15138: (4)              """
15139: (4)              return _mupdf.ll_pdf_sprint_obj(buf, cap, len, obj, tight, ascii)
15140: (0)          def ll_pdf_store_item(key, val, itemsize):
15141: (4)              r""" Low-level wrapper for `::pdf_store_item()`."""
15142: (4)              return _mupdf.ll_pdf_store_item(key, val, itemsize)
15143: (0)          def ll_pdf_string_from_annot_type(type):
15144: (4)              r""" Low-level wrapper for `::pdf_string_from_annot_type()`."""
15145: (4)              return _mupdf.ll_pdf_string_from_annot_type(type)
15146: (0)          def ll_pdf_string_from_intent(intent):
15147: (4)              r""" Low-level wrapper for `::pdf_string_from_intent()`."""
15148: (4)              return _mupdf.ll_pdf_string_from_intent(intent)
15149: (0)          def ll_pdf_string_from_line_ending(end):
15150: (4)              r""" Low-level wrapper for `::pdf_string_from_line_ending()`."""
15151: (4)              return _mupdf.ll_pdf_string_from_line_ending(end)
15152: (0)          def ll_pdf_structure_type(role_map, tag):
15153: (4)              r""" Low-level wrapper for `::pdf_structure_type()`."""
15154: (4)              return _mupdf.ll_pdf_structure_type(role_map, tag)
15155: (0)          def ll_pdf_subset_fonts(doc, pages_len, pages):
15156: (4)              r""" Low-level wrapper for `::pdf_subset_fonts()`."""
15157: (4)              return _mupdf.ll_pdf_subset_fonts(doc, pages_len, pages)
15158: (0)          def ll_pdf_subset_fonts2(doc, pages):
```

```
15159: (4)              r""" Low-level wrapper for `::pdf_subset_fonts2()`.  Swig-friendly wrapper
for pdf_subset_fonts()."""
15160: (4)              return _mupdf.ll_pdf_subset_fonts2(doc, pages)
15161: (0)          def ll_pdf_text_widget_format(tw):
15162: (4)              r""" Low-level wrapper for `::pdf_text_widget_format()`."""
15163: (4)              return _mupdf.ll_pdf_text_widget_format(tw)
15164: (0)          def ll_pdf_text_widget_max_len(tw):
15165: (4)              r""" Low-level wrapper for `::pdf_text_widget_max_len()`."""
15166: (4)              return _mupdf.ll_pdf_text_widget_max_len(tw)
15167: (0)          def ll_pdf_to_bool(obj):
15168: (4)              r""" Low-level wrapper for `::pdf_to_bool()`."""
15169: (4)              return _mupdf.ll_pdf_to_bool(obj)
15170: (0)          def ll_pdf_to_bool_default(obj, _def):
15171: (4)              r""" Low-level wrapper for `::pdf_to_bool_default()`."""
15172: (4)              return _mupdf.ll_pdf_to_bool_default(obj, _def)
15173: (0)          def ll_pdf_to_date(time):
15174: (4)              r""" Low-level wrapper for `::pdf_to_date()`."""
15175: (4)              return _mupdf.ll_pdf_to_date(time)
15176: (0)          def ll_pdf_to_gen(obj):
15177: (4)              r""" Low-level wrapper for `::pdf_to_gen()`."""
15178: (4)              return _mupdf.ll_pdf_to_gen(obj)
15179: (0)          def ll_pdf_to_int(obj):
15180: (4)              r""" Low-level wrapper for `::pdf_to_int()`."""
15181: (4)              return _mupdf.ll_pdf_to_int(obj)
15182: (0)          def ll_pdf_to_int64(obj):
15183: (4)              r""" Low-level wrapper for `::pdf_to_int64()`."""
15184: (4)              return _mupdf.ll_pdf_to_int64(obj)
15185: (0)          def ll_pdf_to_int_default(obj, _def):
15186: (4)              r""" Low-level wrapper for `::pdf_to_int_default()`."""
15187: (4)              return _mupdf.ll_pdf_to_int_default(obj, _def)
15188: (0)          def ll_pdf_to_matrix(array):
15189: (4)              r""" Low-level wrapper for `::pdf_to_matrix()`."""
15190: (4)              return _mupdf.ll_pdf_to_matrix(array)
15191: (0)          def ll_pdf_to_name(obj):
15192: (4)              r""" Low-level wrapper for `::pdf_to_name()`."""
15193: (4)              return _mupdf.ll_pdf_to_name(obj)
15194: (0)          def ll_pdf_to_num(obj):
15195: (4)              r""" Low-level wrapper for `::pdf_to_num()`."""
15196: (4)              return _mupdf.ll_pdf_to_num(obj)
15197: (0)          def ll_pdf_to_quad(array, offset):
15198: (4)              r""" Low-level wrapper for `::pdf_to_quad()`."""
15199: (4)              return _mupdf.ll_pdf_to_quad(array, offset)
15200: (0)          def ll_pdf_to_real(obj):
15201: (4)              r""" Low-level wrapper for `::pdf_to_real()`."""
15202: (4)              return _mupdf.ll_pdf_to_real(obj)
15203: (0)          def ll_pdf_to_real_default(obj, _def):
15204: (4)              r""" Low-level wrapper for `::pdf_to_real_default()`."""
15205: (4)              return _mupdf.ll_pdf_to_real_default(obj, _def)
15206: (0)          def ll_pdf_to_rect(array):
15207: (4)              r""" Low-level wrapper for `::pdf_to_rect()`."""
15208: (4)              return _mupdf.ll_pdf_to_rect(array)
15209: (0)          def ll_pdf_to_str_buf(obj):
15210: (4)              r""" Low-level wrapper for `::pdf_to_str_buf()`."""
15211: (4)              return _mupdf.ll_pdf_to_str_buf(obj)
15212: (0)          def ll_pdf_to_str_len(obj):
15213: (4)              r""" Low-level wrapper for `::pdf_to_str_len()`."""
15214: (4)              return _mupdf.ll_pdf_to_str_len(obj)
15215: (0)          def ll_pdf_to_string(obj, sizep):
15216: (4)              r"""
15217: (5)               Low-level wrapper for `::pdf_to_string()`.
15218: (4)              This function has out-params. Python/C# wrappers look like:
15219: (8)                  `ll_pdf_to_string(::pdf_obj *obj)` => `(const char *, size_t sizep)`
15220: (4)              """
15221: (4)              return _mupdf.ll_pdf_to_string(obj, sizep)
15222: (0)          def ll_pdf_to_text_string(obj):
15223: (4)              r""" Low-level wrapper for `::pdf_to_text_string()`."""
15224: (4)              return _mupdf.ll_pdf_to_text_string(obj)
15225: (0)          def ll_pdf_toggle_layer_config_ui(doc, ui):
15226: (4)              r""" Low-level wrapper for `::pdf_toggle_layer_config_ui()`."""
```

```
15227: (4)             return _mupdf.ll_pdf_toggle_layer_config_ui(doc, ui)
15228: (0)         def ll_pdf_toggle_widget(widget):
15229: (4)             r""" Low-level wrapper for `::pdf_toggle_widget()`."""
15230: (4)             return _mupdf.ll_pdf_toggle_widget(widget)
15231: (0)         def ll_pdf_tos_get_text(tos):
15232: (4)             r""" Low-level wrapper for `::pdf_tos_get_text()`."""
15233: (4)             return _mupdf.ll_pdf_tos_get_text(tos)
15234: (0)         def ll_pdf_tos_make_trm(tos, text, fontdesc, cid, trm):
15235: (4)             r""" Low-level wrapper for `::pdf_tos_make_trm()`."""
15236: (4)             return _mupdf.ll_pdf_tos_make_trm(tos, text, fontdesc, cid, trm)
15237: (0)         def ll_pdf_tos_move_after_char(tos):
15238: (4)             r""" Low-level wrapper for `::pdf_tos_move_after_char()`."""
15239: (4)             return _mupdf.ll_pdf_tos_move_after_char(tos)
15240: (0)         def ll_pdf_tos_newline(tos, leading):
15241: (4)             r""" Low-level wrapper for `::pdf_tos_newline()`."""
15242: (4)             return _mupdf.ll_pdf_tos_newline(tos, leading)
15243: (0)         def ll_pdf_tos_reset(tos, render):
15244: (4)             r""" Low-level wrapper for `::pdf_tos_reset()`."""
15245: (4)             return _mupdf.ll_pdf_tos_reset(tos, render)
15246: (0)         def ll_pdf_tos_restore(tos, save):
15247: (4)             r""" Low-level wrapper for `::pdf_tos_restore()`."""
15248: (4)             return _mupdf.ll_pdf_tos_restore(tos, save)
15249: (0)         def ll_pdf_tos_save(tos, save):
15250: (4)             r""" Low-level wrapper for `::pdf_tos_save()`."""
15251: (4)             return _mupdf.ll_pdf_tos_save(tos, save)
15252: (0)         def ll_pdf_tos_set_matrix(tos, a, b, c, d, e, f):
15253: (4)             r""" Low-level wrapper for `::pdf_tos_set_matrix()`."""
15254: (4)             return _mupdf.ll_pdf_tos_set_matrix(tos, a, b, c, d, e, f)
15255: (0)         def ll_pdf_tos_translate(tos, tx, ty):
15256: (4)             r""" Low-level wrapper for `::pdf_tos_translate()`."""
15257: (4)             return _mupdf.ll_pdf_tos_translate(tos, tx, ty)
15258: (0)         def ll_pdf_trailer(doc):
15259: (4)             r""" Low-level wrapper for `::pdf_trailer()`."""
15260: (4)             return _mupdf.ll_pdf_trailer(doc)
15261: (0)         def ll_pdf_undo(doc):
15262: (4)             r""" Low-level wrapper for `::pdf_undo()`."""
15263: (4)             return _mupdf.ll_pdf_undo(doc)
15264: (0)         def ll_pdf_undoredo_state(doc, steps):
15265: (4)             r"""
15266: (5)              Low-level wrapper for `::pdf_undoredo_state()`.
15267: (4)             This function has out-params. Python/C# wrappers look like:
15268: (8)                 `ll_pdf_undoredo_state(::pdf_document *doc)` => `(int, int steps)`
15269: (4)             """
15270: (4)             return _mupdf.ll_pdf_undoredo_state(doc, steps)
15271: (0)         def ll_pdf_undoredo_step(doc, step):
15272: (4)             r""" Low-level wrapper for `::pdf_undoredo_step()`."""
15273: (4)             return _mupdf.ll_pdf_undoredo_step(doc, step)
15274: (0)         def ll_pdf_unmark_obj(obj):
15275: (4)             r""" Low-level wrapper for `::pdf_unmark_obj()`."""
15276: (4)             return _mupdf.ll_pdf_unmark_obj(obj)
15277: (0)         def ll_pdf_update_annot(annot):
15278: (4)             r""" Low-level wrapper for `::pdf_update_annot()`."""
15279: (4)             return _mupdf.ll_pdf_update_annot(annot)
15280: (0)         def ll_pdf_update_default_colorspaces(old_cs, res):
15281: (4)             r""" Low-level wrapper for `::pdf_update_default_colorspaces()`."""
15282: (4)             return _mupdf.ll_pdf_update_default_colorspaces(old_cs, res)
15283: (0)         def ll_pdf_update_object(doc, num, obj):
15284: (4)             r""" Low-level wrapper for `::pdf_update_object()`."""
15285: (4)             return _mupdf.ll_pdf_update_object(doc, num, obj)
15286: (0)         def ll_pdf_update_page(page):
15287: (4)             r""" Low-level wrapper for `::pdf_update_page()`."""
15288: (4)             return _mupdf.ll_pdf_update_page(page)
15289: (0)         def ll_pdf_update_stream(doc, ref, buf, compressed):
15290: (4)             r""" Low-level wrapper for `::pdf_update_stream()`."""
15291: (4)             return _mupdf.ll_pdf_update_stream(doc, ref, buf, compressed)
15292: (0)         def ll_pdf_update_widget(widget):
15293: (4)             r""" Low-level wrapper for `::pdf_update_widget()`."""
15294: (4)             return _mupdf.ll_pdf_update_widget(widget)
15295: (0)         def ll_pdf_update_xobject(doc, xobj, bbox, mat, res, buffer):
```

```
15296: (4)              r""" Low-level wrapper for `::pdf_update_xobject()`."""
15297: (4)              return _mupdf.ll_pdf_update_xobject(doc, xobj, bbox, mat, res, buffer)
15298: (0)          def ll_pdf_validate_change_history(doc):
15299: (4)              r""" Low-level wrapper for `::pdf_validate_change_history()`."""
15300: (4)              return _mupdf.ll_pdf_validate_change_history(doc)
15301: (0)          def ll_pdf_validate_changes(doc, version):
15302: (4)              r""" Low-level wrapper for `::pdf_validate_changes()`."""
15303: (4)              return _mupdf.ll_pdf_validate_changes(doc, version)
15304: (0)          def ll_pdf_validate_signature(widget):
15305: (4)              r""" Low-level wrapper for `::pdf_validate_signature()`."""
15306: (4)              return _mupdf.ll_pdf_validate_signature(widget)
15307: (0)          def ll_pdf_verify_embedded_file_checksum(fs):
15308: (4)              r""" Low-level wrapper for `::pdf_verify_embedded_file_checksum()`."""
15309: (4)              return _mupdf.ll_pdf_verify_embedded_file_checksum(fs)
15310: (0)          def ll_pdf_version(doc):
15311: (4)              r""" Low-level wrapper for `::pdf_version()`."""
15312: (4)              return _mupdf.ll_pdf_version(doc)
15313: (0)          def ll_pdf_walk_tree(tree, kid_name, arrive, leave, arg, names, values):
15314: (4)              r"""
15315: (5)               Low-level wrapper for `::pdf_walk_tree()`.
15316: (4)              This function has out-params. Python/C# wrappers look like:
15317: (8)                  `ll_pdf_walk_tree(::pdf_obj *tree, ::pdf_obj *kid_name, void (*arrive)
(::fz_context *, ::pdf_obj *, void *, ::pdf_obj **), void (*leave)(::fz_context *, ::pdf_obj *,
void *), void *arg, ::pdf_obj **names, ::pdf_obj **values)` => `()`
15318: (4)              """
15319: (4)              return _mupdf.ll_pdf_walk_tree(tree, kid_name, arrive, leave, arg, names,
values)
15320: (0)          def ll_pdf_was_pure_xfa(doc):
15321: (4)              r""" Low-level wrapper for `::pdf_was_pure_xfa()`."""
15322: (4)              return _mupdf.ll_pdf_was_pure_xfa(doc)
15323: (0)          def ll_pdf_was_repaired(doc):
15324: (4)              r""" Low-level wrapper for `::pdf_was_repaired()`."""
15325: (4)              return _mupdf.ll_pdf_was_repaired(doc)
15326: (0)          def ll_pdf_widget_is_readonly(widget):
15327: (4)              r""" Low-level wrapper for `::pdf_widget_is_readonly()`."""
15328: (4)              return _mupdf.ll_pdf_widget_is_readonly(widget)
15329: (0)          def ll_pdf_widget_is_signed(widget):
15330: (4)              r""" Low-level wrapper for `::pdf_widget_is_signed()`."""
15331: (4)              return _mupdf.ll_pdf_widget_is_signed(widget)
15332: (0)          def ll_pdf_widget_type(widget):
15333: (4)              r""" Low-level wrapper for `::pdf_widget_type()`."""
15334: (4)              return _mupdf.ll_pdf_widget_type(widget)
15335: (0)          def ll_pdf_write_digest(out, byte_range, field, digest_offset, digest_length,
signer):
15336: (4)              r""" Low-level wrapper for `::pdf_write_digest()`."""
15337: (4)              return _mupdf.ll_pdf_write_digest(out, byte_range, field, digest_offset,
digest_length, signer)
15338: (0)          def ll_pdf_write_document(doc, out, opts):
15339: (4)              r""" Low-level wrapper for `::pdf_write_document()`."""
15340: (4)              return _mupdf.ll_pdf_write_document(doc, out, opts)
15341: (0)          def ll_pdf_write_journal(doc, out):
15342: (4)              r""" Low-level wrapper for `::pdf_write_journal()`."""
15343: (4)              return _mupdf.ll_pdf_write_journal(doc, out)
15344: (0)          def ll_pdf_write_snapshot(doc, out):
15345: (4)              r""" Low-level wrapper for `::pdf_write_snapshot()`."""
15346: (4)              return _mupdf.ll_pdf_write_snapshot(doc, out)
15347: (0)          def ll_pdf_xobject_bbox(xobj):
15348: (4)              r""" Low-level wrapper for `::pdf_xobject_bbox()`."""
15349: (4)              return _mupdf.ll_pdf_xobject_bbox(xobj)
15350: (0)          def ll_pdf_xobject_colorspace(xobj):
15351: (4)              r""" Low-level wrapper for `::pdf_xobject_colorspace()`."""
15352: (4)              return _mupdf.ll_pdf_xobject_colorspace(xobj)
15353: (0)          def ll_pdf_xobject_isolated(xobj):
15354: (4)              r""" Low-level wrapper for `::pdf_xobject_isolated()`."""
15355: (4)              return _mupdf.ll_pdf_xobject_isolated(xobj)
15356: (0)          def ll_pdf_xobject_knockout(xobj):
15357: (4)              r""" Low-level wrapper for `::pdf_xobject_knockout()`."""
15358: (4)              return _mupdf.ll_pdf_xobject_knockout(xobj)
15359: (0)          def ll_pdf_xobject_matrix(xobj):
```

```
15360: (4)             r""" Low-level wrapper for `::pdf_xobject_matrix()`."""
15361: (4)             return _mupdf.ll_pdf_xobject_matrix(xobj)
15362: (0)         def ll_pdf_xobject_resources(xobj):
15363: (4)             r""" Low-level wrapper for `::pdf_xobject_resources()`."""
15364: (4)             return _mupdf.ll_pdf_xobject_resources(xobj)
15365: (0)         def ll_pdf_xobject_transparency(xobj):
15366: (4)             r""" Low-level wrapper for `::pdf_xobject_transparency()`."""
15367: (4)             return _mupdf.ll_pdf_xobject_transparency(xobj)
15368: (0)         def ll_pdf_xref_ensure_incremental_object(doc, num):
15369: (4)             r""" Low-level wrapper for `::pdf_xref_ensure_incremental_object()`."""
15370: (4)             return _mupdf.ll_pdf_xref_ensure_incremental_object(doc, num)
15371: (0)         def ll_pdf_xref_ensure_local_object(doc, num):
15372: (4)             r""" Low-level wrapper for `::pdf_xref_ensure_local_object()`."""
15373: (4)             return _mupdf.ll_pdf_xref_ensure_local_object(doc, num)
15374: (0)         def ll_pdf_xref_entry_map(doc, fn, arg):
15375: (4)             r""" Low-level wrapper for `::pdf_xref_entry_map()`."""
15376: (4)             return _mupdf.ll_pdf_xref_entry_map(doc, fn, arg)
15377: (0)         def ll_pdf_xref_is_incremental(doc, num):
15378: (4)             r""" Low-level wrapper for `::pdf_xref_is_incremental()`."""
15379: (4)             return _mupdf.ll_pdf_xref_is_incremental(doc, num)
15380: (0)         def ll_pdf_xref_len(doc):
15381: (4)             r""" Low-level wrapper for `::pdf_xref_len()`."""
15382: (4)             return _mupdf.ll_pdf_xref_len(doc)
15383: (0)         def ll_pdf_xref_obj_is_unsaved_signature(doc, obj):
15384: (4)             r""" Low-level wrapper for `::pdf_xref_obj_is_unsaved_signature()`."""
15385: (4)             return _mupdf.ll_pdf_xref_obj_is_unsaved_signature(doc, obj)
15386: (0)         def ll_pdf_xref_remove_unsaved_signature(doc, field):
15387: (4)             r""" Low-level wrapper for `::pdf_xref_remove_unsaved_signature()`."""
15388: (4)             return _mupdf.ll_pdf_xref_remove_unsaved_signature(doc, field)
15389: (0)         def ll_pdf_xref_store_unsaved_signature(doc, field, signer):
15390: (4)             r""" Low-level wrapper for `::pdf_xref_store_unsaved_signature()`."""
15391: (4)             return _mupdf.ll_pdf_xref_store_unsaved_signature(doc, field, signer)
15392: (0)         def to_string_fz_aa_context(s):
15393: (4)             r""" Returns string containing a fz_aa_context's members, labelled and
inside (...), using operator<<."""
15394: (4)             return _mupdf.to_string_fz_aa_context(s)
15395: (0)         def to_string_fz_color_params(s):
15396: (4)             r""" Returns string containing a fz_color_params's members, labelled and
inside (...), using operator<<."""
15397: (4)             return _mupdf.to_string_fz_color_params(s)
15398: (0)         def to_string_fz_cookie(s):
15399: (4)             r""" Returns string containing a fz_cookie's members, labelled and inside
(...), using operator<<."""
15400: (4)             return _mupdf.to_string_fz_cookie(s)
15401: (0)         def to_string_fz_draw_options(s):
15402: (4)             r""" Returns string containing a fz_draw_options's members, labelled and
inside (...), using operator<<."""
15403: (4)             return _mupdf.to_string_fz_draw_options(s)
15404: (0)         def to_string_fz_install_load_system_font_funcs_args(s):
15405: (4)             r""" Returns string containing a fz_install_load_system_font_funcs_args's
members, labelled and inside (...), using operator<<."""
15406: (4)             return _mupdf.to_string_fz_install_load_system_font_funcs_args(s)
15407: (0)         def to_string_fz_irect(s):
15408: (4)             r""" Returns string containing a fz_irect's members, labelled and inside
(...), using operator<<."""
15409: (4)             return _mupdf.to_string_fz_irect(s)
15410: (0)         def to_string_fz_location(s):
15411: (4)             r""" Returns string containing a fz_location's members, labelled and
inside (...), using operator<<."""
15412: (4)             return _mupdf.to_string_fz_location(s)
15413: (0)         def to_string_fz_matrix(s):
15414: (4)             r""" Returns string containing a fz_matrix's members, labelled and inside
(...), using operator<<."""
15415: (4)             return _mupdf.to_string_fz_matrix(s)
15416: (0)         def to_string_fz_md5(s):
15417: (4)             r""" Returns string containing a fz_md5's members, labelled and inside
(...), using operator<<."""
15418: (4)             return _mupdf.to_string_fz_md5(s)
15419: (0)         def to_string_fz_outline_item(s):
```

```
15420: (4)              r""" Returns string containing a fz_outline_item's members, labelled and
inside (...), using operator<<."""
15421: (4)              return _mupdf.to_string_fz_outline_item(s)
15422: (0)          def to_string_fz_pdfocr_options(s):
15423: (4)              r""" Returns string containing a fz_pdfocr_options's members, labelled and
inside (...), using operator<<."""
15424: (4)              return _mupdf.to_string_fz_pdfocr_options(s)
15425: (0)          def to_string_fz_point(s):
15426: (4)              r""" Returns string containing a fz_point's members, labelled and inside
(...), using operator<<."""
15427: (4)              return _mupdf.to_string_fz_point(s)
15428: (0)          def to_string_fz_pwg_options(s):
15429: (4)              r""" Returns string containing a fz_pwg_options's members, labelled and
inside (...), using operator<<."""
15430: (4)              return _mupdf.to_string_fz_pwg_options(s)
15431: (0)          def to_string_fz_quad(s):
15432: (4)              r""" Returns string containing a fz_quad's members, labelled and inside
(...), using operator<<."""
15433: (4)              return _mupdf.to_string_fz_quad(s)
15434: (0)          def to_string_fz_rect(s):
15435: (4)              r""" Returns string containing a fz_rect's members, labelled and inside
(...), using operator<<."""
15436: (4)              return _mupdf.to_string_fz_rect(s)
15437: (0)          def to_string_fz_stext_options(s):
15438: (4)              r""" Returns string containing a fz_stext_options's members, labelled and
inside (...), using operator<<."""
15439: (4)              return _mupdf.to_string_fz_stext_options(s)
15440: (0)          def to_string_fz_story_element_position(s):
15441: (4)              r""" Returns string containing a fz_story_element_position's members,
labelled and inside (...), using operator<<."""
15442: (4)              return _mupdf.to_string_fz_story_element_position(s)
15443: (0)          def to_string_fz_transition(s):
15444: (4)              r""" Returns string containing a fz_transition's members, labelled and
inside (...), using operator<<."""
15445: (4)              return _mupdf.to_string_fz_transition(s)
15446: (0)          def to_string_pdf_clean_options(s):
15447: (4)              r""" Returns string containing a pdf_clean_options's members, labelled and
inside (...), using operator<<."""
15448: (4)              return _mupdf.to_string_pdf_clean_options(s)
15449: (0)          def to_string_pdf_filter_factory(s):
15450: (4)              r""" Returns string containing a pdf_filter_factory's members, labelled
and inside (...), using operator<<."""
15451: (4)              return _mupdf.to_string_pdf_filter_factory(s)
15452: (0)          def to_string_pdf_filter_options(s):
15453: (4)              r""" Returns string containing a pdf_filter_options's members, labelled
and inside (...), using operator<<."""
15454: (4)              return _mupdf.to_string_pdf_filter_options(s)
15455: (0)          def to_string_pdf_image_rewriter_options(s):
15456: (4)              r""" Returns string containing a pdf_image_rewriter_options's members,
labelled and inside (...), using operator<<."""
15457: (4)              return _mupdf.to_string_pdf_image_rewriter_options(s)
15458: (0)          def to_string_pdf_layer_config(s):
15459: (4)              r""" Returns string containing a pdf_layer_config's members, labelled and
inside (...), using operator<<."""
15460: (4)              return _mupdf.to_string_pdf_layer_config(s)
15461: (0)          def to_string_pdf_layer_config_ui(s):
15462: (4)              r""" Returns string containing a pdf_layer_config_ui's members, labelled
and inside (...), using operator<<."""
15463: (4)              return _mupdf.to_string_pdf_layer_config_ui(s)
15464: (0)          def to_string_pdf_redact_options(s):
15465: (4)              r""" Returns string containing a pdf_redact_options's members, labelled
and inside (...), using operator<<."""
15466: (4)              return _mupdf.to_string_pdf_redact_options(s)
15467: (0)          def to_string_pdf_sanitize_filter_options(s):
15468: (4)              r""" Returns string containing a pdf_sanitize_filter_options's members,
labelled and inside (...), using operator<<."""
15469: (4)              return _mupdf.to_string_pdf_sanitize_filter_options(s)
15470: (0)          def to_string_pdf_write_options(s):
15471: (4)              r""" Returns string containing a pdf_write_options's members, labelled and
```

```
inside (...), using operator<<."""
15472: (4)              return _mupdf.to_string_pdf_write_options(s)
15473: (0)          def to_string(*args):
15474: (4)              r"""
15475: (4)              *Overload 1:*
15476: (5)               Returns string containing a fz_aa_context's members, labelled and inside
(...), using operator<<.
15477: (4)              (Convenience overload).
15478: (4)              |
15479: (4)              *Overload 2:*
15480: (5)               Returns string containing a fz_color_params's members, labelled and
inside (...), using operator<<.
15481: (4)              (Convenience overload).
15482: (4)              |
15483: (4)              *Overload 3:*
15484: (5)               Returns string containing a fz_cookie's members, labelled and inside
(...), using operator<<.
15485: (4)              (Convenience overload).
15486: (4)              |
15487: (4)              *Overload 4:*
15488: (5)               Returns string containing a fz_draw_options's members, labelled and
inside (...), using operator<<.
15489: (4)              (Convenience overload).
15490: (4)              |
15491: (4)              *Overload 5:*
15492: (5)               Returns string containing a fz_install_load_system_font_funcs_args's
members, labelled and inside (...), using operator<<.
15493: (4)              (Convenience overload).
15494: (4)              |
15495: (4)              *Overload 6:*
15496: (5)               Returns string containing a fz_irect's members, labelled and inside
(...), using operator<<.
15497: (4)              (Convenience overload).
15498: (4)              |
15499: (4)              *Overload 7:*
15500: (5)               Returns string containing a fz_location's members, labelled and inside
(...), using operator<<.
15501: (4)              (Convenience overload).
15502: (4)              |
15503: (4)              *Overload 8:*
15504: (5)               Returns string containing a fz_matrix's members, labelled and inside
(...), using operator<<.
15505: (4)              (Convenience overload).
15506: (4)              |
15507: (4)              *Overload 9:*
15508: (5)               Returns string containing a fz_md5's members, labelled and inside (...),
using operator<<.
15509: (4)              (Convenience overload).
15510: (4)              |
15511: (4)              *Overload 10:*
15512: (5)               Returns string containing a fz_outline_item's members, labelled and
inside (...), using operator<<.
15513: (4)              (Convenience overload).
15514: (4)              |
15515: (4)              *Overload 11:*
15516: (5)               Returns string containing a fz_pdfocr_options's members, labelled and
inside (...), using operator<<.
15517: (4)              (Convenience overload).
15518: (4)              |
15519: (4)              *Overload 12:*
15520: (5)               Returns string containing a fz_point's members, labelled and inside
(...), using operator<<.
15521: (4)              (Convenience overload).
15522: (4)              |
15523: (4)              *Overload 13:*
15524: (5)               Returns string containing a fz_pwg_options's members, labelled and inside
(...), using operator<<.
15525: (4)              (Convenience overload).
15526: (4)              |
```

```
15527: (4)              *Overload 14:*
15528: (5)               Returns string containing a fz_quad's members, labelled and inside (...),
using operator<<.
15529: (4)              (Convenience overload).
15530: (4)              |
15531: (4)              *Overload 15:*
15532: (5)               Returns string containing a fz_rect's members, labelled and inside (...),
using operator<<.
15533: (4)              (Convenience overload).
15534: (4)              |
15535: (4)              *Overload 16:*
15536: (5)               Returns string containing a fz_stext_options's members, labelled and
inside (...), using operator<<.
15537: (4)              (Convenience overload).
15538: (4)              |
15539: (4)              *Overload 17:*
15540: (5)               Returns string containing a fz_story_element_position's members, labelled
and inside (...), using operator<<.
15541: (4)              (Convenience overload).
15542: (4)              |
15543: (4)              *Overload 18:*
15544: (5)               Returns string containing a fz_transition's members, labelled and inside
(...), using operator<<.
15545: (4)              (Convenience overload).
15546: (4)              |
15547: (4)              *Overload 19:*
15548: (5)               Returns string containing a pdf_clean_options's members, labelled and
inside (...), using operator<<.
15549: (4)              (Convenience overload).
15550: (4)              |
15551: (4)              *Overload 20:*
15552: (5)               Returns string containing a pdf_filter_factory's members, labelled and
inside (...), using operator<<.
15553: (4)              (Convenience overload).
15554: (4)              |
15555: (4)              *Overload 21:*
15556: (5)               Returns string containing a pdf_filter_options's members, labelled and
inside (...), using operator<<.
15557: (4)              (Convenience overload).
15558: (4)              |
15559: (4)              *Overload 22:*
15560: (5)               Returns string containing a pdf_image_rewriter_options's members,
labelled and inside (...), using operator<<.
15561: (4)              (Convenience overload).
15562: (4)              |
15563: (4)              *Overload 23:*
15564: (5)               Returns string containing a pdf_layer_config's members, labelled and
inside (...), using operator<<.
15565: (4)              (Convenience overload).
15566: (4)              |
15567: (4)              *Overload 24:*
15568: (5)               Returns string containing a pdf_layer_config_ui's members, labelled and
inside (...), using operator<<.
15569: (4)              (Convenience overload).
15570: (4)              |
15571: (4)              *Overload 25:*
15572: (5)               Returns string containing a pdf_redact_options's members, labelled and
inside (...), using operator<<.
15573: (4)              (Convenience overload).
15574: (4)              |
15575: (4)              *Overload 26:*
15576: (5)               Returns string containing a pdf_sanitize_filter_options's members,
labelled and inside (...), using operator<<.
15577: (4)              (Convenience overload).
15578: (4)              |
15579: (4)              *Overload 27:*
15580: (5)               Returns string containing a pdf_write_options's members, labelled and
inside (...), using operator<<.
15581: (4)              (Convenience overload).
```

```
15582: (4)              """
15583: (4)              return _mupdf.to_string(*args)
15584: (0)          def reinit_singlethreaded():
15585: (4)              r"""
15586: (5)               Reinitializes the MuPDF context for single-threaded use, which
15587: (4)              is slightly faster when calling code is single threaded.
15588: (4)              This should be called before any other use of MuPDF.
15589: (4)              """
15590: (4)              return _mupdf.reinit_singlethreaded()
15591: (0)          class FzAaContext(object):
15592: (4)              r""" Wrapper class for struct `fz_aa_context`."""
15593: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15594: (4)              __repr__ = _swig_repr
15595: (4)              def __init__(self, *args):
15596: (8)                  r"""
15597: (8)                  *Overload 1:*
15598: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
15599: (8)                  |
15600: (8)                  *Overload 2:*
15601: (8)                  Constructor using raw copy of pre-existing `::fz_aa_context`.
15602: (8)                  |
15603: (8)                  *Overload 3:*
15604: (8)                  Constructor using raw copy of pre-existing `::fz_aa_context`.
15605: (8)                  """
15606: (8)                  _mupdf.FzAaContext_swiginit(self, _mupdf.new_FzAaContext(*args))
15607: (4)              def internal(self, *args):
15608: (8)                  r"""
15609: (8)                  *Overload 1:*
15610: (8)                  Access as underlying struct.
15611: (8)                  |
15612: (8)                  *Overload 2:*
15613: (8)                  Access as underlying struct.
15614: (8)                  """
15615: (8)                  return _mupdf.FzAaContext_internal(self, *args)
15616: (4)              __swig_destroy__ = _mupdf.delete_FzAaContext
15617: (4)              hscale = property(_mupdf.FzAaContext_hscale_get,
_mupdf.FzAaContext_hscale_set)
15618: (4)              vscale = property(_mupdf.FzAaContext_vscale_get,
_mupdf.FzAaContext_vscale_set)
15619: (4)              scale = property(_mupdf.FzAaContext_scale_get,
_mupdf.FzAaContext_scale_set)
15620: (4)              bits = property(_mupdf.FzAaContext_bits_get, _mupdf.FzAaContext_bits_set)
15621: (4)              text_bits = property(_mupdf.FzAaContext_text_bits_get,
_mupdf.FzAaContext_text_bits_set)
15622: (4)              min_line_width = property(_mupdf.FzAaContext_min_line_width_get,
_mupdf.FzAaContext_min_line_width_set)
15623: (4)              s_num_instances = property(_mupdf.FzAaContext_s_num_instances_get,
_mupdf.FzAaContext_s_num_instances_set)
15624: (4)              def to_string(self):
15625: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
15626: (8)                  return _mupdf.FzAaContext_to_string(self)
15627: (4)              def __eq__(self, rhs):
15628: (8)                  r""" Comparison method."""
15629: (8)                  return _mupdf.FzAaContext___eq__(self, rhs)
15630: (4)              def __ne__(self, rhs):
15631: (8)                  r""" Comparison method."""
15632: (8)                  return _mupdf.FzAaContext___ne__(self, rhs)
15633: (0)          _mupdf.FzAaContext_swigregister(FzAaContext)
15634: (0)          metadata_keys = cvar.metadata_keys
15635: (0)          class FzAes(object):
15636: (4)              r"""
15637: (4)              Wrapper class for struct `fz_aes`. Not copyable or assignable.
15638: (4)              Structure definitions are public to enable stack
15639: (4)              based allocation. Do not access the members directly.
15640: (4)              """
15641: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
15642: (4)              __repr__ = _swig_repr
15643: (4)              def fz_aes_crypt_cbc(self, mode, length, iv, input, output):
15644: (8)                  r"""
15645: (8)                  Class-aware wrapper for `::fz_aes_crypt_cbc()`.
15646: (12)                     AES block processing. Encrypts or Decrypts (according to mode,
15647: (12)                     which must match what was initially set up) length bytes (which
15648: (12)                     must be a multiple of 16), using (and modifying) the insertion
15649: (12)                     vector iv, reading from input, and writing to output.
15650: (12)                     Never throws an exception.
15651: (8)                  """
15652: (8)                  return _mupdf.FzAes_fz_aes_crypt_cbc(self, mode, length, iv, input,
output)
15653: (4)              def fz_aes_setkey_dec(self, key, keysize):
15654: (8)                  r"""
15655: (8)                  Class-aware wrapper for `::fz_aes_setkey_dec()`.
15656: (12)                     AES decryption intialisation. Fills in the supplied context
15657: (12)                     and prepares for decryption using the given key.
15658: (12)                     Returns non-zero for error (key size other than 128/192/256).
15659: (12)                     Never throws an exception.
15660: (8)                  """
15661: (8)                  return _mupdf.FzAes_fz_aes_setkey_dec(self, key, keysize)
15662: (4)              def fz_aes_setkey_enc(self, key, keysize):
15663: (8)                  r"""
15664: (8)                  Class-aware wrapper for `::fz_aes_setkey_enc()`.
15665: (12)                     AES encryption intialisation. Fills in the supplied context
15666: (12)                     and prepares for encryption using the given key.
15667: (12)                     Returns non-zero for error (key size other than 128/192/256).
15668: (12)                     Never throws an exception.
15669: (8)                  """
15670: (8)                  return _mupdf.FzAes_fz_aes_setkey_enc(self, key, keysize)
15671: (4)              def __init__(self, *args):
15672: (8)                  r"""
15673: (8)                  *Overload 1:*
15674: (8)                  Default constructor, sets `m_internal` to null.
15675: (8)                  |
15676: (8)                  *Overload 2:*
15677: (8)                  Constructor using raw copy of pre-existing `::fz_aes`.
15678: (8)                  """
15679: (8)                  _mupdf.FzAes_swiginit(self, _mupdf.new_FzAes(*args))
15680: (4)              __swig_destroy__ = _mupdf.delete_FzAes
15681: (4)              def m_internal_value(self):
15682: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15683: (8)                  return _mupdf.FzAes_m_internal_value(self)
15684: (4)              m_internal = property(_mupdf.FzAes_m_internal_get,
_mupdf.FzAes_m_internal_set, doc=r""" Pointer to wrapped data.""")
15685: (4)              s_num_instances = property(_mupdf.FzAes_s_num_instances_get,
_mupdf.FzAes_s_num_instances_set)
15686: (0)          _mupdf.FzAes_swigregister(FzAes)
15687: (0)          class FzAllocContext(object):
15688: (4)              r"""
15689: (4)              Wrapper class for struct `fz_alloc_context`. Not copyable or assignable.
15690: (4)              Allocator structure; holds callbacks and private data pointer.
15691: (4)              """
15692: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15693: (4)              __repr__ = _swig_repr
15694: (4)              def __init__(self, *args):
15695: (8)                  r"""
15696: (8)                  *Overload 1:*
15697: (8)                  Default constructor, sets `m_internal` to null.
15698: (8)                  |
15699: (8)                  *Overload 2:*
15700: (8)                  Constructor using raw copy of pre-existing `::fz_alloc_context`.
15701: (8)                  """
15702: (8)                  _mupdf.FzAllocContext_swiginit(self, _mupdf.new_FzAllocContext(*args))
15703: (4)              __swig_destroy__ = _mupdf.delete_FzAllocContext
15704: (4)              def m_internal_value(self):
```

```
15705: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15706: (8)                  return _mupdf.FzAllocContext_m_internal_value(self)
15707: (4)              m_internal = property(_mupdf.FzAllocContext_m_internal_get,
_mupdf.FzAllocContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
15708: (4)              s_num_instances = property(_mupdf.FzAllocContext_s_num_instances_get,
_mupdf.FzAllocContext_s_num_instances_set)
15709: (0)          _mupdf.FzAllocContext_swigregister(FzAllocContext)
15710: (0)          class FzArc4(object):
15711: (4)              r"""
15712: (4)              Wrapper class for struct `fz_arc4`. Not copyable or assignable.
15713: (4)              Structure definition is public to enable stack
15714: (4)              based allocation. Do not access the members directly.
15715: (4)              """
15716: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15717: (4)              __repr__ = _swig_repr
15718: (4)              def fz_arc4_encrypt(self, dest, src, len):
15719: (8)                  r"""
15720: (8)                  Class-aware wrapper for `::fz_arc4_encrypt()`.
15721: (12)                     RC4 block encrypt operation; encrypt src into dst (both of
15722: (12)                     length len) updating the RC4 state as we go.
15723: (12)                     Never throws an exception.
15724: (8)                  """
15725: (8)                  return _mupdf.FzArc4_fz_arc4_encrypt(self, dest, src, len)
15726: (4)              def fz_arc4_final(self):
15727: (8)                  r"""
15728: (8)                  Class-aware wrapper for `::fz_arc4_final()`.
15729: (12)                     RC4 finalization. Zero the context.
15730: (12)                     Never throws an exception.
15731: (8)                  """
15732: (8)                  return _mupdf.FzArc4_fz_arc4_final(self)
15733: (4)              def fz_arc4_init(self, key, len):
15734: (8)                  r"""
15735: (8)                  Class-aware wrapper for `::fz_arc4_init()`.
15736: (12)                     RC4 initialization. Begins an RC4 operation, writing a new
15737: (12)                     context.
15738: (12)                     Never throws an exception.
15739: (8)                  """
15740: (8)                  return _mupdf.FzArc4_fz_arc4_init(self, key, len)
15741: (4)              def __init__(self, *args):
15742: (8)                  r"""
15743: (8)                  *Overload 1:*
15744: (8)                  Default constructor, sets `m_internal` to null.
15745: (8)                  |
15746: (8)                  *Overload 2:*
15747: (8)                  Constructor using raw copy of pre-existing `::fz_arc4`.
15748: (8)                  """
15749: (8)                  _mupdf.FzArc4_swiginit(self, _mupdf.new_FzArc4(*args))
15750: (4)              __swig_destroy__ = _mupdf.delete_FzArc4
15751: (4)              def m_internal_value(self):
15752: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15753: (8)                  return _mupdf.FzArc4_m_internal_value(self)
15754: (4)              m_internal = property(_mupdf.FzArc4_m_internal_get,
_mupdf.FzArc4_m_internal_set, doc=r""" Pointer to wrapped data.""")
15755: (4)              s_num_instances = property(_mupdf.FzArc4_s_num_instances_get,
_mupdf.FzArc4_s_num_instances_set)
15756: (0)          _mupdf.FzArc4_swigregister(FzArc4)
15757: (0)          class FzArchive(object):
15758: (4)              r"""
15759: (4)              Wrapper class for struct `fz_archive`.
15760: (4)              fz_archive:
15761: (4)              fz_archive provides methods for accessing "archive" files.
15762: (4)              An archive file is a conceptual entity that contains multiple
15763: (4)              files, which can be counted, enumerated, and read.
15764: (4)              Implementations of fz_archive based upon directories, zip
15765: (4)              and tar files are included.
15766: (4)              """
```

```
(4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
(4)             __repr__ = _swig_repr
(4)             def fz_archive_format(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_archive_format()`.
(12)                    Return a pointer to a string describing the format of the
(12)                    archive.
(12)                    The lifetime of the string is unspecified (in current
(12)                    implementations the string will persist until the archive
(12)                    is closed, but this is not guaranteed).
(8)                 """
(8)                 return _mupdf.FzArchive_fz_archive_format(self)
(4)             def fz_count_archive_entries(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_count_archive_entries()`.
(12)                    Number of entries in archive.
(12)                    Will always return a value >= 0.
(12)                    May throw an exception if this type of archive cannot count the
(12)                    entries (such as a directory).
(8)                 """
(8)                 return _mupdf.FzArchive_fz_count_archive_entries(self)
(4)             def fz_has_archive_entry(self, name):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_has_archive_entry()`.
(12)                    Check if entry by given name exists.
(12)                    If named entry does not exist 0 will be returned, if it does
(12)                    exist 1 is returned.
(12)                    name: Entry name to look for, this must be an exact match to
(12)                    the entry name in the archive.
(8)                 """
(8)                 return _mupdf.FzArchive_fz_has_archive_entry(self, name)
(4)             def fz_list_archive_entry(self, idx):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_list_archive_entry()`.
(12)                    Get listed name of entry position idx.
(12)                    idx: Must be a value >= 0 < return value from
(12)                    fz_count_archive_entries. If not in range NULL will be
(12)                    returned.
(12)                    May throw an exception if this type of archive cannot list the
(12)                    entries (such as a directory).
(8)                 """
(8)                 return _mupdf.FzArchive_fz_list_archive_entry(self, idx)
(4)             def fz_mount_multi_archive(self, sub, path):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_mount_multi_archive()`.
(12)                    Add an archive to the set of archives handled by a multi
(12)                    archive.
(12)                    If path is NULL, then the archive contents will appear at the
(12)                    top level, otherwise, the archives contents will appear prefixed
(12)                    by path.
(8)                 """
(8)                 return _mupdf.FzArchive_fz_mount_multi_archive(self, sub, path)
(4)             def fz_open_archive_entry(self, name):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_archive_entry()`.
(12)                    Opens an archive entry as a stream.
(12)                    name: Entry name to look for, this must be an exact match to
(12)                    the entry name in the archive.
(12)                    Throws an exception if a matching entry cannot be found.
(8)                 """
(8)                 return _mupdf.FzArchive_fz_open_archive_entry(self, name)
(4)             def fz_parse_xml_archive_entry(self, filename, preserve_white):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_parse_xml_archive_entry()`.
(12)                    Parse the contents of an archive entry into a tree of xml nodes.
(12)                    preserve_white: whether to keep or delete all-whitespace nodes.
(8)                 """
(8)                 return _mupdf.FzArchive_fz_parse_xml_archive_entry(self, filename,
```

```
preserve_white)
    def fz_read_archive_entry(self, name):
        r"""
        Class-aware wrapper for `::fz_read_archive_entry()`.
            Reads all bytes in an archive entry
            into a buffer.
            name: Entry name to look for, this must be an exact match to
            the entry name in the archive.
            Throws an exception if a matching entry cannot be found.
        """
        return _mupdf.FzArchive_fz_read_archive_entry(self, name)
    def fz_tree_archive_add_buffer(self, name, buf):
        r"""
        Class-aware wrapper for `::fz_tree_archive_add_buffer()`.
            Add a named buffer to an existing tree archive.
            The tree will take a new reference to the buffer. Ownership
            is not transferred.
        """
        return _mupdf.FzArchive_fz_tree_archive_add_buffer(self, name, buf)
    def fz_tree_archive_add_data(self, name, data, size):
        r"""
        Class-aware wrapper for `::fz_tree_archive_add_data()`.
            Add a named block of data to an existing tree archive.
            The data will be copied into a buffer, and so the caller
            may free it as soon as this returns.
        """
        return _mupdf.FzArchive_fz_tree_archive_add_data(self, name, data,
size)
    def fz_try_open_archive_entry(self, name):
        r"""
        Class-aware wrapper for `::fz_try_open_archive_entry()`.
            Opens an archive entry as a stream.
            Returns NULL if a matching entry cannot be found, otherwise
            behaves exactly as fz_open_archive_entry.
        """
        return _mupdf.FzArchive_fz_try_open_archive_entry(self, name)
    def fz_try_parse_xml_archive_entry(self, filename, preserve_white):
        r"""
        Class-aware wrapper for `::fz_try_parse_xml_archive_entry()`.
            Try and parse the contents of an archive entry into a tree of xml
nodes.

            preserve_white: whether to keep or delete all-whitespace nodes.
            Will return NULL if the archive entry can't be found. Otherwise
behaves

            the same as fz_parse_xml_archive_entry. May throw exceptions.
        """
        return _mupdf.FzArchive_fz_try_parse_xml_archive_entry(self, filename,
preserve_white)
    def fz_try_read_archive_entry(self, name):
        r"""
        Class-aware wrapper for `::fz_try_read_archive_entry()`.
            Reads all bytes in an archive entry
            into a buffer.
            name: Entry name to look for, this must be an exact match to
            the entry name in the archive.
            Returns NULL if a matching entry cannot be found. Otherwise
behaves

            the same as fz_read_archive_entry. Exceptions may be thrown.
        """
        return _mupdf.FzArchive_fz_try_read_archive_entry(self, name)
    def __init__(self, *args):
        r"""
        *Overload 1:*
         == Constructors.  Constructor using `fz_new_archive_of_size()`.
        |
        *Overload 2:*
         Constructor using `fz_new_multi_archive()`.
                Create a new multi archive (initially empty).
        |
```

```
15898: (8)                  *Overload 3:*
15899: (9)                   Constructor using `fz_new_tree_archive()`.
15900: (16)                         Create an archive that holds named buffers.
15901: (16)                         tree can either be a preformed tree with fz_buffers as values,
15902: (16)                         or it can be NULL for an empty tree.
15903: (8)                  |
15904: (8)                  *Overload 4:*
15905: (9)                   Copy constructor using `fz_keep_archive()`.
15906: (8)                  |
15907: (8)                  *Overload 5:*
15908: (9)                   Constructor using raw copy of pre-existing `::fz_archive`.
15909: (8)                  """
15910: (8)                  _mupdf.FzArchive_swiginit(self, _mupdf.new_FzArchive(*args))
15911: (4)              __swig_destroy__ = _mupdf.delete_FzArchive
15912: (4)              def m_internal_value(self):
15913: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15914: (8)                  return _mupdf.FzArchive_m_internal_value(self)
15915: (4)              m_internal = property(_mupdf.FzArchive_m_internal_get,
_mupdf.FzArchive_m_internal_set, doc=r""" Pointer to wrapped data.""")
15916: (4)              s_num_instances = property(_mupdf.FzArchive_s_num_instances_get,
_mupdf.FzArchive_s_num_instances_set)
15917: (0)          _mupdf.FzArchive_swigregister(FzArchive)
15918: (0)          class FzArchiveHandler(object):
15919: (4)              r""" Wrapper class for struct `fz_archive_handler`. Not copyable or
assignable."""
15920: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15921: (4)              __repr__ = _swig_repr
15922: (4)              def fz_register_archive_handler(self):
15923: (8)                  r""" Class-aware wrapper for `::fz_register_archive_handler()`."""
15924: (8)                  return _mupdf.FzArchiveHandler_fz_register_archive_handler(self)
15925: (4)              def __init__(self, *args):
15926: (8)                  r"""
15927: (8)                  *Overload 1:*
15928: (8)                  Default constructor, sets `m_internal` to null.
15929: (8)                  |
15930: (8)                  *Overload 2:*
15931: (8)                  Constructor using raw copy of pre-existing `::fz_archive_handler`.
15932: (8)                  """
15933: (8)                  _mupdf.FzArchiveHandler_swiginit(self,
_mupdf.new_FzArchiveHandler(*args))
15934: (4)              __swig_destroy__ = _mupdf.delete_FzArchiveHandler
15935: (4)              def m_internal_value(self):
15936: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15937: (8)                  return _mupdf.FzArchiveHandler_m_internal_value(self)
15938: (4)              m_internal = property(_mupdf.FzArchiveHandler_m_internal_get,
_mupdf.FzArchiveHandler_m_internal_set, doc=r""" Pointer to wrapped data.""")
15939: (4)              s_num_instances = property(_mupdf.FzArchiveHandler_s_num_instances_get,
_mupdf.FzArchiveHandler_s_num_instances_set)
15940: (0)          _mupdf.FzArchiveHandler_swigregister(FzArchiveHandler)
15941: (0)          class FzArchiveHandlerContext(object):
15942: (4)              r""" Wrapper class for struct `fz_archive_handler_context`. Not copyable
or assignable."""
15943: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15944: (4)              __repr__ = _swig_repr
15945: (4)              def __init__(self, *args):
15946: (8)                  r"""
15947: (8)                  *Overload 1:*
15948: (8)                  Default constructor, sets `m_internal` to null.
15949: (8)                  |
15950: (8)                  *Overload 2:*
15951: (8)                  Constructor using raw copy of pre-existing
`::fz_archive_handler_context`.
15952: (8)                  """
15953: (8)                  _mupdf.FzArchiveHandlerContext_swiginit(self,
_mupdf.new_FzArchiveHandlerContext(*args))
```

```
15954: (4)              __swig_destroy__ = _mupdf.delete_FzArchiveHandlerContext
15955: (4)              def m_internal_value(self):
15956: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
15957: (8)                  return _mupdf.FzArchiveHandlerContext_m_internal_value(self)
15958: (4)              m_internal = property(_mupdf.FzArchiveHandlerContext_m_internal_get,
_mupdf.FzArchiveHandlerContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
15959: (4)              s_num_instances =
property(_mupdf.FzArchiveHandlerContext_s_num_instances_get,
_mupdf.FzArchiveHandlerContext_s_num_instances_set)
15960: (0)          _mupdf.FzArchiveHandlerContext_swigregister(FzArchiveHandlerContext)
15961: (0)          class FzBandWriter(object):
15962: (4)              r"""
15963: (4)              Wrapper class for struct `fz_band_writer`. Not copyable or assignable.
15964: (4)              fz_band_writer
15965: (4)              """
15966: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
15967: (4)              __repr__ = _swig_repr
15968: (4)              MONO = _mupdf.FzBandWriter_MONO
15969: (4)              COLOR = _mupdf.FzBandWriter_COLOR
15970: (4)              PNG = _mupdf.FzBandWriter_PNG
15971: (4)              PNM = _mupdf.FzBandWriter_PNM
15972: (4)              PAM = _mupdf.FzBandWriter_PAM
15973: (4)              PBM = _mupdf.FzBandWriter_PBM
15974: (4)              PKM = _mupdf.FzBandWriter_PKM
15975: (4)              PS = _mupdf.FzBandWriter_PS
15976: (4)              PSD = _mupdf.FzBandWriter_PSD
15977: (4)              def fz_close_band_writer(self):
15978: (8)                  r"""
15979: (8)                  Class-aware wrapper for `::fz_close_band_writer()`.
15980: (12)                     Finishes up the output and closes the band writer. After this
15981: (12)                     call no more headers or bands may be written.
15982: (8)                  """
15983: (8)                  return _mupdf.FzBandWriter_fz_close_band_writer(self)
15984: (4)              def fz_pdfocr_band_writer_set_progress(self, progress_fn, progress_arg):
15985: (8)                  r"""
15986: (8)                  Class-aware wrapper for `::fz_pdfocr_band_writer_set_progress()`.
15987: (12)                     Set the progress callback for a pdfocr bandwriter.
15988: (8)                  """
15989: (8)                  return _mupdf.FzBandWriter_fz_pdfocr_band_writer_set_progress(self,
progress_fn, progress_arg)
15990: (4)              def fz_write_band(self, stride, band_height, samples):
15991: (8)                  r"""
15992: (8)                  Class-aware wrapper for `::fz_write_band()`.
15993: (12)                     Cause a band writer to write the next band
15994: (12)                     of data for an image.
15995: (12)                     stride: The byte offset from the first byte of the data
15996: (12)                     for a pixel to the first byte of the data for the same pixel
15997: (12)                     on the row below.
15998: (12)                     band_height: The number of lines in this band.
15999: (12)                     samples: Pointer to first byte of the data.
16000: (8)                  """
16001: (8)                  return _mupdf.FzBandWriter_fz_write_band(self, stride, band_height,
samples)
16002: (4)              def fz_write_header(self, w, h, n, alpha, xres, yres, pagenum, cs, seps):
16003: (8)                  r"""
16004: (8)                  Class-aware wrapper for `::fz_write_header()`.
16005: (12)                     Cause a band writer to write the header for
16006: (12)                     a banded image with the given properties/dimensions etc. This
16007: (12)                     also configures the bandwriter for the format of the data to be
16008: (12)                     passed in future calls.
16009: (12)                     w, h: Width and Height of the entire page.
16010: (12)                     n: Number of components (including spots and alphas).
16011: (12)                     alpha: Number of alpha components.
16012: (12)                     xres, yres: X and Y resolutions in dpi.
16013: (12)                     cs: Colorspace (NULL for bitmaps)
16014: (12)                     seps: Separation details (or NULL).
16015: (8)                  """
```

```
16016: (8)                  return _mupdf.FzBandWriter_fz_write_header(self, w, h, n, alpha, xres,
yres, pagenum, cs, seps)
16017: (4)              def __init__(self, *args):
16018: (8)                  r"""
16019: (8)                  *Overload 1:*
16020: (9)                   == Constructors.  Constructor using `fz_new_band_writer_of_size()`.
16021: (8)                  |
16022: (8)                  *Overload 2:*
16023: (9)                   Constructor using `fz_new_color_pcl_band_writer()`.
16024: (16)                         Create a new band writer, outputing color pcl.
16025: (13)                      Constructor using `fz_new_mono_pcl_band_writer()`.
16026: (16)                         Create a new band writer, outputing monochrome pcl.
16027: (8)                  |
16028: (8)                  *Overload 3:*
16029: (9)                   Constructor using `fz_new_mono_pwg_band_writer()`.
16030: (16)                         Create a new monochrome pwg band writer.
16031: (8)                  |
16032: (8)                  *Overload 4:*
16033: (9)                   Constructor using `fz_new_pam_band_writer()`.
16034: (16)                         Create a band writer targetting pnm (greyscale, rgb or cmyk,
16035: (16)                         with or without alpha).
16036: (13)                      Constructor using `fz_new_pbm_band_writer()`.
16037: (16)                         Create a new band writer, targetting pbm.
16038: (13)                      Constructor using `fz_new_pclm_band_writer()`.
16039: (16)                         Create a new band writer, outputing pclm
16040: (8)                  |
16041: (8)                  *Overload 5:*
16042: (9)                   Constructor using `fz_new_pdfocr_band_writer()`.
16043: (16)                         Create a new band writer, outputing pdfocr.
16044: (16)                         Ownership of output stays with the caller, the band writer
16045: (16)                         borrows the reference. The caller must keep the output around
16046: (16)                         for the duration of the band writer, and then close/drop as
16047: (16)                         appropriate.
16048: (8)                  |
16049: (8)                  *Overload 6:*
16050: (9)                   Constructor using `fz_new_pkm_band_writer()`.
16051: (16)                         Create a new pkm band writer for CMYK pixmaps.
16052: (13)                      Constructor using `fz_new_png_band_writer()`.
16053: (16)                         Create a new png band writer (greyscale or RGB, with or
without
16054: (16)                         alpha).
16055: (8)                  |
16056: (8)                  *Overload 7:*
16057: (9)                   Constructor using `fz_new_pnm_band_writer()`.
16058: (16)                         Create a band writer targetting pnm (greyscale or rgb, no
16059: (16)                         alpha).
16060: (13)                      Constructor using `fz_new_ps_band_writer()`.
16061: (16)                         Create a postscript band writer for gray, rgb, or cmyk, no
16062: (16)                         alpha.
16063: (13)                      Constructor using `fz_new_psd_band_writer()`.
16064: (16)                         Open a PSD band writer.
16065: (13)                      Constructor using `fz_new_pwg_band_writer()`.
16066: (16)                         Create a new color pwg band writer.
16067: (13)                      Constructor using fz_new_mono_pcl_band_writer() or
fz_new_color_pcl_band_writer().
16068: (8)                  |
16069: (8)                  *Overload 8:*
16070: (9)                   Constructor using fz_new_p*_band_writer().
16071: (8)                  |
16072: (8)                  *Overload 9:*
16073: (9)                   Constructor using fz_new_mono_pwg_band_writer() or
fz_new_pwg_band_writer().
16074: (8)                  |
16075: (8)                  *Overload 10:*
16076: (9)                   Default constructor, sets `m_internal` to null.
16077: (8)                  |
16078: (8)                  *Overload 11:*
16079: (9)                   Constructor using raw copy of pre-existing `::fz_band_writer`.
16080: (8)                  """
```

```
16081: (8)                  _mupdf.FzBandWriter_swiginit(self, _mupdf.new_FzBandWriter(*args))
16082: (4)              __swig_destroy__ = _mupdf.delete_FzBandWriter
16083: (4)              def m_internal_value(self):
16084: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
16085: (8)                  return _mupdf.FzBandWriter_m_internal_value(self)
16086: (4)              m_internal = property(_mupdf.FzBandWriter_m_internal_get,
_mupdf.FzBandWriter_m_internal_set, doc=r""" Pointer to wrapped data.""")
16087: (4)              s_num_instances = property(_mupdf.FzBandWriter_s_num_instances_get,
_mupdf.FzBandWriter_s_num_instances_set)
16088: (0)          _mupdf.FzBandWriter_swigregister(FzBandWriter)
16089: (0)          class FzBitmap(object):
16090: (4)              r"""
16091: (4)              Wrapper class for struct `fz_bitmap`.
16092: (4)              Bitmaps have 1 bit per component. Only used for creating
16093: (4)              halftoned versions of contone buffers, and saving out. Samples
16094: (4)              are stored msb first, akin to pbms.
16095: (4)              The internals of this struct are considered implementation
16096: (4)              details and subject to change. Where possible, accessor
16097: (4)              functions should be used in preference.
16098: (4)              """
16099: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16100: (4)              __repr__ = _swig_repr
16101: (4)              def fz_bitmap_details(self, w, h, n, stride):
16102: (8)                  r"""
16103: (8)                  Class-aware wrapper for `::fz_bitmap_details()`.
16104: (8)                  This method has out-params. Python/C# wrappers look like:
16105: (12)                     `fz_bitmap_details()` => `(int w, int h, int n, int stride)`
16106: (12)                     Retrieve details of a given bitmap.
16107: (12)                     bitmap: The bitmap to query.
16108: (12)                     w: Pointer to storage to retrieve width (or NULL).
16109: (12)                     h: Pointer to storage to retrieve height (or NULL).
16110: (12)                     n: Pointer to storage to retrieve number of color components (or
16111: (12)                     NULL).
16112: (12)                     stride: Pointer to storage to retrieve bitmap stride (or NULL).
16113: (8)                  """
16114: (8)                  return _mupdf.FzBitmap_fz_bitmap_details(self, w, h, n, stride)
16115: (4)              def fz_clear_bitmap(self):
16116: (8)                  r"""
16117: (8)                  Class-aware wrapper for `::fz_clear_bitmap()`.
16118: (12)                     Set the entire bitmap to 0.
16119: (12)                     Never throws exceptions.
16120: (8)                  """
16121: (8)                  return _mupdf.FzBitmap_fz_clear_bitmap(self)
16122: (4)              def fz_invert_bitmap(self):
16123: (8)                  r"""
16124: (8)                  Class-aware wrapper for `::fz_invert_bitmap()`.
16125: (12)                     Invert bitmap.
16126: (12)                     Never throws exceptions.
16127: (8)                  """
16128: (8)                  return _mupdf.FzBitmap_fz_invert_bitmap(self)
16129: (4)              def fz_save_bitmap_as_pbm(self, filename):
16130: (8)                  r"""
16131: (8)                  Class-aware wrapper for `::fz_save_bitmap_as_pbm()`.
16132: (12)                     Save a bitmap as a pbm.
16133: (8)                  """
16134: (8)                  return _mupdf.FzBitmap_fz_save_bitmap_as_pbm(self, filename)
16135: (4)              def fz_save_bitmap_as_pcl(self, filename, append, pcl):
16136: (8)                  r"""
16137: (8)                  Class-aware wrapper for `::fz_save_bitmap_as_pcl()`.
16138: (12)                     Save a bitmap as mono PCL.
16139: (8)                  """
16140: (8)                  return _mupdf.FzBitmap_fz_save_bitmap_as_pcl(self, filename, append,
pcl)
16141: (4)              def fz_save_bitmap_as_pkm(self, filename):
16142: (8)                  r"""
16143: (8)                  Class-aware wrapper for `::fz_save_bitmap_as_pkm()`.
16144: (12)                     Save a CMYK bitmap as a pkm.
```

```
(8)                 """
(8)                 return _mupdf.FzBitmap_fz_save_bitmap_as_pkm(self, filename)
(4)             def fz_save_bitmap_as_pwg(self, filename, append, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_bitmap_as_pwg()`.
(12)                    Save a bitmap as a PWG.
(8)                 """
(8)                 return _mupdf.FzBitmap_fz_save_bitmap_as_pwg(self, filename, append,
pwg)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using `fz_new_bitmap()`.
(16)                        Create a new bitmap.
(16)                        w, h: Width and Height for the bitmap
(16)                        n: Number of color components (assumed to be a divisor of 8)
(16)                        xres, yres: X and Y resolutions (in pixels per inch).
(16)                        Returns pointer to created bitmap structure. The bitmap
(16)                        data is uninitialised.
(8)                 |
(8)                 *Overload 2:*
(9)                  Constructor using `fz_new_bitmap_from_pixmap()`.
(16)                        Make a bitmap from a pixmap and a halftone.
(16)                        pix: The pixmap to generate from. Currently must be a single
(16)                        color component with no alpha.
(16)                        ht: The halftone to use. NULL implies the default halftone.
(16)                        Returns the resultant bitmap. Throws exceptions in the case of
(16)                        failure to allocate.
(8)                 |
(8)                 *Overload 3:*
(9)                  Constructor using `fz_new_bitmap_from_pixmap_band()`.
(16)                        Make a bitmap from a pixmap and a
(16)                        halftone, allowing for the position of the pixmap within an
(16)                        overall banded rendering.
(16)                        pix: The pixmap to generate from. Currently must be a single
(16)                        color component with no alpha.
(16)                        ht: The halftone to use. NULL implies the default halftone.
(16)                        band_start: Vertical offset within the overall banded
rendering
(16)                        (in pixels)
(16)                        Returns the resultant bitmap. Throws exceptions in the case of
(16)                        failure to allocate.
(8)                 |
(8)                 *Overload 4:*
(9)                  Copy constructor using `fz_keep_bitmap()`.
(8)                 |
(8)                 *Overload 5:*
(9)                  Default constructor, sets `m_internal` to null.
(8)                 |
(8)                 *Overload 6:*
(9)                  Constructor using raw copy of pre-existing `::fz_bitmap`.
(8)                 """
(8)                 _mupdf.FzBitmap_swiginit(self, _mupdf.new_FzBitmap(*args))
(4)             def refs(self):
(8)                 return _mupdf.FzBitmap_refs(self)
(4)             def w(self):
(8)                 return _mupdf.FzBitmap_w(self)
(4)             def h(self):
(8)                 return _mupdf.FzBitmap_h(self)
(4)             def stride(self):
(8)                 return _mupdf.FzBitmap_stride(self)
(4)             def n(self):
(8)                 return _mupdf.FzBitmap_n(self)
(4)             def xres(self):
(8)                 return _mupdf.FzBitmap_xres(self)
(4)             def yres(self):
(8)                 return _mupdf.FzBitmap_yres(self)
(4)             def samples(self):
(8)                 return _mupdf.FzBitmap_samples(self)
```

```
16212: (4)              __swig_destroy__ = _mupdf.delete_FzBitmap
16213: (4)              def m_internal_value(self):
16214: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
16215: (8)                  return _mupdf.FzBitmap_m_internal_value(self)
16216: (4)              m_internal = property(_mupdf.FzBitmap_m_internal_get,
_mupdf.FzBitmap_m_internal_set, doc=r""" Pointer to wrapped data.""")
16217: (4)              s_num_instances = property(_mupdf.FzBitmap_s_num_instances_get,
_mupdf.FzBitmap_s_num_instances_set)
16218: (0)          _mupdf.FzBitmap_swigregister(FzBitmap)
16219: (0)          class FzBuffer(object):
16220: (4)              r"""
16221: (4)              Wrapper class for struct `fz_buffer`.
16222: (4)              fz_buffer is a wrapper around a dynamically allocated array of
16223: (4)              bytes.
16224: (4)              Buffers have a capacity (the number of bytes storage immediately
16225: (4)              available) and a current size.
16226: (4)              The contents of the structure are considered implementation
16227: (4)              details and are subject to change. Users should use the accessor
16228: (4)              functions in preference.
16229: (4)              """
16230: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16231: (4)              __repr__ = _swig_repr
16232: (4)              @staticmethod
16233: (4)              def fz_new_buffer_from_copied_data(data, size):
16234: (8)                  r"""
16235: (8)                  Class-aware wrapper for `::fz_new_buffer_from_copied_data()`.
16236: (12)                     Create a new buffer containing a copy of the passed data.
16237: (8)                  """
16238: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_copied_data(data, size)
16239: (4)              @staticmethod
16240: (4)              def fz_new_buffer_from_image_as_pnm(image, color_params):
16241: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pnm()`."""
16242: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_image_as_pnm(image,
color_params)
16243: (4)              @staticmethod
16244: (4)              def fz_new_buffer_from_image_as_pam(image, color_params):
16245: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pam()`."""
16246: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_image_as_pam(image,
color_params)
16247: (4)              @staticmethod
16248: (4)              def fz_new_buffer_from_image_as_psd(image, color_params):
16249: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_psd()`."""
16250: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_image_as_psd(image,
color_params)
16251: (4)              @staticmethod
16252: (4)              def fz_new_buffer_from_pixmap_as_pnm(pixmap, color_params):
16253: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_pnm()`."""
16254: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_pixmap_as_pnm(pixmap,
color_params)
16255: (4)              @staticmethod
16256: (4)              def fz_new_buffer_from_pixmap_as_pam(pixmap, color_params):
16257: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_pam()`."""
16258: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_pixmap_as_pam(pixmap,
color_params)
16259: (4)              @staticmethod
16260: (4)              def fz_new_buffer_from_pixmap_as_psd(pix, color_params):
16261: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_psd()`."""
16262: (8)                  return _mupdf.FzBuffer_fz_new_buffer_from_pixmap_as_psd(pix,
color_params)
16263: (4)              def fz_append_base64(self, data, size, newline):
16264: (8)                  r"""
16265: (8)                  Class-aware wrapper for `::fz_append_base64()`.
16266: (12)                     Write a base64 encoded data block, optionally with periodic
newlines.
```

```
16267: (8)                    """
16268: (8)                    return _mupdf.FzBuffer_fz_append_base64(self, data, size, newline)
16269: (4)                def fz_append_base64_buffer(self, data, newline):
16270: (8)                    r"""
16271: (8)                    Class-aware wrapper for `::fz_append_base64_buffer()`.
16272: (12)                       Append a base64 encoded fz_buffer, optionally with periodic
newlines.
16273: (8)                    """
16274: (8)                    return _mupdf.FzBuffer_fz_append_base64_buffer(self, data, newline)
16275: (4)                def fz_append_bits(self, value, count):
16276: (8)                    r""" Class-aware wrapper for `::fz_append_bits()`."""
16277: (8)                    return _mupdf.FzBuffer_fz_append_bits(self, value, count)
16278: (4)                def fz_append_bits_pad(self):
16279: (8)                    r""" Class-aware wrapper for `::fz_append_bits_pad()`."""
16280: (8)                    return _mupdf.FzBuffer_fz_append_bits_pad(self)
16281: (4)                def fz_append_buffer(self, source):
16282: (8)                    r"""
16283: (8)                    Class-aware wrapper for `::fz_append_buffer()`.
16284: (12)                       Append the contents of the source buffer onto the end of the
16285: (12)                       destination buffer, extending automatically as required.
16286: (12)                       Ownership of buffers does not change.
16287: (8)                    """
16288: (8)                    return _mupdf.FzBuffer_fz_append_buffer(self, source)
16289: (4)                def fz_append_byte(self, c):
16290: (8)                    r""" Class-aware wrapper for `::fz_append_byte()`."""
16291: (8)                    return _mupdf.FzBuffer_fz_append_byte(self, c)
16292: (4)                def fz_append_data(self, data, len):
16293: (8)                    r"""
16294: (8)                    Class-aware wrapper for `::fz_append_data()`.
16295: (12)                       fz_append_*: Append data to a buffer.
16296: (12)                       The buffer will automatically grow as required.
16297: (8)                    """
16298: (8)                    return _mupdf.FzBuffer_fz_append_data(self, data, len)
16299: (4)                def fz_append_image_as_data_uri(self, image):
16300: (8)                    r""" Class-aware wrapper for `::fz_append_image_as_data_uri()`."""
16301: (8)                    return _mupdf.FzBuffer_fz_append_image_as_data_uri(self, image)
16302: (4)                def fz_append_int16_be(self, x):
16303: (8)                    r""" Class-aware wrapper for `::fz_append_int16_be()`."""
16304: (8)                    return _mupdf.FzBuffer_fz_append_int16_be(self, x)
16305: (4)                def fz_append_int16_le(self, x):
16306: (8)                    r""" Class-aware wrapper for `::fz_append_int16_le()`."""
16307: (8)                    return _mupdf.FzBuffer_fz_append_int16_le(self, x)
16308: (4)                def fz_append_int32_be(self, x):
16309: (8)                    r""" Class-aware wrapper for `::fz_append_int32_be()`."""
16310: (8)                    return _mupdf.FzBuffer_fz_append_int32_be(self, x)
16311: (4)                def fz_append_int32_le(self, x):
16312: (8)                    r""" Class-aware wrapper for `::fz_append_int32_le()`."""
16313: (8)                    return _mupdf.FzBuffer_fz_append_int32_le(self, x)
16314: (4)                def fz_append_pdf_string(self, text):
16315: (8)                    r"""
16316: (8)                    Class-aware wrapper for `::fz_append_pdf_string()`.
16317: (12)                       fz_append_pdf_string: Append a string with PDF syntax quotes and
16318: (12)                       escapes.
16319: (12)                       The buffer will automatically grow as required.
16320: (8)                    """
16321: (8)                    return _mupdf.FzBuffer_fz_append_pdf_string(self, text)
16322: (4)                def fz_append_pixmap_as_data_uri(self, pixmap):
16323: (8)                    r""" Class-aware wrapper for `::fz_append_pixmap_as_data_uri()`."""
16324: (8)                    return _mupdf.FzBuffer_fz_append_pixmap_as_data_uri(self, pixmap)
16325: (4)                def fz_append_rune(self, c):
16326: (8)                    r""" Class-aware wrapper for `::fz_append_rune()`."""
16327: (8)                    return _mupdf.FzBuffer_fz_append_rune(self, c)
16328: (4)                def fz_append_string(self, data):
16329: (8)                    r""" Class-aware wrapper for `::fz_append_string()`."""
16330: (8)                    return _mupdf.FzBuffer_fz_append_string(self, data)
16331: (4)                def fz_buffer_extract(self, data):
16332: (8)                    r"""
16333: (8)                    Class-aware wrapper for `::fz_buffer_extract()`.
16334: (8)                    This method has out-params. Python/C# wrappers look like:
```

```
16335: (12)                     `fz_buffer_extract()` => `(size_t, unsigned char *data)`
16336: (12)                     Take ownership of buffer contents.
16337: (12)                     Performs the same task as fz_buffer_storage, but ownership of
16338: (12)                     the data buffer returns with this call. The buffer is left
16339: (12)                     empty.
16340: (12)                     Note: Bad things may happen if this is called on a buffer with
16341: (12)                     multiple references that is being used from multiple threads.
16342: (12)                     data: Pointer to place to retrieve data pointer.
16343: (12)                     Returns length of stream.
16344: (8)                  """
16345: (8)                  return _mupdf.FzBuffer_fz_buffer_extract(self, data)
16346: (4)              def fz_buffer_storage(self, datap):
16347: (8)                  r"""
16348: (8)                  Class-aware wrapper for `::fz_buffer_storage()`.
16349: (8)                  This method has out-params. Python/C# wrappers look like:
16350: (12)                     `fz_buffer_storage()` => `(size_t, unsigned char *datap)`
16351: (12)                     Retrieve internal memory of buffer.
16352: (12)                     datap: Output parameter that will be pointed to the data.
16353: (12)                     Returns the current size of the data in bytes.
16354: (8)                  """
16355: (8)                  return _mupdf.FzBuffer_fz_buffer_storage(self, datap)
16356: (4)              def fz_clear_buffer(self):
16357: (8)                  r"""
16358: (8)                  Class-aware wrapper for `::fz_clear_buffer()`.
16359: (12)                     Empties the buffer. Storage is not freed, but is held ready
16360: (12)                     to be reused as the buffer is refilled.
16361: (12)                     Never throws exceptions.
16362: (8)                  """
16363: (8)                  return _mupdf.FzBuffer_fz_clear_buffer(self)
16364: (4)              def fz_clone_buffer(self):
16365: (8)                  r"""
16366: (8)                  Class-aware wrapper for `::fz_clone_buffer()`.
16367: (12)                     Make a new buffer, containing a copy of the data used in
16368: (12)                     the original.
16369: (8)                  """
16370: (8)                  return _mupdf.FzBuffer_fz_clone_buffer(self)
16371: (4)              def fz_grow_buffer(self):
16372: (8)                  r"""
16373: (8)                  Class-aware wrapper for `::fz_grow_buffer()`.
16374: (12)                     Make some space within a buffer (i.e. ensure that
16375: (12)                     capacity > size).
16376: (8)                  """
16377: (8)                  return _mupdf.FzBuffer_fz_grow_buffer(self)
16378: (4)              def fz_load_jbig2_globals(self):
16379: (8)                  r"""
16380: (8)                  Class-aware wrapper for `::fz_load_jbig2_globals()`.
16381: (12)                     Create a jbig2 globals record from a buffer.
16382: (12)                     Immutable once created.
16383: (8)                  """
16384: (8)                  return _mupdf.FzBuffer_fz_load_jbig2_globals(self)
16385: (4)              def fz_md5_buffer(self, digest):
16386: (8)                  r"""
16387: (8)                  Class-aware wrapper for `::fz_md5_buffer()`.
16388: (12)                     Create an MD5 digest from buffer contents.
16389: (12)                     Never throws exceptions.
16390: (8)                  """
16391: (8)                  return _mupdf.FzBuffer_fz_md5_buffer(self, digest)
16392: (4)              def fz_new_display_list_from_svg(self, base_uri, dir, w, h):
16393: (8)                  r"""
16394: (8)                  Class-aware wrapper for `::fz_new_display_list_from_svg()`.
16395: (8)                  This method has out-params. Python/C# wrappers look like:
16396: (12)                     `fz_new_display_list_from_svg(const char *base_uri, ::fz_archive
*dir)` => `(fz_display_list *, float w, float h)`
16397: (12)                     Parse an SVG document into a display-list.
16398: (8)                  """
16399: (8)                  return _mupdf.FzBuffer_fz_new_display_list_from_svg(self, base_uri,
dir, w, h)
16400: (4)              def fz_new_image_from_buffer(self):
16401: (8)                  r"""
```

```
(8)                 Class-aware wrapper for `::fz_new_image_from_buffer()`.
(12)                    Create a new image from a
(12)                    buffer of data, inferring its type from the format
(12)                    of the data.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_new_image_from_buffer(self)
(4)             def fz_new_image_from_svg(self, base_uri, dir):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_image_from_svg()`.
(12)                    Create a scalable image from an SVG document.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_new_image_from_svg(self, base_uri, dir)
(4)             def fz_open_buffer(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_buffer()`.
(12)                    Open a buffer as a stream.
(12)                    buf: The buffer to open. Ownership of the buffer is NOT passed
(12)                    in (this function takes its own reference).
(12)                    Returns pointer to newly created stream. May throw exceptions on
(12)                    failure to allocate.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_open_buffer(self)
(4)             def fz_parse_xml(self, preserve_white):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_parse_xml()`.
(12)                    Parse the contents of buffer into a tree of xml nodes.
(12)                    preserve_white: whether to keep or delete all-whitespace nodes.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_parse_xml(self, preserve_white)
(4)             def fz_parse_xml_from_html5(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_parse_xml_from_html5()`.
(12)                    Parse the contents of a buffer into a tree of XML nodes,
(12)                    using the HTML5 parsing algorithm.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_parse_xml_from_html5(self)
(4)             def fz_resize_buffer(self, capacity):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_resize_buffer()`.
(12)                    Ensure that a buffer has a given capacity,
(12)                    truncating data if required.
(12)                    capacity: The desired capacity for the buffer. If the current
(12)                    size of the buffer contents is smaller than capacity, it is
(12)                    truncated.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_resize_buffer(self, capacity)
(4)             def fz_save_buffer(self, filename):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_buffer()`.
(12)                    Save the contents of a buffer to a file.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_save_buffer(self, filename)
(4)             def fz_slice_buffer(self, start, end):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_slice_buffer()`.
(12)                    Create a new buffer with a (subset of) the data from the buffer.
(12)                    start: if >= 0, offset from start of buffer, if < 0 offset from
end of buffer.
(12)                    end: if >= 0, offset from start of buffer, if < 0 offset from end
of buffer.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_slice_buffer(self, start, end)
(4)             def fz_string_from_buffer(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_string_from_buffer()`.
(12)                    Ensure that a buffer's data ends in a
(12)                    0 byte, and return a pointer to it.
(8)                 """
(8)                 return _mupdf.FzBuffer_fz_string_from_buffer(self)
```

```
16469: (4)              def fz_subset_cff_for_gids(self, gids, num_gids, symbolic, cidfont):
16470: (8)                  r"""
16471: (8)                  Class-aware wrapper for `::fz_subset_cff_for_gids()`.
16472: (8)                  This method has out-params. Python/C# wrappers look like:
16473: (12)                     `fz_subset_cff_for_gids(int num_gids, int symbolic, int cidfont)`
=> `(fz_buffer *, int gids)`
16474: (8)                  """
16475: (8)                  return _mupdf.FzBuffer_fz_subset_cff_for_gids(self, gids, num_gids,
symbolic, cidfont)
16476: (4)              def fz_subset_ttf_for_gids(self, gids, num_gids, symbolic, cidfont):
16477: (8)                  r"""
16478: (8)                  Class-aware wrapper for `::fz_subset_ttf_for_gids()`.
16479: (8)                  This method has out-params. Python/C# wrappers look like:
16480: (12)                     `fz_subset_ttf_for_gids(int num_gids, int symbolic, int cidfont)`
=> `(fz_buffer *, int gids)`
16481: (8)                  """
16482: (8)                  return _mupdf.FzBuffer_fz_subset_ttf_for_gids(self, gids, num_gids,
symbolic, cidfont)
16483: (4)              def fz_terminate_buffer(self):
16484: (8)                  r"""
16485: (8)                  Class-aware wrapper for `::fz_terminate_buffer()`.
16486: (12)                     Zero-terminate buffer in order to use as a C string.
16487: (12)                     This byte is invisible and does not affect the length of the
16488: (12)                     buffer as returned by fz_buffer_storage. The zero byte is
16489: (12)                     written *after* the data, and subsequent writes will overwrite
16490: (12)                     the terminating byte.
16491: (12)                     Subsequent changes to the size of the buffer (such as by
16492: (12)                     fz_buffer_trim, fz_buffer_grow, fz_resize_buffer, etc) may
16493: (12)                     invalidate this.
16494: (8)                  """
16495: (8)                  return _mupdf.FzBuffer_fz_terminate_buffer(self)
16496: (4)              def fz_trim_buffer(self):
16497: (8)                  r"""
16498: (8)                  Class-aware wrapper for `::fz_trim_buffer()`.
16499: (12)                     Trim wasted capacity from a buffer by resizing internal memory.
16500: (8)                  """
16501: (8)                  return _mupdf.FzBuffer_fz_trim_buffer(self)
16502: (4)              def pdf_append_token(self, tok, lex):
16503: (8)                  r""" Class-aware wrapper for `::pdf_append_token()`."""
16504: (8)                  return _mupdf.FzBuffer_pdf_append_token(self, tok, lex)
16505: (4)              def pdf_new_buffer_processor(self, ahxencode, newlines):
16506: (8)                  r""" Class-aware wrapper for `::pdf_new_buffer_processor()`."""
16507: (8)                  return _mupdf.FzBuffer_pdf_new_buffer_processor(self, ahxencode,
newlines)
16508: (4)              def __init__(self, *args):
16509: (8)                  r"""
16510: (8)                  *Overload 1:*
16511: (9)                   == Constructors.  Constructor using `fz_new_buffer()`.
16512: (8)                  |
16513: (8)                  *Overload 2:*
16514: (9)                   Constructor using `fz_new_buffer_from_base64()`.
16515: (16)                         Create a new buffer with data decoded from a base64 input
string.
16516: (8)                  |
16517: (8)                  *Overload 3:*
16518: (9)                   Constructor using `fz_new_buffer_from_data()`.
16519: (16)                         Create a new buffer with existing data.
16520: (16)                         data: Pointer to existing data.
16521: (16)                         size: Size of existing data.
16522: (16)                         Takes ownership of data. Does not make a copy. Calls fz_free
on
16523: (16)                         the data when the buffer is deallocated. Do not use 'data'
after
16524: (16)                         passing to this function.
16525: (16)                         Returns pointer to new buffer. Throws exception on allocation
16526: (16)                         failure.
16527: (8)                  |
16528: (8)                  *Overload 4:*
16529: (9)                   Constructor using `fz_new_buffer_from_display_list()`.
```

```
16530: (8)                  |
16531: (8)                  *Overload 5:*
16532: (9)                   Constructor using `fz_new_buffer_from_image_as_jpeg()`.
16533: (8)                  |
16534: (8)                  *Overload 6:*
16535: (9)                   Constructor using `fz_new_buffer_from_image_as_jpx()`.
16536: (8)                  |
16537: (8)                  *Overload 7:*
16538: (9)                   Constructor using `fz_new_buffer_from_image_as_png()`.
16539: (16)                         Reencode a given image as a PNG into a buffer.
16540: (16)                         Ownership of the buffer is returned.
16541: (8)                  |
16542: (8)                  *Overload 8:*
16543: (9)                   Constructor using `fz_new_buffer_from_page()`.
16544: (8)                  |
16545: (8)                  *Overload 9:*
16546: (9)                   Constructor using `fz_new_buffer_from_page_number()`.
16547: (8)                  |
16548: (8)                  *Overload 10:*
16549: (9)                   Constructor using `fz_new_buffer_from_page_with_format()`.
16550: (16)                         Returns an fz_buffer containing a page after conversion to
specified format.
16551: (16)                         page: The page to convert.
16552: (16)                         format, options: Passed to
fz_new_document_writer_with_output() internally.
16553: (16)                         transform, cookie: Passed to fz_run_page() internally.
16554: (8)                  |
16555: (8)                  *Overload 11:*
16556: (9)                   Constructor using `fz_new_buffer_from_pixmap_as_jpeg()`.
16557: (8)                  |
16558: (8)                  *Overload 12:*
16559: (9)                   Constructor using `fz_new_buffer_from_pixmap_as_jpx()`.
16560: (8)                  |
16561: (8)                  *Overload 13:*
16562: (9)                   Constructor using `fz_new_buffer_from_pixmap_as_png()`.
16563: (16)                         Reencode a given pixmap as a PNG into a buffer.
16564: (16)                         Ownership of the buffer is returned.
16565: (8)                  |
16566: (8)                  *Overload 14:*
16567: (9)                   Constructor using `fz_new_buffer_from_shared_data()`.
16568: (16)                         Like fz_new_buffer, but does not take ownership.
16569: (8)                  |
16570: (8)                  *Overload 15:*
16571: (9)                   Constructor using `fz_new_buffer_from_stext_page()`.
16572: (16)                         Convert structured text into plain text.
16573: (8)                  |
16574: (8)                  *Overload 16:*
16575: (9)                   Constructor using `fz_read_file()`.
16576: (16)                         Read all the contents of a file into a buffer.
16577: (8)                  |
16578: (8)                  *Overload 17:*
16579: (9)                   Copy constructor using `fz_keep_buffer()`.
16580: (8)                  |
16581: (8)                  *Overload 18:*
16582: (9)                   Constructor using raw copy of pre-existing `::fz_buffer`.
16583: (8)                  |
16584: (8)                  *Overload 19:*
16585: (9)                   Constructor using raw copy of pre-existing `::fz_buffer`.
16586: (8)                  """
16587: (8)                  _mupdf.FzBuffer_swiginit(self, _mupdf.new_FzBuffer(*args))
16588: (4)              __swig_destroy__ = _mupdf.delete_FzBuffer
16589: (4)              def m_internal_value(self):
16590: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
16591: (8)                  return _mupdf.FzBuffer_m_internal_value(self)
16592: (4)              m_internal = property(_mupdf.FzBuffer_m_internal_get,
_mupdf.FzBuffer_m_internal_set, doc=r""" Pointer to wrapped data.""")
16593: (4)              s_num_instances = property(_mupdf.FzBuffer_s_num_instances_get,
_mupdf.FzBuffer_s_num_instances_set)
```

```
16594: (0)          _mupdf.FzBuffer_swigregister(FzBuffer)
16595: (0)          class FzColorParams(object):
16596: (4)              r""" Wrapper class for struct `fz_color_params`."""
16597: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16598: (4)              __repr__ = _swig_repr
16599: (4)              def __init__(self, *args):
16600: (8)                  r"""
16601: (8)                  *Overload 1:*
16602: (8)                  Equivalent to fz_default_color_params.
16603: (8)                  |
16604: (8)                  *Overload 2:*
16605: (8)                  We use default copy constructor and operator=.  Constructor using raw
copy of pre-existing `::fz_color_params`.
16606: (8)                  |
16607: (8)                  *Overload 3:*
16608: (8)                  Constructor using raw copy of pre-existing `::fz_color_params`.
16609: (8)                  """
16610: (8)                  _mupdf.FzColorParams_swiginit(self, _mupdf.new_FzColorParams(*args))
16611: (4)              def internal(self, *args):
16612: (8)                  r"""
16613: (8)                  *Overload 1:*
16614: (8)                  Access as underlying struct.
16615: (8)                  |
16616: (8)                  *Overload 2:*
16617: (8)                  Access as underlying struct.
16618: (8)                  """
16619: (8)                  return _mupdf.FzColorParams_internal(self, *args)
16620: (4)              __swig_destroy__ = _mupdf.delete_FzColorParams
16621: (4)              ri = property(_mupdf.FzColorParams_ri_get, _mupdf.FzColorParams_ri_set)
16622: (4)              bp = property(_mupdf.FzColorParams_bp_get, _mupdf.FzColorParams_bp_set)
16623: (4)              op = property(_mupdf.FzColorParams_op_get, _mupdf.FzColorParams_op_set)
16624: (4)              opm = property(_mupdf.FzColorParams_opm_get, _mupdf.FzColorParams_opm_set)
16625: (4)              s_num_instances = property(_mupdf.FzColorParams_s_num_instances_get,
_mupdf.FzColorParams_s_num_instances_set)
16626: (4)              def to_string(self):
16627: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
16628: (8)                  return _mupdf.FzColorParams_to_string(self)
16629: (4)              def __eq__(self, rhs):
16630: (8)                  r""" Comparison method."""
16631: (8)                  return _mupdf.FzColorParams___eq__(self, rhs)
16632: (4)              def __ne__(self, rhs):
16633: (8)                  r""" Comparison method."""
16634: (8)                  return _mupdf.FzColorParams___ne__(self, rhs)
16635: (0)          _mupdf.FzColorParams_swigregister(FzColorParams)
16636: (0)          class FzColorspace(object):
16637: (4)              r"""
16638: (4)              Wrapper class for struct `fz_colorspace`.
16639: (4)              Describes a given colorspace.
16640: (4)              """
16641: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16642: (4)              __repr__ = _swig_repr
16643: (4)              Fixed_GRAY = _mupdf.FzColorspace_Fixed_GRAY
16644: (4)              Fixed_RGB = _mupdf.FzColorspace_Fixed_RGB
16645: (4)              Fixed_BGR = _mupdf.FzColorspace_Fixed_BGR
16646: (4)              Fixed_CMYK = _mupdf.FzColorspace_Fixed_CMYK
16647: (4)              Fixed_LAB = _mupdf.FzColorspace_Fixed_LAB
16648: (4)              def fz_base_colorspace(self):
16649: (8)                  r"""
16650: (8)                  Class-aware wrapper for `::fz_base_colorspace()`.
16651: (12)                     Get the 'base' colorspace for a colorspace.
16652: (12)                     For indexed colorspaces, this is the colorspace the index
16653: (12)                     decodes into. For all other colorspaces, it is the colorspace
16654: (12)                     itself.
16655: (12)                     The returned colorspace is 'borrowed' (i.e. no additional
16656: (12)                     references are taken or dropped).
16657: (8)                  """
```

```
(8)                 return _mupdf.FzColorspace_fz_base_colorspace(self)
(4)             def fz_clamp_color(self, _in, out):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clamp_color()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_clamp_color(const float *in)` => float out
(12)                    Clamp the samples in a color to the correct ranges for a
(12)                    given colorspace.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_clamp_color(self, _in, out)
(4)             def fz_colorspace_colorant(self, n):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_colorant()`.
(12)                    Retrieve a the name for a colorant.
(12)                    Returns a pointer with the same lifespan as the colorspace.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_colorant(self, n)
(4)             def fz_colorspace_device_n_has_cmyk(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_device_n_has_cmyk()`.
(12)                    True if DeviceN color space has cyan magenta yellow or black as
(12)                    one of its colorants.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_device_n_has_cmyk(self)
(4)             def fz_colorspace_device_n_has_only_cmyk(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_device_n_has_only_cmyk()`.
(12)                    True if DeviceN color space has only colorants from the CMYK set.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_device_n_has_only_cmyk(self)
(4)             def fz_colorspace_is_cmyk(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_cmyk()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_cmyk(self)
(4)             def fz_colorspace_is_device(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_device()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_device(self)
(4)             def fz_colorspace_is_device_cmyk(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_device_cmyk()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_device_cmyk(self)
(4)             def fz_colorspace_is_device_gray(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_device_gray()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_device_gray(self)
(4)             def fz_colorspace_is_device_n(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_device_n()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_device_n(self)
(4)             def fz_colorspace_is_gray(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_is_gray()`.
(12)                    Tests for particular types of colorspaces
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_gray(self)
(4)             def fz_colorspace_is_indexed(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_indexed()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_indexed(self)
(4)             def fz_colorspace_is_lab(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_lab()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_lab(self)
(4)             def fz_colorspace_is_lab_icc(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_lab_icc()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_lab_icc(self)
(4)             def fz_colorspace_is_rgb(self):
(8)                 r""" Class-aware wrapper for `::fz_colorspace_is_rgb()`."""
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_rgb(self)
(4)             def fz_colorspace_is_subtractive(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_is_subtractive()`.
(12)                    True for CMYK, Separation and DeviceN colorspaces.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_is_subtractive(self)
```

```
(4)             def fz_colorspace_n(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_n()`.
(12)                    Query the number of colorants in a colorspace.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_n(self)
(4)             def fz_colorspace_name(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_name()`.
(12)                    Query the name of a colorspace.
(12)                    The returned string has the same lifespan as the colorspace
(12)                    does. Caller should not free it.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_name(self)
(4)             def fz_colorspace_name_colorant(self, n, name):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_name_colorant()`.
(12)                    Assign a name for a given colorant in a colorspace.
(12)                    Used while initially setting up a colorspace. The string is
(12)                    copied into local storage, so need not be retained by the
(12)                    caller.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_name_colorant(self, n, name)
(4)             def fz_colorspace_type(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_colorspace_type()`.
(12)                    Query the type of colorspace.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_colorspace_type(self)
(4)             def fz_convert_color(self, sv, ds, dv, _is, params):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_convert_color()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_convert_color(const float *sv, ::fz_colorspace *ds,
::fz_colorspace *is, ::fz_color_params params)` => float dv
(12)                    Convert color values sv from colorspace ss into colorvalues dv
(12)                    for colorspace ds, via an optional intervening space is,
(12)                    respecting the given color_params.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_convert_color(self, sv, ds, dv, _is,
params)
(4)             def fz_convert_separation_colors(self, src_color, dst_seps, dst_cs,
dst_color, color_params):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_convert_separation_colors()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_convert_separation_colors(const float *src_color,
::fz_separations *dst_seps, ::fz_colorspace *dst_cs, ::fz_color_params color_params)` => float
dst_color
(12)                    Convert a color given in terms of one colorspace,
(12)                    to a color in terms of another colorspace/separations.
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_convert_separation_colors(self,
src_color, dst_seps, dst_cs, dst_color, color_params)
(4)             def fz_is_valid_blend_colorspace(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_valid_blend_colorspace()`.
(12)                    Check to see that a colorspace is appropriate to be used as
(12)                    a blending space (i.e. only grey, rgb or cmyk).
(8)                 """
(8)                 return _mupdf.FzColorspace_fz_is_valid_blend_colorspace(self)
(4)             def fz_new_indexed_colorspace(self, high, lookup):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_indexed_colorspace()`.
(12)                    Create an indexed colorspace.
(12)                    The supplied lookup table is high palette entries long. Each
(12)                    entry is n bytes long, where n is given by the number of
(12)                    colorants in the base colorspace, one byte per colorant.
(12)                    Ownership of lookup is passed it; it will be freed on
```

```
16790: (12)                     destruction, so must be heap allocated.
16791: (12)                     The colorspace will keep an additional reference to the base
16792: (12)                     colorspace that will be dropped on destruction.
16793: (12)                     The returned reference should be dropped when it is finished
16794: (12)                     with.
16795: (12)                     Colorspaces are immutable once created.
16796: (8)                  """
16797: (8)                  return _mupdf.FzColorspace_fz_new_indexed_colorspace(self, high,
lookup)
16798: (4)              def fz_new_pixmap(self, w, h, seps, alpha):
16799: (8)                  r"""
16800: (8)                  Class-aware wrapper for `::fz_new_pixmap()`.
16801: (12)                     Create a new pixmap, with its origin at (0,0)
16802: (12)                     cs: The colorspace to use for the pixmap, or NULL for an alpha
16803: (12)                     plane/mask.
16804: (12)                     w: The width of the pixmap (in pixels)
16805: (12)                     h: The height of the pixmap (in pixels)
16806: (12)                     seps: Details of separations.
16807: (12)                     alpha: 0 for no alpha, 1 for alpha.
16808: (12)                     Returns a pointer to the new pixmap. Throws exception on failure
16809: (12)                     to allocate.
16810: (8)                  """
16811: (8)                  return _mupdf.FzColorspace_fz_new_pixmap(self, w, h, seps, alpha)
16812: (4)              def fz_new_pixmap_with_bbox(self, bbox, seps, alpha):
16813: (8)                  r"""
16814: (8)                  Class-aware wrapper for `::fz_new_pixmap_with_bbox()`.
16815: (12)                     Create a pixmap of a given size, location and pixel format.
16816: (12)                     The bounding box specifies the size of the created pixmap and
16817: (12)                     where it will be located. The colorspace determines the number
16818: (12)                     of components per pixel. Alpha is always present. Pixmaps are
16819: (12)                     reference counted, so drop references using fz_drop_pixmap.
16820: (12)                     colorspace: Colorspace format used for the created pixmap. The
16821: (12)                     pixmap will keep a reference to the colorspace.
16822: (12)                     bbox: Bounding box specifying location/size of created pixmap.
16823: (12)                     seps: Details of separations.
16824: (12)                     alpha: 0 for no alpha, 1 for alpha.
16825: (12)                     Returns a pointer to the new pixmap. Throws exception on failure
16826: (12)                     to allocate.
16827: (8)                  """
16828: (8)                  return _mupdf.FzColorspace_fz_new_pixmap_with_bbox(self, bbox, seps,
alpha)
16829: (4)              def fz_new_pixmap_with_bbox_and_data(self, rect, seps, alpha, samples):
16830: (8)                  r"""
16831: (8)                  Class-aware wrapper for `::fz_new_pixmap_with_bbox_and_data()`.
16832: (12)                     Create a pixmap of a given size, location and pixel format,
16833: (12)                     using the supplied data block.
16834: (12)                     The bounding box specifies the size of the created pixmap and
16835: (12)                     where it will be located. The colorspace determines the number
16836: (12)                     of components per pixel. Alpha is always present. Pixmaps are
16837: (12)                     reference counted, so drop references using fz_drop_pixmap.
16838: (12)                     colorspace: Colorspace format used for the created pixmap. The
16839: (12)                     pixmap will keep a reference to the colorspace.
16840: (12)                     rect: Bounding box specifying location/size of created pixmap.
16841: (12)                     seps: Details of separations.
16842: (12)                     alpha: Number of alpha planes (0 or 1).
16843: (12)                     samples: The data block to keep the samples in.
16844: (12)                     Returns a pointer to the new pixmap. Throws exception on failure
16845: (12)                     to allocate.
16846: (8)                  """
16847: (8)                  return _mupdf.FzColorspace_fz_new_pixmap_with_bbox_and_data(self,
rect, seps, alpha, samples)
16848: (4)              def fz_new_pixmap_with_data(self, w, h, seps, alpha, stride, samples):
16849: (8)                  r"""
16850: (8)                  Class-aware wrapper for `::fz_new_pixmap_with_data()`.
16851: (12)                     Create a new pixmap, with its origin at
16852: (12)                     (0,0) using the supplied data block.
16853: (12)                     cs: The colorspace to use for the pixmap, or NULL for an alpha
16854: (12)                     plane/mask.
16855: (12)                     w: The width of the pixmap (in pixels)
```

```
16856: (12)                     h: The height of the pixmap (in pixels)
16857: (12)                     seps: Details of separations.
16858: (12)                     alpha: 0 for no alpha, 1 for alpha.
16859: (12)                     stride: The byte offset from the pixel data in a row to the
16860: (12)                     pixel data in the next row.
16861: (12)                     samples: The data block to keep the samples in.
16862: (12)                     Returns a pointer to the new pixmap. Throws exception on failure
to
16863: (12)                     allocate.
16864: (8)                  """
16865: (8)                  return _mupdf.FzColorspace_fz_new_pixmap_with_data(self, w, h, seps,
alpha, stride, samples)
16866: (4)              def pdf_is_tint_colorspace(self):
16867: (8)                  r""" Class-aware wrapper for `::pdf_is_tint_colorspace()`."""
16868: (8)                  return _mupdf.FzColorspace_pdf_is_tint_colorspace(self)
16869: (4)              def __init__(self, *args):
16870: (8)                  r"""
16871: (8)                  *Overload 1:*
16872: (9)                   == Constructors.  Constructor using `fz_new_cal_gray_colorspace()`.
16873: (16)                         Create a calibrated gray colorspace.
16874: (16)                         The returned reference should be dropped when it is finished
16875: (16)                         with.
16876: (16)                         Colorspaces are immutable once created.
16877: (8)                  |
16878: (8)                  *Overload 2:*
16879: (9)                   Constructor using `fz_new_cal_rgb_colorspace()`.
16880: (16)                         Create a calibrated rgb colorspace.
16881: (16)                         The returned reference should be dropped when it is finished
16882: (16)                         with.
16883: (16)                         Colorspaces are immutable once created.
16884: (8)                  |
16885: (8)                  *Overload 3:*
16886: (9)                   Constructor using `fz_new_colorspace()`.
16887: (16)                         Creates a new colorspace instance and returns a reference.
16888: (16)                         No internal checking is done that the colorspace type (e.g.
16889: (16)                         CMYK) matches with the flags (e.g. FZ_COLORSPACE_HAS_CMYK) or
16890: (16)                         colorant count (n) or name.
16891: (16)                         The reference should be dropped when it is finished with.
16892: (16)                         Colorspaces are immutable once created (with the exception of
16893: (16)                         setting up colorant names for separation spaces).
16894: (8)                  |
16895: (8)                  *Overload 4:*
16896: (9)                   Constructor using `fz_new_icc_colorspace()`.
16897: (16)                         Create a colorspace from an ICC profile supplied in buf.
16898: (16)                         Limited checking is done to ensure that the colorspace type is
16899: (16)                         appropriate for the supplied ICC profile.
16900: (16)                         An additional reference is taken to buf, which will be dropped
16901: (16)                         on destruction. Ownership is NOT passed in.
16902: (16)                         The returned reference should be dropped when it is finished
16903: (16)                         with.
16904: (16)                         Colorspaces are immutable once created.
16905: (8)                  |
16906: (8)                  *Overload 5:*
16907: (9)                   Constructor using `fz_new_indexed_colorspace()`.
16908: (16)                         Create an indexed colorspace.
16909: (16)                         The supplied lookup table is high palette entries long. Each
16910: (16)                         entry is n bytes long, where n is given by the number of
16911: (16)                         colorants in the base colorspace, one byte per colorant.
16912: (16)                         Ownership of lookup is passed it; it will be freed on
16913: (16)                         destruction, so must be heap allocated.
16914: (16)                         The colorspace will keep an additional reference to the base
16915: (16)                         colorspace that will be dropped on destruction.
16916: (16)                         The returned reference should be dropped when it is finished
16917: (16)                         with.
16918: (16)                         Colorspaces are immutable once created.
16919: (8)                  |
16920: (8)                  *Overload 6:*
16921: (9)                   Construct using one of: fz_device_gray(), fz_device_rgb(),
fz_device_bgr(), fz_device_cmyk(), fz_device_lab().
```

```
16922: (8)                  |
16923: (8)                  *Overload 7:*
16924: (9)                   Copy constructor using `fz_keep_colorspace()`.
16925: (8)                  |
16926: (8)                  *Overload 8:*
16927: (9)                   Default constructor, sets `m_internal` to null.
16928: (8)                  |
16929: (8)                  *Overload 9:*
16930: (9)                   Constructor using raw copy of pre-existing `::fz_colorspace`.
16931: (8)                  """
16932: (8)                  _mupdf.FzColorspace_swiginit(self, _mupdf.new_FzColorspace(*args))
16933: (4)              __swig_destroy__ = _mupdf.delete_FzColorspace
16934: (4)              def m_internal_value(self):
16935: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
16936: (8)                  return _mupdf.FzColorspace_m_internal_value(self)
16937: (4)              m_internal = property(_mupdf.FzColorspace_m_internal_get,
_mupdf.FzColorspace_m_internal_set, doc=r""" Pointer to wrapped data.""")
16938: (4)              s_num_instances = property(_mupdf.FzColorspace_s_num_instances_get,
_mupdf.FzColorspace_s_num_instances_set)
16939: (0)          _mupdf.FzColorspace_swigregister(FzColorspace)
16940: (0)          class FzColorspaceContext(object):
16941: (4)              r""" Wrapper class for struct `fz_colorspace_context`. Not copyable or
assignable."""
16942: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16943: (4)              __repr__ = _swig_repr
16944: (4)              def __init__(self, *args):
16945: (8)                  r"""
16946: (8)                  *Overload 1:*
16947: (8)                  Default constructor, sets `m_internal` to null.
16948: (8)                  |
16949: (8)                  *Overload 2:*
16950: (8)                  Constructor using raw copy of pre-existing `::fz_colorspace_context`.
16951: (8)                  """
16952: (8)                  _mupdf.FzColorspaceContext_swiginit(self,
_mupdf.new_FzColorspaceContext(*args))
16953: (4)              __swig_destroy__ = _mupdf.delete_FzColorspaceContext
16954: (4)              def m_internal_value(self):
16955: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
16956: (8)                  return _mupdf.FzColorspaceContext_m_internal_value(self)
16957: (4)              m_internal = property(_mupdf.FzColorspaceContext_m_internal_get,
_mupdf.FzColorspaceContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
16958: (4)              s_num_instances = property(_mupdf.FzColorspaceContext_s_num_instances_get,
_mupdf.FzColorspaceContext_s_num_instances_set)
16959: (0)          _mupdf.FzColorspaceContext_swigregister(FzColorspaceContext)
16960: (0)          class FzCompressedBuffer(object):
16961: (4)              r"""
16962: (4)              Wrapper class for struct `fz_compressed_buffer`.
16963: (4)              Buffers of compressed data; typically for the source data
16964: (4)              for images.
16965: (4)              """
16966: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
16967: (4)              __repr__ = _swig_repr
16968: (4)              def fz_compressed_buffer_size(self):
16969: (8)                  r"""
16970: (8)                  Class-aware wrapper for `::fz_compressed_buffer_size()`.
16971: (12)                     Return the storage size used for a buffer and its data.
16972: (12)                     Used in implementing store handling.
16973: (12)                     Never throws exceptions.
16974: (8)                  """
16975: (8)                  return _mupdf.FzCompressedBuffer_fz_compressed_buffer_size(self)
16976: (4)              def fz_open_compressed_buffer(self):
16977: (8)                  r"""
16978: (8)                  Class-aware wrapper for `::fz_open_compressed_buffer()`.
16979: (12)                     Open a stream to read the decompressed version of a buffer.
16980: (8)                  """
```

```
16981: (8)                  return _mupdf.FzCompressedBuffer_fz_open_compressed_buffer(self)
16982: (4)              def fz_open_image_decomp_stream_from_buffer(self, l2factor):
16983: (8)                  r"""
16984: (8)                  Class-aware wrapper for `::fz_open_image_decomp_stream_from_buffer()`.
16985: (8)                  This method has out-params. Python/C# wrappers look like:
16986: (12)                     `fz_open_image_decomp_stream_from_buffer()` => `(fz_stream *, int
l2factor)`
16987: (12)                     Open a stream to read the decompressed version of a buffer,
16988: (12)                     with optional log2 subsampling.
16989: (12)                     l2factor = NULL for no subsampling, or a pointer to an integer
16990: (12)                     containing the maximum log2 subsample factor acceptable (0 =
16991: (12)                     none, 1 = halve dimensions, 2 = quarter dimensions etc). If
16992: (12)                     non-NULL, then *l2factor will be updated on exit with the actual
16993: (12)                     log2 subsample factor achieved.
16994: (8)                  """
16995: (8)                  return
_mupdf.FzCompressedBuffer_fz_open_image_decomp_stream_from_buffer(self, l2factor)
16996: (4)              def get_buffer(self):
16997: (8)                  r""" Returns wrapper class for fz_buffer *m_internal.buffer."""
16998: (8)                  return _mupdf.FzCompressedBuffer_get_buffer(self)
16999: (4)              def __init__(self, *args):
17000: (8)                  r"""
17001: (8)                  *Overload 1:*
17002: (9)                   == Constructors.  Constructor using `fz_new_compressed_buffer()`.
17003: (16)                         Create a new, UNKNOWN format, compressed_buffer.
17004: (8)                  |
17005: (8)                  *Overload 2:*
17006: (9)                   Copy constructor using `fz_keep_compressed_buffer()`.
17007: (8)                  |
17008: (8)                  *Overload 3:*
17009: (9)                   Constructor using raw copy of pre-existing `::fz_compressed_buffer`.
17010: (8)                  """
17011: (8)                  _mupdf.FzCompressedBuffer_swiginit(self,
_mupdf.new_FzCompressedBuffer(*args))
17012: (4)              __swig_destroy__ = _mupdf.delete_FzCompressedBuffer
17013: (4)              def m_internal_value(self):
17014: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17015: (8)                  return _mupdf.FzCompressedBuffer_m_internal_value(self)
17016: (4)              m_internal = property(_mupdf.FzCompressedBuffer_m_internal_get,
_mupdf.FzCompressedBuffer_m_internal_set, doc=r""" Pointer to wrapped data.""")
17017: (4)              s_num_instances = property(_mupdf.FzCompressedBuffer_s_num_instances_get,
_mupdf.FzCompressedBuffer_s_num_instances_set)
17018: (0)          _mupdf.FzCompressedBuffer_swigregister(FzCompressedBuffer)
17019: (0)          class FzCompressedImage(object):
17020: (4)              r""" Wrapper class for struct `fz_compressed_image`. Not copyable or
assignable."""
17021: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17022: (4)              __repr__ = _swig_repr
17023: (4)              def fz_set_compressed_image_buffer(self, buf):
17024: (8)                  r""" Class-aware wrapper for `::fz_set_compressed_image_buffer()`."""
17025: (8)                  return _mupdf.FzCompressedImage_fz_set_compressed_image_buffer(self,
buf)
17026: (4)              def __init__(self, *args):
17027: (8)                  r"""
17028: (8)                  *Overload 1:*
17029: (8)                  Default constructor, sets `m_internal` to null.
17030: (8)                  |
17031: (8)                  *Overload 2:*
17032: (8)                  Constructor using raw copy of pre-existing `::fz_compressed_image`.
17033: (8)                  """
17034: (8)                  _mupdf.FzCompressedImage_swiginit(self,
_mupdf.new_FzCompressedImage(*args))
17035: (4)              __swig_destroy__ = _mupdf.delete_FzCompressedImage
17036: (4)              def m_internal_value(self):
17037: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17038: (8)                  return _mupdf.FzCompressedImage_m_internal_value(self)
```

```
17039: (4)              m_internal = property(_mupdf.FzCompressedImage_m_internal_get,
_mupdf.FzCompressedImage_m_internal_set, doc=r""" Pointer to wrapped data.""")
17040: (4)              s_num_instances = property(_mupdf.FzCompressedImage_s_num_instances_get,
_mupdf.FzCompressedImage_s_num_instances_set)
17041: (0)          _mupdf.FzCompressedImage_swigregister(FzCompressedImage)
17042: (0)          class FzCompressionParams(object):
17043: (4)              r"""
17044: (4)              Wrapper class for struct `fz_compression_params`. Not copyable or
assignable.
17045: (4)              Compression parameters used for buffers of compressed data;
17046: (4)              typically for the source data for images.
17047: (4)              """
17048: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17049: (4)              __repr__ = _swig_repr
17050: (4)              def __init__(self, *args):
17051: (8)                  r"""
17052: (8)                  *Overload 1:*
17053: (8)                  Default constructor, sets `m_internal` to null.
17054: (8)                  |
17055: (8)                  *Overload 2:*
17056: (8)                  Constructor using raw copy of pre-existing `::fz_compression_params`.
17057: (8)                  """
17058: (8)                  _mupdf.FzCompressionParams_swiginit(self,
_mupdf.new_FzCompressionParams(*args))
17059: (4)              __swig_destroy__ = _mupdf.delete_FzCompressionParams
17060: (4)              def m_internal_value(self):
17061: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17062: (8)                  return _mupdf.FzCompressionParams_m_internal_value(self)
17063: (4)              m_internal = property(_mupdf.FzCompressionParams_m_internal_get,
_mupdf.FzCompressionParams_m_internal_set, doc=r""" Pointer to wrapped data.""")
17064: (4)              s_num_instances = property(_mupdf.FzCompressionParams_s_num_instances_get,
_mupdf.FzCompressionParams_s_num_instances_set)
17065: (0)          _mupdf.FzCompressionParams_swigregister(FzCompressionParams)
17066: (0)          class FzContext(object):
17067: (4)              r""" Wrapper class for struct `fz_context`. Not copyable or assignable."""
17068: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17069: (4)              __repr__ = _swig_repr
17070: (4)              def __init__(self, *args):
17071: (8)                  r"""
17072: (8)                  *Overload 1:*
17073: (8)                  == Constructors.  Constructor using `fz_new_context_imp()`.
17074: (8)                  |
17075: (8)                  *Overload 2:*
17076: (8)                  Default constructor, sets `m_internal` to null.
17077: (8)                  |
17078: (8)                  *Overload 3:*
17079: (8)                  Constructor using raw copy of pre-existing `::fz_context`.
17080: (8)                  """
17081: (8)                  _mupdf.FzContext_swiginit(self, _mupdf.new_FzContext(*args))
17082: (4)              __swig_destroy__ = _mupdf.delete_FzContext
17083: (4)              def m_internal_value(self):
17084: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17085: (8)                  return _mupdf.FzContext_m_internal_value(self)
17086: (4)              m_internal = property(_mupdf.FzContext_m_internal_get,
_mupdf.FzContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
17087: (4)              s_num_instances = property(_mupdf.FzContext_s_num_instances_get,
_mupdf.FzContext_s_num_instances_set)
17088: (0)          _mupdf.FzContext_swigregister(FzContext)
17089: (0)          class FzCookie(object):
17090: (4)              r"""
17091: (4)              Wrapper class for struct `fz_cookie`. Not copyable or assignable.
17092: (4)              Provide two-way communication between application and library.
17093: (4)              Intended for multi-threaded applications where one thread is
17094: (4)              rendering pages and another thread wants to read progress
17095: (4)              feedback or abort a job that takes a long time to finish. The
```

```
17096: (4)              communication is unsynchronized without locking.
17097: (4)              abort: The application should set this field to 0 before
17098: (4)              calling fz_run_page to render a page. At any point when the
17099: (4)              page is being rendered the application my set this field to 1
17100: (4)              which will cause the rendering to finish soon. This field is
17101: (4)              checked periodically when the page is rendered, but exactly
17102: (4)              when is not known, therefore there is no upper bound on
17103: (4)              exactly when the rendering will abort. If the application
17104: (4)              did not provide a set of locks to fz_new_context, it must also
17105: (4)              await the completion of fz_run_page before issuing another
17106: (4)              call to fz_run_page. Note that once the application has set
17107: (4)              this field to 1 after it called fz_run_page it may not change
17108: (4)              the value again.
17109: (4)              progress: Communicates rendering progress back to the
17110: (4)              application and is read only. Increments as a page is being
17111: (4)              rendered. The value starts out at 0 and is limited to less
17112: (4)              than or equal to progress_max, unless progress_max is -1.
17113: (4)              progress_max: Communicates the known upper bound of rendering
17114: (4)              back to the application and is read only. The maximum value
17115: (4)              that the progress field may take. If there is no known upper
17116: (4)              bound on how long the rendering may take this value is -1 and
17117: (4)              progress is not limited. Note that the value of progress_max
17118: (4)              may change from -1 to a positive value once an upper bound is
17119: (4)              known, so take this into consideration when comparing the
17120: (4)              value of progress to that of progress_max.
17121: (4)              errors: count of errors during current rendering.
17122: (4)              incomplete: Initially should be set to 0. Will be set to
17123: (4)              non-zero if a TRYLATER error is thrown during rendering.
17124: (4)              """
17125: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17126: (4)              __repr__ = _swig_repr
17127: (4)              def __init__(self):
17128: (8)                  r""" Default constructor sets all fields to default values."""
17129: (8)                  _mupdf.FzCookie_swiginit(self, _mupdf.new_FzCookie())
17130: (4)              def set_abort(self):
17131: (8)                  r""" Sets m_internal.abort to 1."""
17132: (8)                  return _mupdf.FzCookie_set_abort(self)
17133: (4)              def increment_errors(self, delta):
17134: (8)                  r""" Increments m_internal.errors by <delta>."""
17135: (8)                  return _mupdf.FzCookie_increment_errors(self, delta)
17136: (4)              def abort(self):
17137: (8)                  return _mupdf.FzCookie_abort(self)
17138: (4)              def progress(self):
17139: (8)                  return _mupdf.FzCookie_progress(self)
17140: (4)              def progress_max(self):
17141: (8)                  return _mupdf.FzCookie_progress_max(self)
17142: (4)              def errors(self):
17143: (8)                  return _mupdf.FzCookie_errors(self)
17144: (4)              def incomplete(self):
17145: (8)                  return _mupdf.FzCookie_incomplete(self)
17146: (4)              __swig_destroy__ = _mupdf.delete_FzCookie
17147: (4)              m_internal = property(_mupdf.FzCookie_m_internal_get,
_mupdf.FzCookie_m_internal_set)
17148: (4)              s_num_instances = property(_mupdf.FzCookie_s_num_instances_get,
_mupdf.FzCookie_s_num_instances_set, doc=r""" Wrapped data is held by value.""")
17149: (4)              def to_string(self):
17150: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
17151: (8)                  return _mupdf.FzCookie_to_string(self)
17152: (4)              def __eq__(self, rhs):
17153: (8)                  r""" Comparison method."""
17154: (8)                  return _mupdf.FzCookie___eq__(self, rhs)
17155: (4)              def __ne__(self, rhs):
17156: (8)                  r""" Comparison method."""
17157: (8)                  return _mupdf.FzCookie___ne__(self, rhs)
17158: (0)          _mupdf.FzCookie_swigregister(FzCookie)
17159: (0)          class FzDefaultColorspaces(object):
17160: (4)              r"""
```

```
(4)             Wrapper class for struct `fz_default_colorspaces`.
(4)             Structure to hold default colorspaces.
(4)             """
(4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v), doc="The membership flag")
(4)             __repr__ = _swig_repr
(4)             def fz_clone_default_colorspaces(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clone_default_colorspaces()`.
(12)                    Returns a reference to a newly cloned default colorspaces
(12)                    structure.
(12)                    The new clone may safely be altered without fear of race
(12)                    conditions as the caller is the only reference holder.
(8)                 """
(8)                 return _mupdf.FzDefaultColorspaces_fz_clone_default_colorspaces(self)
(4)             def fz_default_cmyk(self):
(8)                 r""" Class-aware wrapper for `::fz_default_cmyk()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_default_cmyk(self)
(4)             def fz_default_gray(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_default_gray()`.
(12)                    Retrieve default colorspaces (typically page local).
(12)                    If default_cs is non NULL, the default is retrieved from there,
(12)                    otherwise the global default is retrieved.
(12)                    These return borrowed references that should not be dropped,
(12)                    unless they are kept first.
(8)                 """
(8)                 return _mupdf.FzDefaultColorspaces_fz_default_gray(self)
(4)             def fz_default_output_intent(self):
(8)                 r""" Class-aware wrapper for `::fz_default_output_intent()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_default_output_intent(self)
(4)             def fz_default_rgb(self):
(8)                 r""" Class-aware wrapper for `::fz_default_rgb()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_default_rgb(self)
(4)             def fz_set_default_cmyk(self, cs):
(8)                 r""" Class-aware wrapper for `::fz_set_default_cmyk()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_set_default_cmyk(self, cs)
(4)             def fz_set_default_gray(self, cs):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_set_default_gray()`.
(12)                    Set new defaults within the default colorspace structure.
(12)                    New references are taken to the new default, and references to
(12)                    the old defaults dropped.
(12)                    Never throws exceptions.
(8)                 """
(8)                 return _mupdf.FzDefaultColorspaces_fz_set_default_gray(self, cs)
(4)             def fz_set_default_output_intent(self, cs):
(8)                 r""" Class-aware wrapper for `::fz_set_default_output_intent()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_set_default_output_intent(self, cs)
(4)             def fz_set_default_rgb(self, cs):
(8)                 r""" Class-aware wrapper for `::fz_set_default_rgb()`."""
(8)                 return _mupdf.FzDefaultColorspaces_fz_set_default_rgb(self, cs)
(4)             def pdf_update_default_colorspaces(self, res):
(8)                 r""" Class-aware wrapper for `::pdf_update_default_colorspaces()`."""
(8)                 return _mupdf.FzDefaultColorspaces_pdf_update_default_colorspaces(self, res)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using `fz_new_default_colorspaces()`.
(16)                        Create a new default colorspace structure with values inherited
(16)                        from the context, and return a reference to it.
(16)                        These can be overridden using fz_set_default_xxxx.
(16)                        These should not be overridden while more than one caller has
(16)                        the reference for fear of race conditions.
(16)                        The caller should drop this reference once finished with it.
(8)                 |
```

```
17226: (8)                  *Overload 2:*
17227: (9)                   Copy constructor using `fz_keep_default_colorspaces()`.
17228: (8)                  |
17229: (8)                  *Overload 3:*
17230: (9)                   Constructor using raw copy of pre-existing
`::fz_default_colorspaces`.
17231: (8)                  """
17232: (8)                  _mupdf.FzDefaultColorspaces_swiginit(self,
_mupdf.new_FzDefaultColorspaces(*args))
17233: (4)              __swig_destroy__ = _mupdf.delete_FzDefaultColorspaces
17234: (4)              def m_internal_value(self):
17235: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17236: (8)                  return _mupdf.FzDefaultColorspaces_m_internal_value(self)
17237: (4)              m_internal = property(_mupdf.FzDefaultColorspaces_m_internal_get,
_mupdf.FzDefaultColorspaces_m_internal_set, doc=r""" Pointer to wrapped data.""")
17238: (4)              s_num_instances =
property(_mupdf.FzDefaultColorspaces_s_num_instances_get,
_mupdf.FzDefaultColorspaces_s_num_instances_set)
17239: (0)          _mupdf.FzDefaultColorspaces_swigregister(FzDefaultColorspaces)
17240: (0)          class FzDevice(object):
17241: (4)              r"""
17242: (4)              Wrapper class for struct `fz_device`.
17243: (4)              The different format handlers (pdf, xps etc) interpret pages to
17244: (4)              a device. These devices can then process the stream of calls
17245: (4)              they receive in various ways:
17246: (8)                  The trace device outputs debugging information for the calls.
17247: (8)                  The draw device will render them.
17248: (8)                  The list device stores them in a list to play back later.
17249: (8)                  The text device performs text extraction and searching.
17250: (8)                  The bbox device calculates the bounding box for the page.
17251: (4)              Other devices can (and will) be written in the future.
17252: (4)              """
17253: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17254: (4)              __repr__ = _swig_repr
17255: (4)              @staticmethod
17256: (4)              def fz_new_xmltext_device(out):
17257: (8)                  r"""
17258: (8)                  Class-aware wrapper for `::fz_new_xmltext_device()`.
17259: (12)                     Create a device to output raw information.
17260: (8)                  """
17261: (8)                  return _mupdf.FzDevice_fz_new_xmltext_device(out)
17262: (4)              @staticmethod
17263: (4)              def fz_new_draw_device_type3(transform, dest):
17264: (8)                  r""" Class-aware wrapper for `::fz_new_draw_device_type3()`."""
17265: (8)                  return _mupdf.FzDevice_fz_new_draw_device_type3(transform, dest)
17266: (4)              def fz_begin_group(self, area, cs, isolated, knockout, blendmode, alpha):
17267: (8)                  r""" Class-aware wrapper for `::fz_begin_group()`."""
17268: (8)                  return _mupdf.FzDevice_fz_begin_group(self, area, cs, isolated,
knockout, blendmode, alpha)
17269: (4)              def fz_begin_layer(self, layer_name):
17270: (8)                  r""" Class-aware wrapper for `::fz_begin_layer()`."""
17271: (8)                  return _mupdf.FzDevice_fz_begin_layer(self, layer_name)
17272: (4)              def fz_begin_mask(self, area, luminosity, colorspace, bc, color_params):
17273: (8)                  r""" Class-aware wrapper for `::fz_begin_mask()`."""
17274: (8)                  return _mupdf.FzDevice_fz_begin_mask(self, area, luminosity,
colorspace, bc, color_params)
17275: (4)              def fz_begin_metatext(self, meta, text):
17276: (8)                  r""" Class-aware wrapper for `::fz_begin_metatext()`."""
17277: (8)                  return _mupdf.FzDevice_fz_begin_metatext(self, meta, text)
17278: (4)              def fz_begin_structure(self, standard, raw, idx):
17279: (8)                  r""" Class-aware wrapper for `::fz_begin_structure()`."""
17280: (8)                  return _mupdf.FzDevice_fz_begin_structure(self, standard, raw, idx)
17281: (4)              def fz_begin_tile(self, area, view, xstep, ystep, ctm):
17282: (8)                  r""" Class-aware wrapper for `::fz_begin_tile()`."""
17283: (8)                  return _mupdf.FzDevice_fz_begin_tile(self, area, view, xstep, ystep,
ctm)
17284: (4)              def fz_begin_tile_id(self, area, view, xstep, ystep, ctm, id):
```

```
(8)                 r""" Class-aware wrapper for `::fz_begin_tile_id()`."""
(8)                 return _mupdf.FzDevice_fz_begin_tile_id(self, area, view, xstep,
ystep, ctm, id)
(4)             def fz_clip_image_mask(self, image, ctm, scissor):
(8)                 r""" Class-aware wrapper for `::fz_clip_image_mask()`."""
(8)                 return _mupdf.FzDevice_fz_clip_image_mask(self, image, ctm, scissor)
(4)             def fz_clip_path(self, path, even_odd, ctm, scissor):
(8)                 r""" Class-aware wrapper for `::fz_clip_path()`."""
(8)                 return _mupdf.FzDevice_fz_clip_path(self, path, even_odd, ctm,
scissor)
(4)             def fz_clip_stroke_path(self, path, stroke, ctm, scissor):
(8)                 r""" Class-aware wrapper for `::fz_clip_stroke_path()`."""
(8)                 return _mupdf.FzDevice_fz_clip_stroke_path(self, path, stroke, ctm,
scissor)
(4)             def fz_clip_stroke_text(self, text, stroke, ctm, scissor):
(8)                 r""" Class-aware wrapper for `::fz_clip_stroke_text()`."""
(8)                 return _mupdf.FzDevice_fz_clip_stroke_text(self, text, stroke, ctm,
scissor)
(4)             def fz_clip_text(self, text, ctm, scissor):
(8)                 r""" Class-aware wrapper for `::fz_clip_text()`."""
(8)                 return _mupdf.FzDevice_fz_clip_text(self, text, ctm, scissor)
(4)             def fz_close_device(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_close_device()`.
(12)                    Signal the end of input, and flush any buffered output.
(12)                    This is NOT called implicitly on fz_drop_device. This
(12)                    may throw exceptions.
(8)                 """
(8)                 return _mupdf.FzDevice_fz_close_device(self)
(4)             def fz_device_current_scissor(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_device_current_scissor()`.
(12)                    Find current scissor region as tracked by the device.
(8)                 """
(8)                 return _mupdf.FzDevice_fz_device_current_scissor(self)
(4)             def fz_disable_device_hints(self, hints):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_disable_device_hints()`.
(12)                    Disable (clear) hint bits within the hint bitfield for a device.
(8)                 """
(8)                 return _mupdf.FzDevice_fz_disable_device_hints(self, hints)
(4)             def fz_enable_device_hints(self, hints):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_enable_device_hints()`.
(12)                    Enable (set) hint bits within the hint bitfield for a device.
(8)                 """
(8)                 return _mupdf.FzDevice_fz_enable_device_hints(self, hints)
(4)             def fz_end_group(self):
(8)                 r""" Class-aware wrapper for `::fz_end_group()`."""
(8)                 return _mupdf.FzDevice_fz_end_group(self)
(4)             def fz_end_layer(self):
(8)                 r""" Class-aware wrapper for `::fz_end_layer()`."""
(8)                 return _mupdf.FzDevice_fz_end_layer(self)
(4)             def fz_end_mask(self):
(8)                 r""" Class-aware wrapper for `::fz_end_mask()`."""
(8)                 return _mupdf.FzDevice_fz_end_mask(self)
(4)             def fz_end_mask_tr(self, fn):
(8)                 r""" Class-aware wrapper for `::fz_end_mask_tr()`."""
(8)                 return _mupdf.FzDevice_fz_end_mask_tr(self, fn)
(4)             def fz_end_metatext(self):
(8)                 r""" Class-aware wrapper for `::fz_end_metatext()`."""
(8)                 return _mupdf.FzDevice_fz_end_metatext(self)
(4)             def fz_end_structure(self):
(8)                 r""" Class-aware wrapper for `::fz_end_structure()`."""
(8)                 return _mupdf.FzDevice_fz_end_structure(self)
(4)             def fz_end_tile(self):
(8)                 r""" Class-aware wrapper for `::fz_end_tile()`."""
(8)                 return _mupdf.FzDevice_fz_end_tile(self)
(4)             def fz_fill_image(self, image, ctm, alpha, color_params):
```

```
17350: (8)                  r""" Class-aware wrapper for `::fz_fill_image()`."""
17351: (8)                  return _mupdf.FzDevice_fz_fill_image(self, image, ctm, alpha,
color_params)
17352: (4)              def fz_fill_image_mask(self, image, ctm, colorspace, color, alpha,
color_params):
17353: (8)                  r""" Class-aware wrapper for `::fz_fill_image_mask()`."""
17354: (8)                  return _mupdf.FzDevice_fz_fill_image_mask(self, image, ctm,
colorspace, color, alpha, color_params)
17355: (4)              def fz_fill_path(self, path, even_odd, ctm, colorspace, color, alpha,
color_params):
17356: (8)                  r"""
17357: (8)                  Class-aware wrapper for `::fz_fill_path()`.
17358: (12)                     Device calls; graphics primitives and containers.
17359: (8)                  """
17360: (8)                  return _mupdf.FzDevice_fz_fill_path(self, path, even_odd, ctm,
colorspace, color, alpha, color_params)
17361: (4)              def fz_fill_shade(self, shade, ctm, alpha, color_params):
17362: (8)                  r""" Class-aware wrapper for `::fz_fill_shade()`."""
17363: (8)                  return _mupdf.FzDevice_fz_fill_shade(self, shade, ctm, alpha,
color_params)
17364: (4)              def fz_fill_text(self, text, ctm, colorspace, color, alpha, color_params):
17365: (8)                  r""" Class-aware wrapper for `::fz_fill_text()`."""
17366: (8)                  return _mupdf.FzDevice_fz_fill_text(self, text, ctm, colorspace,
color, alpha, color_params)
17367: (4)              def fz_ignore_text(self, text, ctm):
17368: (8)                  r""" Class-aware wrapper for `::fz_ignore_text()`."""
17369: (8)                  return _mupdf.FzDevice_fz_ignore_text(self, text, ctm)
17370: (4)              def fz_new_ocr_device(self, ctm, mediabox, with_list, language, datadir,
progress, progress_arg):
17371: (8)                  r"""
17372: (8)                  Class-aware wrapper for `::fz_new_ocr_device()`.
17373: (12)                     Create a device to OCR the text on the page.
17374: (12)                     Renders the page internally to a bitmap that is then OCRd. Text
17375: (12)                     is then forwarded onto the target device.
17376: (12)                     target: The target device to receive the OCRd text.
17377: (12)                     ctm: The transform to apply to the mediabox to get the size for
17378: (12)                     the rendered page image. Also used to calculate the resolution
17379: (12)                     for the page image. In general, this will be the same as the CTM
17380: (12)                     that you pass to fz_run_page (or fz_run_display_list) to feed
17381: (12)                     this device.
17382: (12)                     mediabox: The mediabox (in points). Combined with the CTM to get
17383: (12)                     the bounds of the pixmap used internally for the rendered page
17384: (12)                     image.
17385: (12)                     with_list: If with_list is false, then all non-text operations
17386: (12)                     are forwarded instantly to the target device. This results in
17387: (12)                     the target device seeing all NON-text operations, followed by
17388: (12)                     all the text operations (derived from OCR).
17389: (12)                     If with_list is true, then all the marking operations are
17390: (12)                     collated into a display list which is then replayed to the
17391: (12)                     target device at the end.
17392: (12)                     language: NULL (for "eng"), or a pointer to a string to describe
17393: (12)                     the languages/scripts that should be used for OCR (e.g.
17394: (12)                     "eng,ara").
17395: (12)                     datadir: NULL (for ""), or a pointer to a path string otherwise
17396: (12)                     provided to Tesseract in the TESSDATA_PREFIX environment variable.
17397: (12)                     progress: NULL, or function to be called periodically to indicate
17398: (12)                     progress. Return 0 to continue, or 1 to cancel. progress_arg is
17399: (12)                     returned as the void *. The int is a value between 0 and 100 to
17400: (12)                     indicate progress.
17401: (12)                     progress_arg: A void * value to be parrotted back to the progress
17402: (12)                     function.
17403: (8)                  """
17404: (8)                  return _mupdf.FzDevice_fz_new_ocr_device(self, ctm, mediabox,
with_list, language, datadir, progress, progress_arg)
17405: (4)              def fz_pop_clip(self):
17406: (8)                  r""" Class-aware wrapper for `::fz_pop_clip()`."""
17407: (8)                  return _mupdf.FzDevice_fz_pop_clip(self)
17408: (4)              def fz_render_flags(self, set, clear):
17409: (8)                  r""" Class-aware wrapper for `::fz_render_flags()`."""
```

```
17410: (8)                  return _mupdf.FzDevice_fz_render_flags(self, set, clear)
17411: (4)              def fz_render_t3_glyph_direct(self, font, gid, trm, gstate, def_cs):
17412: (8)                  r"""
17413: (8)                  Class-aware wrapper for `::fz_render_t3_glyph_direct()`.
17414: (12)                     Nasty PDF interpreter specific hernia, required to allow the
17415: (12)                     interpreter to replay glyphs from a type3 font directly into
17416: (12)                     the target device.
17417: (12)                     This is only used in exceptional circumstances (such as type3
17418: (12)                     glyphs that inherit current graphics state, or nested type3
17419: (12)                     glyphs).
17420: (8)                  """
17421: (8)                  return _mupdf.FzDevice_fz_render_t3_glyph_direct(self, font, gid, trm,
gstate, def_cs)
17422: (4)              def fz_set_default_colorspaces(self, default_cs):
17423: (8)                  r""" Class-aware wrapper for `::fz_set_default_colorspaces()`."""
17424: (8)                  return _mupdf.FzDevice_fz_set_default_colorspaces(self, default_cs)
17425: (4)              def fz_stroke_path(self, path, stroke, ctm, colorspace, color, alpha,
color_params):
17426: (8)                  r""" Class-aware wrapper for `::fz_stroke_path()`."""
17427: (8)                  return _mupdf.FzDevice_fz_stroke_path(self, path, stroke, ctm,
colorspace, color, alpha, color_params)
17428: (4)              def fz_stroke_text(self, text, stroke, ctm, colorspace, color, alpha,
color_params):
17429: (8)                  r""" Class-aware wrapper for `::fz_stroke_text()`."""
17430: (8)                  return _mupdf.FzDevice_fz_stroke_text(self, text, stroke, ctm,
colorspace, color, alpha, color_params)
17431: (4)              def __init__(self, *args):
17432: (8)                  r"""
17433: (8)                  *Overload 1:*
17434: (9)                   == Constructors.  Constructor using `fz_new_bbox_device()`.
17435: (16)                         Create a device to compute the bounding
17436: (16)                         box of all marks on a page.
17437: (16)                         The returned bounding box will be the union of all bounding
17438: (16)                         boxes of all objects on a page.
17439: (8)                  |
17440: (8)                  *Overload 2:*
17441: (9)                   Constructor using `fz_new_device_of_size()`.
17442: (16)                         Devices are created by calls to device implementations, for
17443: (16)                         instance: foo_new_device(). These will be implemented by
calling
17444: (16)                         fz_new_derived_device(ctx, foo_device) where foo_device is a
17445: (16)                         structure "derived from" fz_device, for instance
17446: (16)                         typedef struct { fz_device base;  ...extras...} foo_device;
17447: (8)                  |
17448: (8)                  *Overload 3:*
17449: (9)                   Constructor using `fz_new_draw_device()`.
17450: (16)                         Create a device to draw on a pixmap.
17451: (16)                         dest: Target pixmap for the draw device. See fz_new_pixmap*
17452: (16)                         for how to obtain a pixmap. The pixmap is not cleared by the
17453: (16)                         draw device, see fz_clear_pixmap* for how to clear it prior to
17454: (16)                         calling fz_new_draw_device. Free the device by calling
17455: (16)                         fz_drop_device.
17456: (16)                         transform: Transform from user space in points to device space
17457: (16)                         in pixels.
17458: (8)                  |
17459: (8)                  *Overload 4:*
17460: (9)                   Constructor using `fz_new_draw_device_with_bbox()`.
17461: (16)                         Create a device to draw on a pixmap.
17462: (16)                         dest: Target pixmap for the draw device. See fz_new_pixmap*
17463: (16)                         for how to obtain a pixmap. The pixmap is not cleared by the
17464: (16)                         draw device, see fz_clear_pixmap* for how to clear it prior to
17465: (16)                         calling fz_new_draw_device. Free the device by calling
17466: (16)                         fz_drop_device.
17467: (16)                         transform: Transform from user space in points to device space
17468: (16)                         in pixels.
17469: (16)                         clip: Bounding box to restrict any marking operations of the
17470: (16)                         draw device.
17471: (8)                  |
17472: (8)                  *Overload 5:*
```

```
(9)                 Constructor using `fz_new_draw_device_with_bbox_proof()`.
(16)                        Create a device to draw on a pixmap.
(16)                        dest: Target pixmap for the draw device. See fz_new_pixmap*
(16)                        for how to obtain a pixmap. The pixmap is not cleared by the
(16)                        draw device, see fz_clear_pixmap* for how to clear it prior to
(16)                        calling fz_new_draw_device. Free the device by calling
(16)                        fz_drop_device.
(16)                        transform: Transform from user space in points to device space
(16)                        in pixels.
(16)                        clip: Bounding box to restrict any marking operations of the
(16)                        draw device.
(16)                        proof_cs: Color space to render to prior to mapping to color
(16)                        space defined by pixmap.
(8)                 |
(8)                 *Overload 6:*
(9)                 Constructor using `fz_new_draw_device_with_options()`.
(16)                        Create a new pixmap and draw device, using the specified
options.
(16)                        options: Options to configure the draw device, and choose the
(16)                        resolution and colorspace.
(16)                        mediabox: The bounds of the page in points.
(16)                        pixmap: An out parameter containing the newly created pixmap.
(8)                 |
(8)                 *Overload 7:*
(9)                 Constructor using `fz_new_draw_device_with_proof()`.
(16)                        Create a device to draw on a pixmap.
(16)                        dest: Target pixmap for the draw device. See fz_new_pixmap*
(16)                        for how to obtain a pixmap. The pixmap is not cleared by the
(16)                        draw device, see fz_clear_pixmap* for how to clear it prior to
(16)                        calling fz_new_draw_device. Free the device by calling
(16)                        fz_drop_device.
(16)                        transform: Transform from user space in points to device space
(16)                        in pixels.
(16)                        proof_cs: Intermediate color space to map though when mapping
to
(16)                        color space defined by pixmap.
(8)                 |
(8)                 *Overload 8:*
(9)                 Constructor using `fz_new_list_device()`.
(16)                        Create a rendering device for a display list.
(16)                        When the device is rendering a page it will populate the
(16)                        display list with drawing commands (text, images, etc.). The
(16)                        display list can later be reused to render a page many times
(16)                        without having to re-interpret the page from the document file
(16)                        for each rendering. Once the device is no longer needed, free
(16)                        it with fz_drop_device.
(16)                        list: A display list that the list device takes a reference
to.
(8)                 |
(8)                 *Overload 9:*
(9)                 Constructor using `fz_new_ocr_device()`.
(16)                        Create a device to OCR the text on the page.
(16)                        Renders the page internally to a bitmap that is then OCRd.
Text
(16)                        is then forwarded onto the target device.
(16)                        target: The target device to receive the OCRd text.
(16)                        ctm: The transform to apply to the mediabox to get the size
for
(16)                        the rendered page image. Also used to calculate the resolution
(16)                        for the page image. In general, this will be the same as the
CTM
(16)                        that you pass to fz_run_page (or fz_run_display_list) to feed
(16)                        this device.
(16)                        mediabox: The mediabox (in points). Combined with the CTM to
get
(16)                        the bounds of the pixmap used internally for the rendered page
(16)                        image.
(16)                        with_list: If with_list is false, then all non-text operations
(16)                        are forwarded instantly to the target device. This results in
```

```
17535: (16)                         the target device seeing all NON-text operations, followed by
17536: (16)                         all the text operations (derived from OCR).
17537: (16)                         If with_list is true, then all the marking operations are
17538: (16)                         collated into a display list which is then replayed to the
17539: (16)                         target device at the end.
17540: (16)                         language: NULL (for "eng"), or a pointer to a string to
describe
17541: (16)                         the languages/scripts that should be used for OCR (e.g.
17542: (16)                         "eng,ara").
17543: (16)                         datadir: NULL (for ""), or a pointer to a path string
otherwise
17544: (16)                         provided to Tesseract in the TESSDATA_PREFIX environment
variable.
17545: (16)                         progress: NULL, or function to be called periodically to
indicate
17546: (16)                         progress. Return 0 to continue, or 1 to cancel. progress_arg
is
17547: (16)                         returned as the void *. The int is a value between 0 and 100
to
17548: (16)                         indicate progress.
17549: (16)                         progress_arg: A void * value to be parrotted back to the
progress
17550: (16)                         function.
17551: (8)                  |
17552: (8)                  *Overload 10:*
17553: (9)                   Constructor using `fz_new_stext_device()`.
17554: (16)                         Create a device to extract the text on a page.
17555: (16)                         Gather the text on a page into blocks and lines.
17556: (16)                         The reading order is taken from the order the text is drawn in
17557: (16)                         the source file, so may not be accurate.
17558: (16)                         page: The text page to which content should be added. This
will
17559: (16)                         usually be a newly created (empty) text page, but it can be
one
17560: (16)                         containing data already (for example when merging multiple
17561: (16)                         pages, or watermarking).
17562: (16)                         options: Options to configure the stext device.
17563: (8)                  |
17564: (8)                  *Overload 11:*
17565: (9)                   Constructor using `fz_new_svg_device()`.
17566: (16)                         Create a device that outputs (single page) SVG files to
17567: (16)                         the given output stream.
17568: (16)                         Equivalent to fz_new_svg_device_with_id passing id = NULL.
17569: (8)                  |
17570: (8)                  *Overload 12:*
17571: (9)                   Constructor using `fz_new_svg_device_with_id()`.
17572: (16)                         Create a device that outputs (single page) SVG files to
17573: (16)                         the given output stream.
17574: (16)                         output: The output stream to send the constructed SVG page to.
17575: (16)                         page_width, page_height: The page dimensions to use (in
points).
17576: (16)                         text_format: How to emit text. One of the following values:
17577: (20)                             FZ_SVG_TEXT_AS_TEXT: As <text> elements with possible
17578: (20)                             layout errors and mismatching fonts.
17579: (20)                             FZ_SVG_TEXT_AS_PATH: As <path> elements with exact
17580: (20)                             visual appearance.
17581: (16)                         reuse_images: Share image resources using <symbol>
definitions.
17582: (16)                         id: ID parameter to keep generated IDs unique across SVG
files.
17583: (8)                  |
17584: (8)                  *Overload 13:*
17585: (9)                   Constructor using `fz_new_test_device()`.
17586: (16)                         Create a device to test for features.
17587: (16)                         Currently only tests for the presence of non-grayscale colors.
17588: (16)                         is_color: Possible values returned:
17589: (20)                             0: Definitely greyscale
17590: (20)                             1: Probably color (all colors were grey, but there
17591: (20)                             were images or shadings in a non grey colorspace).
```

```
17592: (20)                             2: Definitely color
17593: (16)                         threshold: The difference from grayscale that will be
tolerated.
17594: (16)                         Typical values to use are either 0 (be exact) and 0.02 (allow
an
17595: (16)                         imperceptible amount of slop).
17596: (16)                         options: A set of bitfield options, from the FZ_TEST_OPT set.
17597: (16)                         passthrough: A device to pass all calls through to, or NULL.
17598: (16)                         If set, then the test device can both test and pass through to
17599: (16)                         an underlying device (like, say, the display list device).
This
17600: (16)                         means that a display list can be created and at the end we'll
17601: (16)                         know if it's colored or not.
17602: (16)                         In the absence of a passthrough device, the device will throw
17603: (16)                         an exception to stop page interpretation when color is found.
17604: (8)                  |
17605: (8)                  *Overload 14:*
17606: (9)                   Constructor using `fz_new_trace_device()`.
17607: (16)                         Create a device to print a debug trace of all device calls.
17608: (8)                  |
17609: (8)                  *Overload 15:*
17610: (9)                   Constructor using `pdf_new_pdf_device()`.
17611: (8)                  |
17612: (8)                  *Overload 16:*
17613: (9)                   Copy constructor using `fz_keep_device()`.
17614: (8)                  |
17615: (8)                  *Overload 17:*
17616: (9)                   Default constructor, sets `m_internal` to null.
17617: (8)                  |
17618: (8)                  *Overload 18:*
17619: (9)                   Constructor using raw copy of pre-existing `::fz_device`.
17620: (8)                  """
17621: (8)                  _mupdf.FzDevice_swiginit(self, _mupdf.new_FzDevice(*args))
17622: (4)              __swig_destroy__ = _mupdf.delete_FzDevice
17623: (4)              def m_internal_value(self):
17624: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17625: (8)                  return _mupdf.FzDevice_m_internal_value(self)
17626: (4)              m_internal = property(_mupdf.FzDevice_m_internal_get,
_mupdf.FzDevice_m_internal_set, doc=r""" Pointer to wrapped data.""")
17627: (4)              s_num_instances = property(_mupdf.FzDevice_s_num_instances_get,
_mupdf.FzDevice_s_num_instances_set)
17628: (0)          _mupdf.FzDevice_swigregister(FzDevice)
17629: (0)          class FzDevice2(FzDevice):
17630: (4)              r""" Wrapper class for struct fz_device with virtual fns for each fnptr;
this is for use as a SWIG Director class."""
17631: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17632: (4)              __repr__ = _swig_repr
17633: (4)              def __init__(self):
17634: (8)                  r""" == Constructor."""
17635: (8)                  if self.__class__ == FzDevice2:
17636: (12)                     _self = None
17637: (8)                  else:
17638: (12)                     _self = self
17639: (8)                  _mupdf.FzDevice2_swiginit(self, _mupdf.new_FzDevice2(_self, ))
17640: (4)              def use_virtual_close_device(self, use=True):
17641: (8)                  r"""
17642: (8)                  These methods set the function pointers in *m_internal
17643: (8)                  to point to internal callbacks that call our virtual methods.
17644: (8)                  """
17645: (8)                  return _mupdf.FzDevice2_use_virtual_close_device(self, use)
17646: (4)              def use_virtual_drop_device(self, use=True):
17647: (8)                  return _mupdf.FzDevice2_use_virtual_drop_device(self, use)
17648: (4)              def use_virtual_fill_path(self, use=True):
17649: (8)                  return _mupdf.FzDevice2_use_virtual_fill_path(self, use)
17650: (4)              def use_virtual_stroke_path(self, use=True):
17651: (8)                  return _mupdf.FzDevice2_use_virtual_stroke_path(self, use)
17652: (4)              def use_virtual_clip_path(self, use=True):
```

```
(8)                 return _mupdf.FzDevice2_use_virtual_clip_path(self, use)
(4)             def use_virtual_clip_stroke_path(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_clip_stroke_path(self, use)
(4)             def use_virtual_fill_text(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_fill_text(self, use)
(4)             def use_virtual_stroke_text(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_stroke_text(self, use)
(4)             def use_virtual_clip_text(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_clip_text(self, use)
(4)             def use_virtual_clip_stroke_text(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_clip_stroke_text(self, use)
(4)             def use_virtual_ignore_text(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_ignore_text(self, use)
(4)             def use_virtual_fill_shade(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_fill_shade(self, use)
(4)             def use_virtual_fill_image(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_fill_image(self, use)
(4)             def use_virtual_fill_image_mask(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_fill_image_mask(self, use)
(4)             def use_virtual_clip_image_mask(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_clip_image_mask(self, use)
(4)             def use_virtual_pop_clip(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_pop_clip(self, use)
(4)             def use_virtual_begin_mask(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_mask(self, use)
(4)             def use_virtual_end_mask(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_mask(self, use)
(4)             def use_virtual_begin_group(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_group(self, use)
(4)             def use_virtual_end_group(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_group(self, use)
(4)             def use_virtual_begin_tile(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_tile(self, use)
(4)             def use_virtual_end_tile(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_tile(self, use)
(4)             def use_virtual_render_flags(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_render_flags(self, use)
(4)             def use_virtual_set_default_colorspaces(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_set_default_colorspaces(self, use)
(4)             def use_virtual_begin_layer(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_layer(self, use)
(4)             def use_virtual_end_layer(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_layer(self, use)
(4)             def use_virtual_begin_structure(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_structure(self, use)
(4)             def use_virtual_end_structure(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_structure(self, use)
(4)             def use_virtual_begin_metatext(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_begin_metatext(self, use)
(4)             def use_virtual_end_metatext(self, use=True):
(8)                 return _mupdf.FzDevice2_use_virtual_end_metatext(self, use)
(4)             def close_device(self, arg_0):
(8)                 r""" Default virtual method implementations; these all throw an
exception."""
(8)                 return _mupdf.FzDevice2_close_device(self, arg_0)
(4)             def drop_device(self, arg_0):
(8)                 return _mupdf.FzDevice2_drop_device(self, arg_0)
(4)             def fill_path(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7,
arg_8):
(8)                 return _mupdf.FzDevice2_fill_path(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7, arg_8)
(4)             def stroke_path(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7,
arg_8):
(8)                 return _mupdf.FzDevice2_stroke_path(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7, arg_8)
(4)             def clip_path(self, arg_0, arg_2, arg_3, arg_4, arg_5):
(8)                 return _mupdf.FzDevice2_clip_path(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
(4)             def clip_stroke_path(self, arg_0, arg_2, arg_3, arg_4, arg_5):
```

```
17716: (8)                  return _mupdf.FzDevice2_clip_stroke_path(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
17717: (4)              def fill_text(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
17718: (8)                  return _mupdf.FzDevice2_fill_text(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
17719: (4)              def stroke_text(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7,
arg_8):
17720: (8)                  return _mupdf.FzDevice2_stroke_text(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7, arg_8)
17721: (4)              def clip_text(self, arg_0, arg_2, arg_3, arg_4):
17722: (8)                  return _mupdf.FzDevice2_clip_text(self, arg_0, arg_2, arg_3, arg_4)
17723: (4)              def clip_stroke_text(self, arg_0, arg_2, arg_3, arg_4, arg_5):
17724: (8)                  return _mupdf.FzDevice2_clip_stroke_text(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
17725: (4)              def ignore_text(self, arg_0, arg_2, arg_3):
17726: (8)                  return _mupdf.FzDevice2_ignore_text(self, arg_0, arg_2, arg_3)
17727: (4)              def fill_shade(self, arg_0, arg_2, arg_3, arg_4, arg_5):
17728: (8)                  return _mupdf.FzDevice2_fill_shade(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
17729: (4)              def fill_image(self, arg_0, arg_2, arg_3, arg_4, arg_5):
17730: (8)                  return _mupdf.FzDevice2_fill_image(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
17731: (4)              def fill_image_mask(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6,
arg_7):
17732: (8)                  return _mupdf.FzDevice2_fill_image_mask(self, arg_0, arg_2, arg_3,
arg_4, arg_5, arg_6, arg_7)
17733: (4)              def clip_image_mask(self, arg_0, arg_2, arg_3, arg_4):
17734: (8)                  return _mupdf.FzDevice2_clip_image_mask(self, arg_0, arg_2, arg_3,
arg_4)
17735: (4)              def pop_clip(self, arg_0):
17736: (8)                  return _mupdf.FzDevice2_pop_clip(self, arg_0)
17737: (4)              def begin_mask(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6):
17738: (8)                  return _mupdf.FzDevice2_begin_mask(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6)
17739: (4)              def end_mask(self, arg_0, arg_2):
17740: (8)                  return _mupdf.FzDevice2_end_mask(self, arg_0, arg_2)
17741: (4)              def begin_group(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
17742: (8)                  return _mupdf.FzDevice2_begin_group(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
17743: (4)              def end_group(self, arg_0):
17744: (8)                  return _mupdf.FzDevice2_end_group(self, arg_0)
17745: (4)              def begin_tile(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
17746: (8)                  return _mupdf.FzDevice2_begin_tile(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
17747: (4)              def end_tile(self, arg_0):
17748: (8)                  return _mupdf.FzDevice2_end_tile(self, arg_0)
17749: (4)              def render_flags(self, arg_0, arg_2, arg_3):
17750: (8)                  return _mupdf.FzDevice2_render_flags(self, arg_0, arg_2, arg_3)
17751: (4)              def set_default_colorspaces(self, arg_0, arg_2):
17752: (8)                  return _mupdf.FzDevice2_set_default_colorspaces(self, arg_0, arg_2)
17753: (4)              def begin_layer(self, arg_0, arg_2):
17754: (8)                  return _mupdf.FzDevice2_begin_layer(self, arg_0, arg_2)
17755: (4)              def end_layer(self, arg_0):
17756: (8)                  return _mupdf.FzDevice2_end_layer(self, arg_0)
17757: (4)              def begin_structure(self, arg_0, arg_2, arg_3, arg_4):
17758: (8)                  return _mupdf.FzDevice2_begin_structure(self, arg_0, arg_2, arg_3,
arg_4)
17759: (4)              def end_structure(self, arg_0):
17760: (8)                  return _mupdf.FzDevice2_end_structure(self, arg_0)
17761: (4)              def begin_metatext(self, arg_0, arg_2, arg_3):
17762: (8)                  return _mupdf.FzDevice2_begin_metatext(self, arg_0, arg_2, arg_3)
17763: (4)              def end_metatext(self, arg_0):
17764: (8)                  return _mupdf.FzDevice2_end_metatext(self, arg_0)
17765: (4)              __swig_destroy__ = _mupdf.delete_FzDevice2
17766: (4)              def __disown__(self):
17767: (8)                  self.this.disown()
17768: (8)                  _mupdf.disown_FzDevice2(self)
17769: (8)                  return weakref.proxy(self)
17770: (0)          _mupdf.FzDevice2_swigregister(FzDevice2)
```

```
17771: (0)          class FzDeviceContainerStack(object):
17772: (4)              r"""
17773: (4)              Wrapper class for struct `fz_device_container_stack`. Not copyable or
assignable.
17774: (4)              Devices can keep track of containers (clips/masks/groups/tiles)
17775: (4)              as they go to save callers having to do it.
17776: (4)              """
17777: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17778: (4)              __repr__ = _swig_repr
17779: (4)              def __init__(self, *args):
17780: (8)                  r"""
17781: (8)                  *Overload 1:*
17782: (8)                  Default constructor, sets `m_internal` to null.
17783: (8)                  |
17784: (8)                  *Overload 2:*
17785: (8)                  Constructor using raw copy of pre-existing
`::fz_device_container_stack`.
17786: (8)                  """
17787: (8)                  _mupdf.FzDeviceContainerStack_swiginit(self,
_mupdf.new_FzDeviceContainerStack(*args))
17788: (4)              __swig_destroy__ = _mupdf.delete_FzDeviceContainerStack
17789: (4)              def m_internal_value(self):
17790: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
17791: (8)                  return _mupdf.FzDeviceContainerStack_m_internal_value(self)
17792: (4)              m_internal = property(_mupdf.FzDeviceContainerStack_m_internal_get,
_mupdf.FzDeviceContainerStack_m_internal_set, doc=r""" Pointer to wrapped data.""")
17793: (4)              s_num_instances =
property(_mupdf.FzDeviceContainerStack_s_num_instances_get,
_mupdf.FzDeviceContainerStack_s_num_instances_set)
17794: (0)          _mupdf.FzDeviceContainerStack_swigregister(FzDeviceContainerStack)
17795: (0)          class FzDisplayList(object):
17796: (4)              r"""
17797: (4)              Wrapper class for struct `fz_display_list`.
17798: (4)              fz_display_list is a list containing drawing commands (text,
17799: (4)              images, etc.). The intent is two-fold: as a caching-mechanism
17800: (4)              to reduce parsing of a page, and to be used as a data
17801: (4)              structure in multi-threading where one thread parses the page
17802: (4)              and another renders pages.
17803: (4)              Create a display list with fz_new_display_list, hand it over to
17804: (4)              fz_new_list_device to have it populated, and later replay the
17805: (4)              list (once or many times) by calling fz_run_display_list. When
17806: (4)              the list is no longer needed drop it with fz_drop_display_list.
17807: (4)              """
17808: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
17809: (4)              __repr__ = _swig_repr
17810: (4)              @staticmethod
17811: (4)              def fz_new_display_list_from_page_contents(page):
17812: (8)                  r"""
17813: (8)                  Class-aware wrapper for `::fz_new_display_list_from_page_contents()`.
17814: (12)                     Create a display list from page contents (no annotations).
17815: (12)                     Ownership of the display list is returned to the caller.
17816: (8)                  """
17817: (8)                  return
_mupdf.FzDisplayList_fz_new_display_list_from_page_contents(page)
17818: (4)              def fz_bound_display_list(self):
17819: (8)                  r"""
17820: (8)                  Class-aware wrapper for `::fz_bound_display_list()`.
17821: (12)                     Return the bounding box of the page recorded in a display list.
17822: (8)                  """
17823: (8)                  return _mupdf.FzDisplayList_fz_bound_display_list(self)
17824: (4)              def fz_display_list_is_empty(self):
17825: (8)                  r"""
17826: (8)                  Class-aware wrapper for `::fz_display_list_is_empty()`.
17827: (12)                     Check for a display list being empty
17828: (12)                     list: The list to check.
17829: (12)                     Returns true if empty, false otherwise.
```

```
17830: (8)                  """
17831: (8)                  return _mupdf.FzDisplayList_fz_display_list_is_empty(self)
17832: (4)              def fz_fill_pixmap_from_display_list(self, ctm, pix):
17833: (8)                  r""" Class-aware wrapper for
`::fz_fill_pixmap_from_display_list()`."""
17834: (8)                  return _mupdf.FzDisplayList_fz_fill_pixmap_from_display_list(self,
ctm, pix)
17835: (4)              def fz_new_buffer_from_display_list(self, options):
17836: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_display_list()`."""
17837: (8)                  return _mupdf.FzDisplayList_fz_new_buffer_from_display_list(self,
options)
17838: (4)              def fz_new_list_device(self):
17839: (8)                  r"""
17840: (8)                  Class-aware wrapper for `::fz_new_list_device()`.
17841: (12)                     Create a rendering device for a display list.
17842: (12)                     When the device is rendering a page it will populate the
17843: (12)                     display list with drawing commands (text, images, etc.). The
17844: (12)                     display list can later be reused to render a page many times
17845: (12)                     without having to re-interpret the page from the document file
17846: (12)                     for each rendering. Once the device is no longer needed, free
17847: (12)                     it with fz_drop_device.
17848: (12)                     list: A display list that the list device takes a reference to.
17849: (8)                  """
17850: (8)                  return _mupdf.FzDisplayList_fz_new_list_device(self)
17851: (4)              def fz_new_pixmap_from_display_list(self, ctm, cs, alpha):
17852: (8)                  r"""
17853: (8)                  Class-aware wrapper for `::fz_new_pixmap_from_display_list()`.
17854: (12)                     Render the page to a pixmap using the transform and colorspace.
17855: (12)                     Ownership of the pixmap is returned to the caller.
17856: (8)                  """
17857: (8)                  return _mupdf.FzDisplayList_fz_new_pixmap_from_display_list(self, ctm,
cs, alpha)
17858: (4)              def fz_new_pixmap_from_display_list_with_separations(self, ctm, cs, seps,
alpha):
17859: (8)                  r"""
17860: (8)                  Class-aware wrapper for
`::fz_new_pixmap_from_display_list_with_separations()`.
17861: (12)                     Render the page contents with control over spot colors.
17862: (12)                     Ownership of the pixmap is returned to the caller.
17863: (8)                  """
17864: (8)                  return
_mupdf.FzDisplayList_fz_new_pixmap_from_display_list_with_separations(self, ctm, cs, seps, alpha)
17865: (4)              def fz_run_display_list(self, dev, ctm, scissor, cookie):
17866: (8)                  r"""
17867: (8)                  Class-aware wrapper for `::fz_run_display_list()`.
17868: (12)                     (Re)-run a display list through a device.
17869: (12)                     list: A display list, created by fz_new_display_list and
17870: (12)                     populated with objects from a page by running fz_run_page on a
17871: (12)                     device obtained from fz_new_list_device.
17872: (12)                     ctm: Transform to apply to display list contents. May include
17873: (12)                     for example scaling and rotation, see fz_scale, fz_rotate and
17874: (12)                     fz_concat. Set to fz_identity if no transformation is desired.
17875: (12)                     scissor: Only the part of the contents of the display list
17876: (12)                     visible within this area will be considered when the list is
17877: (12)                     run through the device. This does not imply for tile objects
17878: (12)                     contained in the display list.
17879: (12)                     cookie: Communication mechanism between caller and library
17880: (12)                     running the page. Intended for multi-threaded applications,
17881: (12)                     while single-threaded applications set cookie to NULL. The
17882: (12)                     caller may abort an ongoing page run. Cookie also communicates
17883: (12)                     progress information back to the caller. The fields inside
17884: (12)                     cookie are continually updated while the page is being run.
17885: (8)                  """
17886: (8)                  return _mupdf.FzDisplayList_fz_run_display_list(self, dev, ctm,
scissor, cookie)
17887: (4)              def fz_search_display_list(self, needle, hit_mark, hit_bbox, hit_max):
17888: (8)                  r"""
17889: (8)                  Class-aware wrapper for `::fz_search_display_list()`.
17890: (8)                  This method has out-params. Python/C# wrappers look like:
```

```
(12)                    `fz_search_display_list(const char *needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
(8)                 """
(8)                 return _mupdf.FzDisplayList_fz_search_display_list(self, needle, hit_mark, hit_bbox, hit_max)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using `fz_new_display_list()`.
(16)                        Create an empty display list.
(16)                        A display list contains drawing commands (text, images, etc.).
(16)                        Use fz_new_list_device for populating the list.
(16)                        mediabox: Bounds of the page (in points) represented by the
(16)                        display list.
(8)                 |
(8)                 *Overload 2:*
(9)                  Constructor using `fz_new_display_list_from_page()`.
(16)                        Create a display list.
(16)                        Ownership of the display list is returned to the caller.
(8)                 |
(8)                 *Overload 3:*
(9)                  Constructor using `fz_new_display_list_from_page_number()`.
(8)                 |
(8)                 *Overload 4:*
(9)                  Constructor using `fz_new_display_list_from_svg()`.
(16)                        Parse an SVG document into a display-list.
(8)                 |
(8)                 *Overload 5:*
(9)                  Constructor using `fz_new_display_list_from_svg_xml()`.
(16)                        Parse an SVG document into a display-list.
(8)                 |
(8)                 *Overload 6:*
(9)                  Constructor using `pdf_new_display_list_from_annot()`.
(8)                 |
(8)                 *Overload 7:*
(9)                  Copy constructor using `fz_keep_display_list()`.
(8)                 |
(8)                 *Overload 8:*
(9)                  Default constructor, sets `m_internal` to null.
(8)                 |
(8)                 *Overload 9:*
(9)                  Constructor using raw copy of pre-existing `::fz_display_list`.
(8)                 """
(8)                 _mupdf.FzDisplayList_swiginit(self, _mupdf.new_FzDisplayList(*args))
(4)             __swig_destroy__ = _mupdf.delete_FzDisplayList
(4)             def m_internal_value(self):
(8)                 r""" Return numerical value of .m_internal; helps with Python debugging."""
(8)                 return _mupdf.FzDisplayList_m_internal_value(self)
(4)             m_internal = property(_mupdf.FzDisplayList_m_internal_get, _mupdf.FzDisplayList_m_internal_set, doc=r""" Pointer to wrapped data.""")
(4)             s_num_instances = property(_mupdf.FzDisplayList_s_num_instances_get, _mupdf.FzDisplayList_s_num_instances_set)
(0)         _mupdf.FzDisplayList_swigregister(FzDisplayList)
(0)         class FzDocument(object):
(4)             r""" Wrapper class for struct `fz_document`."""
(4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v), doc="The membership flag")
(4)             __repr__ = _swig_repr
(4)             def fz_authenticate_password(self, password):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_authenticate_password()`.
(12)                    Test if the given password can decrypt the document.
(12)                    password: The password string to be checked. Some document
(12)                    specifications do not specify any particular text encoding, so
(12)                    neither do we.
(12)                    Returns 0 for failure to authenticate, non-zero for success.
(12)                    For PDF documents, further information can be given by examining
(12)                    the bits in the return code.
```

```
17954: (16)                         Bit 0 => No password required
17955: (16)                         Bit 1 => User password authenticated
17956: (16)                         Bit 2 => Owner password authenticated
17957: (8)                  """
17958: (8)                  return _mupdf.FzDocument_fz_authenticate_password(self, password)
17959: (4)              def fz_clamp_location(self, loc):
17960: (8)                  r"""
17961: (8)                  Class-aware wrapper for `::fz_clamp_location()`.
17962: (12)                     Clamps a location into valid chapter/page range. (First clamps
17963: (12)                     the chapter into range, then the page into range).
17964: (8)                  """
17965: (8)                  return _mupdf.FzDocument_fz_clamp_location(self, loc)
17966: (4)              def fz_count_chapter_pages(self, chapter):
17967: (8)                  r"""
17968: (8)                  Class-aware wrapper for `::fz_count_chapter_pages()`.
17969: (12)                     Return the number of pages in a chapter.
17970: (12)                     May return 0.
17971: (8)                  """
17972: (8)                  return _mupdf.FzDocument_fz_count_chapter_pages(self, chapter)
17973: (4)              def fz_count_chapters(self):
17974: (8)                  r"""
17975: (8)                  Class-aware wrapper for `::fz_count_chapters()`.
17976: (12)                     Return the number of chapters in the document.
17977: (12)                     At least 1.
17978: (8)                  """
17979: (8)                  return _mupdf.FzDocument_fz_count_chapters(self)
17980: (4)              def fz_count_pages(self):
17981: (8)                  r"""
17982: (8)                  Class-aware wrapper for `::fz_count_pages()`.
17983: (12)                     Return the number of pages in document
17984: (12)                     May return 0 for documents with no pages.
17985: (8)                  """
17986: (8)                  return _mupdf.FzDocument_fz_count_pages(self)
17987: (4)              def fz_document_output_intent(self):
17988: (8)                  r"""
17989: (8)                  Class-aware wrapper for `::fz_document_output_intent()`.
17990: (12)                     Find the output intent colorspace if the document has defined
17991: (12)                     one.
17992: (12)                     Returns a borrowed reference that should not be dropped, unless
17993: (12)                     it is kept first.
17994: (8)                  """
17995: (8)                  return _mupdf.FzDocument_fz_document_output_intent(self)
17996: (4)              def fz_document_supports_accelerator(self):
17997: (8)                  r"""
17998: (8)                  Class-aware wrapper for `::fz_document_supports_accelerator()`.
17999: (12)                     Query if the document supports the saving of accelerator data.
18000: (8)                  """
18001: (8)                  return _mupdf.FzDocument_fz_document_supports_accelerator(self)
18002: (4)              def fz_format_link_uri(self, dest):
18003: (8)                  r"""
18004: (8)                  Class-aware wrapper for `::fz_format_link_uri()`.
18005: (12)                     Format an internal link to a page number, location, and possible
viewing parameters,
18006: (12)                     suitable for use with fz_create_link.
18007: (12)                     Returns a newly allocated string that the caller must free.
18008: (8)                  """
18009: (8)                  return _mupdf.FzDocument_fz_format_link_uri(self, dest)
18010: (4)              def fz_has_permission(self, p):
18011: (8)                  r"""
18012: (8)                  Class-aware wrapper for `::fz_has_permission()`.
18013: (12)                     Check permission flags on document.
18014: (8)                  """
18015: (8)                  return _mupdf.FzDocument_fz_has_permission(self, p)
18016: (4)              def fz_is_document_reflowable(self):
18017: (8)                  r"""
18018: (8)                  Class-aware wrapper for `::fz_is_document_reflowable()`.
18019: (12)                     Is the document reflowable.
18020: (12)                     Returns 1 to indicate reflowable documents, otherwise 0.
18021: (8)                  """
```

```
(8)                 return _mupdf.FzDocument_fz_is_document_reflowable(self)
(4)             def fz_last_page(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_last_page()`.
(12)                    Function to get the location for the last page in the document.
(12)                    Using this can be far more efficient in some cases than calling
(12)                    fz_count_pages and using the page number.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_last_page(self)
(4)             def fz_layout_document(self, w, h, em):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_layout_document()`.
(12)                    Layout reflowable document types.
(12)                    w, h: Page size in points.
(12)                    em: Default font size in points.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_layout_document(self, w, h, em)
(4)             def fz_load_chapter_page(self, chapter, page):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_load_chapter_page()`.
(12)                    Load a page.
(12)                    After fz_load_page is it possible to retrieve the size of the
(12)                    page using fz_bound_page, or to render the page using
(12)                    fz_run_page_*. Free the page by calling fz_drop_page.
(12)                    chapter: chapter number, 0 is the first chapter of the document.
(12)                    number: page number, 0 is the first page of the chapter.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_load_chapter_page(self, chapter, page)
(4)             def fz_load_outline(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_load_outline()`.
(12)                    Load the hierarchical document outline.
(12)                    Should be freed by fz_drop_outline.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_load_outline(self)
(4)             def fz_load_page(self, number):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_load_page()`.
(12)                    Load a given page number from a document. This may be much less
(12)                    efficient than loading by location (chapter+page) for some
(12)                    document types.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_load_page(self, number)
(4)             def fz_location_from_page_number(self, number):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_location_from_page_number()`.
(12)                    Converts from page number to chapter+page. This may cause many
(12)                    chapters to be laid out in order to calculate the number of
(12)                    pages within those chapters.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_location_from_page_number(self, number)
(4)             def fz_lookup_bookmark(self, mark):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_lookup_bookmark()`.
(12)                    Find a bookmark and return its page number.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_lookup_bookmark(self, mark)
(4)             def fz_lookup_metadata(self, key, buf, size):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_lookup_metadata()`.
(12)                    Retrieve document meta data strings.
(12)                    doc: The document to query.
(12)                    key: Which meta data key to retrieve...
(12)                    Basic information:
(16)                        'format'    -- Document format and version.
(16)                        'encryption'    -- Description of the encryption used.
(12)                    From the document information dictionary:
(16)                        'info:Title'
(16)                        'info:Author'
```

```
18091: (16)                         'info:Subject'
18092: (16)                         'info:Keywords'
18093: (16)                         'info:Creator'
18094: (16)                         'info:Producer'
18095: (16)                         'info:CreationDate'
18096: (16)                         'info:ModDate'
18097: (12)                     buf: The buffer to hold the results (a nul-terminated UTF-8
18098: (12)                     string).
18099: (12)                     size: Size of 'buf'.
18100: (12)                     Returns the number of bytes need to store the string plus
terminator
18101: (12)                     (will be larger than 'size' if the output was truncated), or -1 if
the
18102: (12)                     key is not recognized or found.
18103: (8)                  """
18104: (8)                  return _mupdf.FzDocument_fz_lookup_metadata(self, key, buf, size)
18105: (4)              def fz_lookup_metadata2(self, key):
18106: (8)                  r"""
18107: (8)                  Class-aware wrapper for `::fz_lookup_metadata2()`.
18108: (8)                  C++ alternative to `fz_lookup_metadata()` that returns a `std::string`
18109: (8)                  or calls `fz_throw()` if not found.
18110: (8)                  """
18111: (8)                  return _mupdf.FzDocument_fz_lookup_metadata2(self, key)
18112: (4)              def fz_needs_password(self):
18113: (8)                  r"""
18114: (8)                  Class-aware wrapper for `::fz_needs_password()`.
18115: (12)                     Check if a document is encrypted with a
18116: (12)                     non-blank password.
18117: (8)                  """
18118: (8)                  return _mupdf.FzDocument_fz_needs_password(self)
18119: (4)              def fz_new_buffer_from_page_number(self, number, options):
18120: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_page_number()`."""
18121: (8)                  return _mupdf.FzDocument_fz_new_buffer_from_page_number(self, number,
options)
18122: (4)              def fz_new_display_list_from_page_number(self, number):
18123: (8)                  r""" Class-aware wrapper for
`::fz_new_display_list_from_page_number()`."""
18124: (8)                  return _mupdf.FzDocument_fz_new_display_list_from_page_number(self,
number)
18125: (4)              def fz_new_pdf_document_from_fz_document(self):
18126: (8)                  r""" Class-aware wrapper for
`::fz_new_pdf_document_from_fz_document()`."""
18127: (8)                  return _mupdf.FzDocument_fz_new_pdf_document_from_fz_document(self)
18128: (4)              def fz_new_pixmap_from_page_number(self, number, ctm, cs, alpha):
18129: (8)                  r""" Class-aware wrapper for `::fz_new_pixmap_from_page_number()`."""
18130: (8)                  return _mupdf.FzDocument_fz_new_pixmap_from_page_number(self, number,
ctm, cs, alpha)
18131: (4)              def fz_new_pixmap_from_page_number_with_separations(self, number, ctm, cs,
seps, alpha):
18132: (8)                  r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_number_with_separations()`."""
18133: (8)                  return
_mupdf.FzDocument_fz_new_pixmap_from_page_number_with_separations(self, number, ctm, cs, seps,
alpha)
18134: (4)              def fz_new_xhtml_document_from_document(self, *args):
18135: (8)                  r"""
18136: (8)                  *Overload 1:*
18137: (9)                   Class-aware wrapper for `::fz_new_xhtml_document_from_document()`.
18138: (16)                         Use text extraction to convert the input document into XHTML,
18139: (16)                         then open the result as a new document that can be reflowed.
18140: (8)                  |
18141: (8)                  *Overload 2:*
18142: (9)                   Class-aware wrapper for `::fz_new_xhtml_document_from_document()`.
18143: (16)                         Use text extraction to convert the input document into XHTML,
18144: (16)                         then open the result as a new document that can be reflowed.
18145: (8)                  """
18146: (8)                  return _mupdf.FzDocument_fz_new_xhtml_document_from_document(self,
*args)
18147: (4)              def fz_next_page(self, loc):
```

```
18148: (8)                  r"""
18149: (8)                  Class-aware wrapper for `::fz_next_page()`.
18150: (12)                     Function to get the location of the next page (allowing for the
18151: (12)                     end of chapters etc). If at the end of the document, returns the
18152: (12)                     current location.
18153: (8)                  """
18154: (8)                  return _mupdf.FzDocument_fz_next_page(self, loc)
18155: (4)              def fz_open_reflowed_document(self, opts):
18156: (8)                  r""" Class-aware wrapper for `::fz_open_reflowed_document()`."""
18157: (8)                  return _mupdf.FzDocument_fz_open_reflowed_document(self, opts)
18158: (4)              def fz_output_accelerator(self, accel):
18159: (8)                  r"""
18160: (8)                  Class-aware wrapper for `::fz_output_accelerator()`.
18161: (12)                     Output accelerator data for the document to a given output
18162: (12)                     stream.
18163: (8)                  """
18164: (8)                  return _mupdf.FzDocument_fz_output_accelerator(self, accel)
18165: (4)              def fz_page_number_from_location(self, loc):
18166: (8)                  r"""
18167: (8)                  Class-aware wrapper for `::fz_page_number_from_location()`.
18168: (12)                     Converts from chapter+page to page number. This may cause many
18169: (12)                     chapters to be laid out in order to calculate the number of
18170: (12)                     pages within those chapters.
18171: (8)                  """
18172: (8)                  return _mupdf.FzDocument_fz_page_number_from_location(self, loc)
18173: (4)              def fz_previous_page(self, loc):
18174: (8)                  r"""
18175: (8)                  Class-aware wrapper for `::fz_previous_page()`.
18176: (12)                     Function to get the location of the previous page (allowing for
18177: (12)                     the end of chapters etc). If already at the start of the
18178: (12)                     document, returns the current page.
18179: (8)                  """
18180: (8)                  return _mupdf.FzDocument_fz_previous_page(self, loc)
18181: (4)              def fz_process_opened_pages(self, process_openend_page, state):
18182: (8)                  r"""
18183: (8)                  Class-aware wrapper for `::fz_process_opened_pages()`.
18184: (12)                     Iterates over all opened pages of the document, calling the
18185: (12)                     provided callback for each page for processing. If the callback
18186: (12)                     returns non-NULL then the iteration stops and that value is
returned
18187: (12)                     to the called of fz_process_opened_pages().
18188: (12)                     The state pointer provided to fz_process_opened_pages() is
18189: (12)                     passed on to the callback but is owned by the caller.
18190: (12)                     Returns the first non-NULL value returned by the callback,
18191: (12)                     or NULL if the callback returned NULL for all opened pages.
18192: (8)                  """
18193: (8)                  return _mupdf.FzDocument_fz_process_opened_pages(self,
process_openend_page, state)
18194: (4)              def fz_resolve_link(self, uri, xp, yp):
18195: (8)                  r"""
18196: (8)                  Class-aware wrapper for `::fz_resolve_link()`.
18197: (8)                  This method has out-params. Python/C# wrappers look like:
18198: (12)                     `fz_resolve_link(const char *uri)` => `(fz_location, float xp,
float yp)`
18199: (12)                     Resolve an internal link to a page number.
18200: (12)                     xp, yp: Pointer to store coordinate of destination on the page.
18201: (12)                     Returns (-1,-1) if the URI cannot be resolved.
18202: (8)                  """
18203: (8)                  return _mupdf.FzDocument_fz_resolve_link(self, uri, xp, yp)
18204: (4)              def fz_run_document_structure(self, dev, cookie):
18205: (8)                  r"""
18206: (8)                  Class-aware wrapper for `::fz_run_document_structure()`.
18207: (12)                     Run the document structure through a device.
18208: (12)                     doc: Document in question.
18209: (12)                     dev: Device obtained from fz_new_*_device.
18210: (12)                     cookie: Communication mechanism between caller and library.
18211: (12)                     Intended for multi-threaded applications, while
18212: (12)                     single-threaded applications set cookie to NULL. The
18213: (12)                     caller may abort an ongoing rendering of a page. Cookie also
```

```
(12)                    communicates progress information back to the caller. The
(12)                    fields inside cookie are continually updated while the page is
(12)                    rendering.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_run_document_structure(self, dev, cookie)
(4)             def fz_save_accelerator(self, accel):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_accelerator()`.
(12)                    Save accelerator data for the document to a given file.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_save_accelerator(self, accel)
(4)             def fz_search_chapter_page_number(self, chapter, page, needle, hit_mark,
hit_bbox, hit_max):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_search_chapter_page_number()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_search_chapter_page_number(int chapter, int page, const char
*needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
(8)                 """
(8)                 return _mupdf.FzDocument_fz_search_chapter_page_number(self, chapter,
page, needle, hit_mark, hit_bbox, hit_max)
(4)             def fz_search_page2(self, number, needle, hit_max):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_search_page2()`.
(8)                 C++ alternative to fz_search_page() that returns information in a
std::vector.
(8)                 """
(8)                 return _mupdf.FzDocument_fz_search_page2(self, number, needle,
hit_max)
(4)             def fz_search_page_number(self, number, needle, hit_mark, hit_bbox,
hit_max):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_search_page_number()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_search_page_number(int number, const char *needle, ::fz_quad
*hit_bbox, int hit_max)` => `(int, int hit_mark)`
(8)                 """
(8)                 return _mupdf.FzDocument_fz_search_page_number(self, number, needle,
hit_mark, hit_bbox, hit_max)
(4)             def fz_set_metadata(self, key, value):
(8)                 r""" Class-aware wrapper for `::fz_set_metadata()`."""
(8)                 return _mupdf.FzDocument_fz_set_metadata(self, key, value)
(4)             def pdf_count_pages_imp(self, chapter):
(8)                 r""" Class-aware wrapper for `::pdf_count_pages_imp()`."""
(8)                 return _mupdf.FzDocument_pdf_count_pages_imp(self, chapter)
(4)             def pdf_document_from_fz_document(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_from_fz_document()`."""
(8)                 return _mupdf.FzDocument_pdf_document_from_fz_document(self)
(4)             def pdf_load_page_imp(self, chapter, number):
(8)                 r""" Class-aware wrapper for `::pdf_load_page_imp()`."""
(8)                 return _mupdf.FzDocument_pdf_load_page_imp(self, chapter, number)
(4)             def pdf_page_label_imp(self, chapter, page, buf, size):
(8)                 r""" Class-aware wrapper for `::pdf_page_label_imp()`."""
(8)                 return _mupdf.FzDocument_pdf_page_label_imp(self, chapter, page, buf,
size)
(4)             def pdf_specifics(self):
(8)                 r""" Class-aware wrapper for `::pdf_specifics()`."""
(8)                 return _mupdf.FzDocument_pdf_specifics(self)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using `fz_open_accelerated_document()`.
(16)                        Open a document file and read its basic structure so pages and
(16)                        objects can be located. MuPDF will try to repair broken
(16)                        documents (without actually changing the file contents).
(16)                        The returned fz_document is used when calling most other
(16)                        document related functions.
(16)                        filename: a path to a file as it would be given to open(2).
(8)                 |
```

```
18274: (8)                  *Overload 2:*
18275: (9)                   Constructor using `fz_open_accelerated_document_with_stream()`.
18276: (16)                         Open a document using the specified stream object rather than
18277: (16)                         opening a file on disk.
18278: (16)                         magic: a string used to detect document type; either a file
name
18279: (16)                         or mime-type.
18280: (16)                         stream: a stream of the document contents.
18281: (16)                         accel: NULL, or a stream of the 'accelerator' contents for
this document.
18282: (16)                         NOTE: The caller retains ownership of 'stream' and 'accel' -
the document will
18283: (16)                         take its own references if required.
18284: (8)                  |
18285: (8)                  *Overload 3:*
18286: (9)                   Constructor using
`fz_open_accelerated_document_with_stream_and_dir()`.
18287: (16)                         Open a document using the specified stream object rather than
18288: (16)                         opening a file on disk.
18289: (16)                         magic: a string used to detect document type; either a file
name
18290: (16)                         or mime-type.
18291: (16)                         stream: a stream of the document contents.
18292: (16)                         accel: NULL, or a stream of the 'accelerator' contents for
this document.
18293: (16)                         dir: NULL, or the 'directory context' for the stream contents.
18294: (16)                         NOTE: The caller retains ownership of 'stream', 'accel' and
'dir' - the document will
18295: (16)                         take its own references if required.
18296: (8)                  |
18297: (8)                  *Overload 4:*
18298: (9)                   Constructor using `fz_open_document()`.
18299: (16)                         Open a document file and read its basic structure so pages and
18300: (16)                         objects can be located. MuPDF will try to repair broken
18301: (16)                         documents (without actually changing the file contents).
18302: (16)                         The returned fz_document is used when calling most other
18303: (16)                         document related functions.
18304: (16)                         filename: a path to a file as it would be given to open(2).
18305: (8)                  |
18306: (8)                  *Overload 5:*
18307: (9)                   Constructor using `fz_open_document_with_buffer()`.
18308: (16)                         Open a document using a buffer rather than opening a file on
disk.
18309: (8)                  |
18310: (8)                  *Overload 6:*
18311: (9)                   Constructor using `fz_open_document_with_stream()`.
18312: (16)                         Open a document using the specified stream object rather than
18313: (16)                         opening a file on disk.
18314: (16)                         magic: a string used to detect document type; either a file
name
18315: (16)                         or mime-type.
18316: (16)                         stream: a stream representing the contents of the document
file.
18317: (16)                         NOTE: The caller retains ownership of 'stream' - the document
will take its
18318: (16)                         own reference if required.
18319: (8)                  |
18320: (8)                  *Overload 7:*
18321: (9)                   Constructor using `fz_open_document_with_stream_and_dir()`.
18322: (16)                         Open a document using the specified stream object rather than
18323: (16)                         opening a file on disk.
18324: (16)                         magic: a string used to detect document type; either a file
name
18325: (16)                         or mime-type.
18326: (16)                         stream: a stream representing the contents of the document
file.
18327: (16)                         dir: a 'directory context' for those filetypes that need it.
18328: (16)                         NOTE: The caller retains ownership of 'stream' and 'dir' - the
document will
```

```
18329: (16)                         take its own references if required.
18330: (8)                  |
18331: (8)                  *Overload 8:*
18332: (9)                   Returns a FzDocument for pdfdocument.m_internal.super.
18333: (8)                  |
18334: (8)                  *Overload 9:*
18335: (9)                   Copy constructor using `fz_keep_document()`.
18336: (8)                  |
18337: (8)                  *Overload 10:*
18338: (9)                   Constructor using raw copy of pre-existing `::fz_document`.
18339: (8)                  |
18340: (8)                  *Overload 11:*
18341: (9)                   Constructor using raw copy of pre-existing `::fz_document`.
18342: (8)                  """
18343: (8)                  _mupdf.FzDocument_swiginit(self, _mupdf.new_FzDocument(*args))
18344: (4)              __swig_destroy__ = _mupdf.delete_FzDocument
18345: (4)              def m_internal_value(self):
18346: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
18347: (8)                  return _mupdf.FzDocument_m_internal_value(self)
18348: (4)              m_internal = property(_mupdf.FzDocument_m_internal_get,
_mupdf.FzDocument_m_internal_set, doc=r""" Pointer to wrapped data.""")
18349: (4)              s_num_instances = property(_mupdf.FzDocument_s_num_instances_get,
_mupdf.FzDocument_s_num_instances_set)
18350: (0)          _mupdf.FzDocument_swigregister(FzDocument)
18351: (0)          class FzDocumentHandler(object):
18352: (4)              r""" Wrapper class for struct `fz_document_handler`. Not copyable or
assignable."""
18353: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18354: (4)              __repr__ = _swig_repr
18355: (4)              def fz_document_handler_open(self, stream, accel, dir, recognize_state):
18356: (8)                  r"""
18357: (8)                  Class-aware wrapper for `::fz_document_handler_open()`.   Helper for
calling `fz_document_handler::open` function pointer via
18358: (8)                  Swig from Python/C#.
18359: (8)                  """
18360: (8)                  return _mupdf.FzDocumentHandler_fz_document_handler_open(self, stream,
accel, dir, recognize_state)
18361: (4)              def fz_document_handler_recognize(self, magic):
18362: (8)                  r"""
18363: (8)                  Class-aware wrapper for `::fz_document_handler_recognize()`.   Helper
for calling a `fz_document_handler::recognize` function
18364: (8)                  pointer via Swig from Python/C#.
18365: (8)                  """
18366: (8)                  return _mupdf.FzDocumentHandler_fz_document_handler_recognize(self,
magic)
18367: (4)              def fz_register_document_handler(self):
18368: (8)                  r"""
18369: (8)                  Class-aware wrapper for `::fz_register_document_handler()`.
18370: (12)                     Register a handler for a document type.
18371: (12)                     handler: The handler to register. This must live on for the
duration of the
18372: (12)                     use of this handler. It will be passed back to the handler for
calls so
18373: (12)                     the caller can use it to retrieve state.
18374: (8)                  """
18375: (8)                  return _mupdf.FzDocumentHandler_fz_register_document_handler(self)
18376: (4)              def __init__(self, *args):
18377: (8)                  r"""
18378: (8)                  *Overload 1:*
18379: (8)                  Default constructor, sets `m_internal` to null.
18380: (8)                  |
18381: (8)                  *Overload 2:*
18382: (8)                  Constructor using raw copy of pre-existing `::fz_document_handler`.
18383: (8)                  """
18384: (8)                  _mupdf.FzDocumentHandler_swiginit(self,
_mupdf.new_FzDocumentHandler(*args))
18385: (4)              __swig_destroy__ = _mupdf.delete_FzDocumentHandler
```

```
18386: (4)              def m_internal_value(self):
18387: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
18388: (8)                  return _mupdf.FzDocumentHandler_m_internal_value(self)
18389: (4)              m_internal = property(_mupdf.FzDocumentHandler_m_internal_get,
_mupdf.FzDocumentHandler_m_internal_set, doc=r""" Pointer to wrapped data.""")
18390: (4)              s_num_instances = property(_mupdf.FzDocumentHandler_s_num_instances_get,
_mupdf.FzDocumentHandler_s_num_instances_set)
18391: (0)          _mupdf.FzDocumentHandler_swigregister(FzDocumentHandler)
18392: (0)          class FzDocumentHandlerContext(object):
18393: (4)              r""" Wrapper class for struct `fz_document_handler_context`. Not copyable
or assignable."""
18394: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18395: (4)              __repr__ = _swig_repr
18396: (4)              def __init__(self, *args):
18397: (8)                  r"""
18398: (8)                  *Overload 1:*
18399: (8)                  Default constructor, sets `m_internal` to null.
18400: (8)                  |
18401: (8)                  *Overload 2:*
18402: (8)                  Constructor using raw copy of pre-existing
`::fz_document_handler_context`.
18403: (8)                  """
18404: (8)                  _mupdf.FzDocumentHandlerContext_swiginit(self,
_mupdf.new_FzDocumentHandlerContext(*args))
18405: (4)              __swig_destroy__ = _mupdf.delete_FzDocumentHandlerContext
18406: (4)              def m_internal_value(self):
18407: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
18408: (8)                  return _mupdf.FzDocumentHandlerContext_m_internal_value(self)
18409: (4)              m_internal = property(_mupdf.FzDocumentHandlerContext_m_internal_get,
_mupdf.FzDocumentHandlerContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
18410: (4)              s_num_instances =
property(_mupdf.FzDocumentHandlerContext_s_num_instances_get,
_mupdf.FzDocumentHandlerContext_s_num_instances_set)
18411: (0)          _mupdf.FzDocumentHandlerContext_swigregister(FzDocumentHandlerContext)
18412: (0)          class FzDocumentWriter(object):
18413: (4)              r""" Wrapper class for struct `fz_document_writer`. Not copyable or
assignable."""
18414: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18415: (4)              __repr__ = _swig_repr
18416: (4)              PathType_CBZ = _mupdf.FzDocumentWriter_PathType_CBZ
18417: (4)              PathType_DOCX = _mupdf.FzDocumentWriter_PathType_DOCX
18418: (4)              PathType_ODT = _mupdf.FzDocumentWriter_PathType_ODT
18419: (4)              PathType_PAM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PAM_PIXMAP
18420: (4)              PathType_PBM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PBM_PIXMAP
18421: (4)              PathType_PCL = _mupdf.FzDocumentWriter_PathType_PCL
18422: (4)              PathType_PCLM = _mupdf.FzDocumentWriter_PathType_PCLM
18423: (4)              PathType_PDF = _mupdf.FzDocumentWriter_PathType_PDF
18424: (4)              PathType_PDFOCR = _mupdf.FzDocumentWriter_PathType_PDFOCR
18425: (4)              PathType_PGM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PGM_PIXMAP
18426: (4)              PathType_PKM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PKM_PIXMAP
18427: (4)              PathType_PNG_PIXMAP = _mupdf.FzDocumentWriter_PathType_PNG_PIXMAP
18428: (4)              PathType_PNM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PNM_PIXMAP
18429: (4)              PathType_PPM_PIXMAP = _mupdf.FzDocumentWriter_PathType_PPM_PIXMAP
18430: (4)              PathType_PS = _mupdf.FzDocumentWriter_PathType_PS
18431: (4)              PathType_PWG = _mupdf.FzDocumentWriter_PathType_PWG
18432: (4)              PathType_SVG = _mupdf.FzDocumentWriter_PathType_SVG
18433: (4)              OutputType_CBZ = _mupdf.FzDocumentWriter_OutputType_CBZ
18434: (4)              OutputType_DOCX = _mupdf.FzDocumentWriter_OutputType_DOCX
18435: (4)              OutputType_ODT = _mupdf.FzDocumentWriter_OutputType_ODT
18436: (4)              OutputType_PCL = _mupdf.FzDocumentWriter_OutputType_PCL
18437: (4)              OutputType_PCLM = _mupdf.FzDocumentWriter_OutputType_PCLM
18438: (4)              OutputType_PDF = _mupdf.FzDocumentWriter_OutputType_PDF
18439: (4)              OutputType_PDFOCR = _mupdf.FzDocumentWriter_OutputType_PDFOCR
18440: (4)              OutputType_PS = _mupdf.FzDocumentWriter_OutputType_PS
18441: (4)              OutputType_PWG = _mupdf.FzDocumentWriter_OutputType_PWG
```

```
(4)             FormatPathType_DOCUMENT = _mupdf.FzDocumentWriter_FormatPathType_DOCUMENT
(4)             FormatPathType_TEXT = _mupdf.FzDocumentWriter_FormatPathType_TEXT
(4)             def fz_begin_page(self, mediabox):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_begin_page()`.
(12)                    Called to start the process of writing a page to
(12)                    a document.
(12)                    mediabox: page size rectangle in points.
(12)                    Returns a borrowed fz_device to write page contents to. This
(12)                    should be kept if required, and only dropped if it was kept.
(8)                 """
(8)                 return _mupdf.FzDocumentWriter_fz_begin_page(self, mediabox)
(4)             def fz_close_document_writer(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_close_document_writer()`.
(12)                    Called to end the process of writing
(12)                    pages to a document.
(12)                    This writes any file level trailers required. After this
(12)                    completes successfully the file is up to date and complete.
(8)                 """
(8)                 return _mupdf.FzDocumentWriter_fz_close_document_writer(self)
(4)             def fz_end_page(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_end_page()`.
(12)                    Called to end the process of writing a page to a
(12)                    document.
(8)                 """
(8)                 return _mupdf.FzDocumentWriter_fz_end_page(self)
(4)             def fz_pdfocr_writer_set_progress(self, progress, arg_2):
(8)                 r""" Class-aware wrapper for `::fz_pdfocr_writer_set_progress()`."""
(8)                 return _mupdf.FzDocumentWriter_fz_pdfocr_writer_set_progress(self,
progress, arg_2)
(4)             def fz_write_document(self, doc):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_document()`.
(12)                    Convenience function to feed all the pages of a document to
(12)                    fz_begin_page/fz_run_page/fz_end_page.
(8)                 """
(8)                 return _mupdf.FzDocumentWriter_fz_write_document(self, doc)
(4)             def fz_write_stabilized_story(self, user_css, em, contentfn,
contentfn_ref, rectfn, rectfn_ref, pagefn, pagefn_ref, dir):
(8)                 r""" Class-aware wrapper for `::fz_write_stabilized_story()`."""
(8)                 return _mupdf.FzDocumentWriter_fz_write_stabilized_story(self,
user_css, em, contentfn, contentfn_ref, rectfn, rectfn_ref, pagefn, pagefn_ref, dir)
(4)             def fz_write_story(self, story, rectfn, rectfn_ref, positionfn,
positionfn_ref, pagefn, pagefn_ref):
(8)                 r""" Class-aware wrapper for `::fz_write_story()`."""
(8)                 return _mupdf.FzDocumentWriter_fz_write_story(self, story, rectfn,
rectfn_ref, positionfn, positionfn_ref, pagefn, pagefn_ref)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using
`fz_new_document_writer_of_size()`.
(16)                        Internal function to allocate a
(16)                        block for a derived document_writer structure, with the base
(16)                        structure's function pointers populated correctly, and the
extra
(16)                        space zero initialised.
(8)                 |
(8)                 *Overload 2:*
(9)                  Constructor using `fz_new_document_writer_with_buffer()`.
(8)                 |
(8)                 *Overload 3:*
(9)                  Constructor using `fz_new_jpeg_pixmap_writer()`.
(8)                 |
(8)                 *Overload 4:*
(9)                  Constructor using `fz_new_pixmap_writer()`.
(8)                 |
```

```
(8)                 *Overload 5:*
(9)                  Constructor using `fz_new_svg_writer_with_output()`.
(8)                 |
(8)                 *Overload 6:*
(9)                  Constructor using one of:
(16)                        fz_new_cbz_writer()
(16)                        fz_new_docx_writer()
(16)                        fz_new_odt_writer()
(16)                        fz_new_pam_pixmap_writer()
(16)                        fz_new_pbm_pixmap_writer()
(16)                        fz_new_pcl_writer()
(16)                        fz_new_pclm_writer()
(16)                        fz_new_pdf_writer()
(16)                        fz_new_pdfocr_writer()
(16)                        fz_new_pgm_pixmap_writer()
(16)                        fz_new_pkm_pixmap_writer()
(16)                        fz_new_png_pixmap_writer()
(16)                        fz_new_pnm_pixmap_writer()
(16)                        fz_new_ppm_pixmap_writer()
(16)                        fz_new_ps_writer()
(16)                        fz_new_pwg_writer()
(16)                        fz_new_svg_writer()
(8)                 |
(8)                 *Overload 7:*
(9)                  Constructor using one of:
(16)                        fz_new_cbz_writer_with_output()
(16)                        fz_new_docx_writer_with_output()
(16)                        fz_new_odt_writer_with_output()
(16)                        fz_new_pcl_writer_with_output()
(16)                        fz_new_pclm_writer_with_output()
(16)                        fz_new_pdf_writer_with_output()
(16)                        fz_new_pdfocr_writer_with_output()
(16)                        fz_new_ps_writer_with_output()
(16)                        fz_new_pwg_writer_with_output()
(12)                     This constructor takes ownership of <out> -
(12)                     out.m_internal is set to NULL after this constructor
(12)                     returns so <out> must not be used again.
(8)                 |
(8)                 *Overload 8:*
(9)                  Constructor using one of:
(16)                        fz_new_document_writer()
(16)                        fz_new_text_writer()
(8)                 |
(8)                 *Overload 9:*
(9)                  Constructor using fz_new_document_writer_with_output().
(12)                     This constructor takes ownership of <out> -
(12)                     out.m_internal is set to NULL after this constructor
(12)                     returns so <out> must not be used again.
(8)                 |
(8)                 *Overload 10:*
(9)                  Constructor using fz_new_text_writer_with_output().
(12)                     This constructor takes ownership of <out> -
(12)                     out.m_internal is set to NULL after this constructor
(12)                     returns so <out> must not be used again.
(8)                 |
(8)                 *Overload 11:*
(9)                  Default constructor, sets `m_internal` to null.
(8)                 |
(8)                 *Overload 12:*
(9)                  Constructor using raw copy of pre-existing `::fz_document_writer`.
(8)                 """
(8)                 _mupdf.FzDocumentWriter_swiginit(self,
_mupdf.new_FzDocumentWriter(*args))
(4)             __swig_destroy__ = _mupdf.delete_FzDocumentWriter
(4)             def m_internal_value(self):
(8)                 r""" Return numerical value of .m_internal; helps with Python
debugging."""
(8)                 return _mupdf.FzDocumentWriter_m_internal_value(self)
(4)             m_internal = property(_mupdf.FzDocumentWriter_m_internal_get,
```

```
_mupdf.FzDocumentWriter_m_internal_set, doc=r""" Pointer to wrapped data.""")
18571: (4)              s_num_instances = property(_mupdf.FzDocumentWriter_s_num_instances_get,
_mupdf.FzDocumentWriter_s_num_instances_set)
18572: (0)          _mupdf.FzDocumentWriter_swigregister(FzDocumentWriter)
18573: (0)          class FzDrawOptions(object):
18574: (4)              r"""
18575: (4)              Wrapper class for struct `fz_draw_options`. Not copyable or assignable.
18576: (4)              struct fz_draw_options: Options for creating a pixmap and draw
18577: (4)              device.
18578: (4)              """
18579: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18580: (4)              __repr__ = _swig_repr
18581: (4)              def __init__(self, *args):
18582: (8)                  r"""
18583: (8)                  *Overload 1:*
18584: (9)                   == Constructors.  Constructor using `fz_parse_draw_options()`.
18585: (16)                         Parse draw device options from a comma separated key-value
string.
18586: (8)                  |
18587: (8)                  *Overload 2:*
18588: (9)                   Default constructor, sets each member to default value.
18589: (8)                  |
18590: (8)                  *Overload 3:*
18591: (9)                   Constructor using raw copy of pre-existing `::fz_draw_options`.
18592: (8)                  |
18593: (8)                  *Overload 4:*
18594: (9)                   Constructor using raw copy of pre-existing `::fz_draw_options`.
18595: (8)                  """
18596: (8)                  _mupdf.FzDrawOptions_swiginit(self, _mupdf.new_FzDrawOptions(*args))
18597: (4)              def internal(self, *args):
18598: (8)                  r"""
18599: (8)                  *Overload 1:*
18600: (8)                  Access as underlying struct.
18601: (8)                  |
18602: (8)                  *Overload 2:*
18603: (8)                  Access as underlying struct.
18604: (8)                  """
18605: (8)                  return _mupdf.FzDrawOptions_internal(self, *args)
18606: (4)              __swig_destroy__ = _mupdf.delete_FzDrawOptions
18607: (4)              rotate = property(_mupdf.FzDrawOptions_rotate_get,
_mupdf.FzDrawOptions_rotate_set)
18608: (4)              x_resolution = property(_mupdf.FzDrawOptions_x_resolution_get,
_mupdf.FzDrawOptions_x_resolution_set)
18609: (4)              y_resolution = property(_mupdf.FzDrawOptions_y_resolution_get,
_mupdf.FzDrawOptions_y_resolution_set)
18610: (4)              width = property(_mupdf.FzDrawOptions_width_get,
_mupdf.FzDrawOptions_width_set)
18611: (4)              height = property(_mupdf.FzDrawOptions_height_get,
_mupdf.FzDrawOptions_height_set)
18612: (4)              colorspace = property(_mupdf.FzDrawOptions_colorspace_get,
_mupdf.FzDrawOptions_colorspace_set)
18613: (4)              alpha = property(_mupdf.FzDrawOptions_alpha_get,
_mupdf.FzDrawOptions_alpha_set)
18614: (4)              graphics = property(_mupdf.FzDrawOptions_graphics_get,
_mupdf.FzDrawOptions_graphics_set)
18615: (4)              text = property(_mupdf.FzDrawOptions_text_get,
_mupdf.FzDrawOptions_text_set)
18616: (4)              s_num_instances = property(_mupdf.FzDrawOptions_s_num_instances_get,
_mupdf.FzDrawOptions_s_num_instances_set)
18617: (4)              def to_string(self):
18618: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
18619: (8)                  return _mupdf.FzDrawOptions_to_string(self)
18620: (4)              def __eq__(self, rhs):
18621: (8)                  r""" Comparison method."""
18622: (8)                  return _mupdf.FzDrawOptions___eq__(self, rhs)
18623: (4)              def __ne__(self, rhs):
18624: (8)                  r""" Comparison method."""
```

```
18625: (8)                  return _mupdf.FzDrawOptions___ne__(self, rhs)
18626: (0)          _mupdf.FzDrawOptions_swigregister(FzDrawOptions)
18627: (0)          class FzErrorContext(object):
18628: (4)              r""" Wrapper class for struct `fz_error_context`. Not copyable or
assignable."""
18629: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18630: (4)              __repr__ = _swig_repr
18631: (4)              def __init__(self, *args):
18632: (8)                  r"""
18633: (8)                  *Overload 1:*
18634: (8)                  Default constructor, sets `m_internal` to null.
18635: (8)                  |
18636: (8)                  *Overload 2:*
18637: (8)                  Constructor using raw copy of pre-existing `::fz_error_context`.
18638: (8)                  """
18639: (8)                  _mupdf.FzErrorContext_swiginit(self, _mupdf.new_FzErrorContext(*args))
18640: (4)              __swig_destroy__ = _mupdf.delete_FzErrorContext
18641: (4)              def m_internal_value(self):
18642: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
18643: (8)                  return _mupdf.FzErrorContext_m_internal_value(self)
18644: (4)              m_internal = property(_mupdf.FzErrorContext_m_internal_get,
_mupdf.FzErrorContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
18645: (4)              s_num_instances = property(_mupdf.FzErrorContext_s_num_instances_get,
_mupdf.FzErrorContext_s_num_instances_set)
18646: (0)          _mupdf.FzErrorContext_swigregister(FzErrorContext)
18647: (0)          class FzErrorStackSlot(object):
18648: (4)              r""" Wrapper class for struct `fz_error_stack_slot`. Not copyable or
assignable."""
18649: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18650: (4)              __repr__ = _swig_repr
18651: (4)              def __init__(self, *args):
18652: (8)                  r"""
18653: (8)                  *Overload 1:*
18654: (8)                  Default constructor, sets `m_internal` to null.
18655: (8)                  |
18656: (8)                  *Overload 2:*
18657: (8)                  Constructor using raw copy of pre-existing `::fz_error_stack_slot`.
18658: (8)                  """
18659: (8)                  _mupdf.FzErrorStackSlot_swiginit(self,
_mupdf.new_FzErrorStackSlot(*args))
18660: (4)              __swig_destroy__ = _mupdf.delete_FzErrorStackSlot
18661: (4)              def m_internal_value(self):
18662: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
18663: (8)                  return _mupdf.FzErrorStackSlot_m_internal_value(self)
18664: (4)              m_internal = property(_mupdf.FzErrorStackSlot_m_internal_get,
_mupdf.FzErrorStackSlot_m_internal_set, doc=r""" Pointer to wrapped data.""")
18665: (4)              s_num_instances = property(_mupdf.FzErrorStackSlot_s_num_instances_get,
_mupdf.FzErrorStackSlot_s_num_instances_set)
18666: (0)          _mupdf.FzErrorStackSlot_swigregister(FzErrorStackSlot)
18667: (0)          class FzFont(object):
18668: (4)              r"""
18669: (4)              Wrapper class for struct `fz_font`.
18670: (4)              An abstract font handle.
18671: (4)              """
18672: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
18673: (4)              __repr__ = _swig_repr
18674: (4)              def fz_advance_glyph(self, glyph, wmode):
18675: (8)                  r"""
18676: (8)                  Class-aware wrapper for `::fz_advance_glyph()`.
18677: (12)                     Return the advance for a given glyph.
18678: (12)                     font: The font to look for the glyph in.
18679: (12)                     glyph: The glyph to find the advance for.
18680: (12)                     wmode: 1 for vertical mode, 0 for horizontal.
18681: (12)                     Returns the advance for the glyph.
```

```
18682: (8)                  """
18683: (8)                  return _mupdf.FzFont_fz_advance_glyph(self, glyph, wmode)
18684: (4)              def fz_bound_glyph(self, gid, trm):
18685: (8)                  r"""
18686: (8)                  Class-aware wrapper for `::fz_bound_glyph()`.
18687: (12)                     Return a bbox for a given glyph in a font.
18688: (12)                     font: The font to look for the glyph in.
18689: (12)                     gid: The glyph to bound.
18690: (12)                     trm: The matrix to apply to the glyph before bounding.
18691: (12)                     Returns rectangle by value containing the bounds of the given
18692: (12)                     glyph.
18693: (8)                  """
18694: (8)                  return _mupdf.FzFont_fz_bound_glyph(self, gid, trm)
18695: (4)              def fz_decouple_type3_font(self, t3doc):
18696: (8)                  r""" Class-aware wrapper for `::fz_decouple_type3_font()`."""
18697: (8)                  return _mupdf.FzFont_fz_decouple_type3_font(self, t3doc)
18698: (4)              def fz_encode_character(self, unicode):
18699: (8)                  r"""
18700: (8)                  Class-aware wrapper for `::fz_encode_character()`.
18701: (12)                     Find the glyph id for a given unicode
18702: (12)                     character within a font.
18703: (12)                     font: The font to look for the unicode character in.
18704: (12)                     unicode: The unicode character to encode.
18705: (12)                     Returns the glyph id for the given unicode value, or 0 if
18706: (12)                     unknown.
18707: (8)                  """
18708: (8)                  return _mupdf.FzFont_fz_encode_character(self, unicode)
18709: (4)              def fz_encode_character_by_glyph_name(self, glyphname):
18710: (8)                  r"""
18711: (8)                  Class-aware wrapper for `::fz_encode_character_by_glyph_name()`.
18712: (12)                     Encode character.
18713: (12)                     Either by direct lookup of glyphname within a font, or, failing
18714: (12)                     that, by mapping glyphname to unicode and thence to the glyph
18715: (12)                     index within the given font.
18716: (12)                     Returns zero for type3 fonts.
18717: (8)                  """
18718: (8)                  return _mupdf.FzFont_fz_encode_character_by_glyph_name(self,
glyphname)
18719: (4)              def fz_encode_character_sc(self, unicode):
18720: (8)                  r"""
18721: (8)                  Class-aware wrapper for `::fz_encode_character_sc()`.
18722: (12)                     Encode character, preferring small-caps variant if available.
18723: (12)                     font: The font to look for the unicode character in.
18724: (12)                     unicode: The unicode character to encode.
18725: (12)                     Returns the glyph id for the given unicode value, or 0 if
18726: (12)                     unknown.
18727: (8)                  """
18728: (8)                  return _mupdf.FzFont_fz_encode_character_sc(self, unicode)
18729: (4)              def fz_encode_character_with_fallback(self, unicode, script, language,
out_font):
18730: (8)                  r"""
18731: (8)                  Class-aware wrapper for `::fz_encode_character_with_fallback()`.
18732: (8)                  This method has out-params. Python/C# wrappers look like:
18733: (12)                     `fz_encode_character_with_fallback(int unicode, int script, int
language, ::fz_font **out_font)` => `(int)`
18734: (12)                     Find the glyph id for
18735: (12)                     a given unicode character within a font, falling back to
18736: (12)                     an alternative if not found.
18737: (12)                     font: The font to look for the unicode character in.
18738: (12)                     unicode: The unicode character to encode.
18739: (12)                     script: The script in use.
18740: (12)                     language: The language in use.
18741: (12)                     out_font: The font handle in which the given glyph represents
18742: (12)                     the requested unicode character. The caller does not own the
18743: (12)                     reference it is passed, so should call fz_keep_font if it is
18744: (12)                     not simply to be used immediately.
18745: (12)                     Returns the glyph id for the given unicode value in the supplied
18746: (12)                     font (and sets *out_font to font) if it is present. Otherwise
18747: (12)                     an alternative fallback font (based on script/language) is
```

```
18748: (12)                     searched for. If the glyph is found therein, *out_font is set
18749: (12)                     to this reference, and the glyph reference is returned. If it
18750: (12)                     cannot be found anywhere, the function returns 0.
18751: (8)                  """
18752: (8)                  return _mupdf.FzFont_fz_encode_character_with_fallback(self, unicode,
script, language, out_font)
18753: (4)              def fz_extract_ttf_from_ttc(self):
18754: (8)                  r""" Class-aware wrapper for `::fz_extract_ttf_from_ttc()`."""
18755: (8)                  return _mupdf.FzFont_fz_extract_ttf_from_ttc(self)
18756: (4)              def fz_font_ascender(self):
18757: (8)                  r"""
18758: (8)                  Class-aware wrapper for `::fz_font_ascender()`.
18759: (12)                     Retrieve font ascender in ems.
18760: (8)                  """
18761: (8)                  return _mupdf.FzFont_fz_font_ascender(self)
18762: (4)              def fz_font_bbox(self):
18763: (8)                  r"""
18764: (8)                  Class-aware wrapper for `::fz_font_bbox()`.
18765: (12)                     Retrieve the font bbox.
18766: (12)                     font: The font to query.
18767: (12)                     Returns the font bbox by value; it is valid only if
18768: (12)                     fz_font_flags(font)->invalid_bbox is zero.
18769: (8)                  """
18770: (8)                  return _mupdf.FzFont_fz_font_bbox(self)
18771: (4)              def fz_font_descender(self):
18772: (8)                  r"""
18773: (8)                  Class-aware wrapper for `::fz_font_descender()`.
18774: (12)                     Retrieve font descender in ems.
18775: (8)                  """
18776: (8)                  return _mupdf.FzFont_fz_font_descender(self)
18777: (4)              def fz_font_digest(self, digest):
18778: (8)                  r"""
18779: (8)                  Class-aware wrapper for `::fz_font_digest()`.
18780: (12)                     Retrieve the MD5 digest for the font's data.
18781: (8)                  """
18782: (8)                  return _mupdf.FzFont_fz_font_digest(self, digest)
18783: (4)              def fz_font_ft_face(self):
18784: (8)                  r"""
18785: (8)                  Class-aware wrapper for `::fz_font_ft_face()`.
18786: (12)                     Retrieve the FT_Face handle
18787: (12)                     for the font.
18788: (12)                     font: The font to query
18789: (12)                     Returns the FT_Face handle for the font, or NULL
18790: (12)                     if not a freetype handled font. (Cast to void *
18791: (12)                     to avoid nasty header exposure).
18792: (8)                  """
18793: (8)                  return _mupdf.FzFont_fz_font_ft_face(self)
18794: (4)              def fz_font_is_bold(self):
18795: (8)                  r"""
18796: (8)                  Class-aware wrapper for `::fz_font_is_bold()`.
18797: (12)                     Query whether the font flags say that this font is bold.
18798: (8)                  """
18799: (8)                  return _mupdf.FzFont_fz_font_is_bold(self)
18800: (4)              def fz_font_is_italic(self):
18801: (8)                  r"""
18802: (8)                  Class-aware wrapper for `::fz_font_is_italic()`.
18803: (12)                     Query whether the font flags say that this font is italic.
18804: (8)                  """
18805: (8)                  return _mupdf.FzFont_fz_font_is_italic(self)
18806: (4)              def fz_font_is_monospaced(self):
18807: (8)                  r"""
18808: (8)                  Class-aware wrapper for `::fz_font_is_monospaced()`.
18809: (12)                     Query whether the font flags say that this font is monospaced.
18810: (8)                  """
18811: (8)                  return _mupdf.FzFont_fz_font_is_monospaced(self)
18812: (4)              def fz_font_is_serif(self):
18813: (8)                  r"""
18814: (8)                  Class-aware wrapper for `::fz_font_is_serif()`.
18815: (12)                     Query whether the font flags say that this font is serif.
```

```
18816: (8)                    """
18817: (8)                    return _mupdf.FzFont_fz_font_is_serif(self)
18818: (4)                def fz_font_name(self):
18819: (8)                    r"""
18820: (8)                    Class-aware wrapper for `::fz_font_name()`.
18821: (12)                       Retrieve a pointer to the name of the font.
18822: (12)                       font: The font to query.
18823: (12)                       Returns a pointer to an internal copy of the font name.
18824: (12)                       Will never be NULL, but may be the empty string.
18825: (8)                    """
18826: (8)                    return _mupdf.FzFont_fz_font_name(self)
18827: (4)                def fz_font_t3_procs(self):
18828: (8)                    r"""
18829: (8)                    Class-aware wrapper for `::fz_font_t3_procs()`.
18830: (12)                       Retrieve the Type3 procs
18831: (12)                       for a font.
18832: (12)                       font: The font to query
18833: (12)                       Returns the t3_procs pointer. Will be NULL for a
18834: (12)                       non type-3 font.
18835: (8)                    """
18836: (8)                    return _mupdf.FzFont_fz_font_t3_procs(self)
18837: (4)                def fz_get_glyph_name(self, glyph, buf, size):
18838: (8)                    r"""
18839: (8)                    Class-aware wrapper for `::fz_get_glyph_name()`.
18840: (12)                       Find the name of a glyph
18841: (12)                       font: The font to look for the glyph in.
18842: (12)                       glyph: The glyph id to look for.
18843: (12)                       buf: Pointer to a buffer for the name to be inserted into.
18844: (12)                       size: The size of the buffer.
18845: (12)                       If a font contains a name table, then the name of the glyph
18846: (12)                       will be returned in the supplied buffer. Otherwise a name
18847: (12)                       is synthesised. The name will be truncated to fit in
18848: (12)                       the buffer.
18849: (8)                    """
18850: (8)                    return _mupdf.FzFont_fz_get_glyph_name(self, glyph, buf, size)
18851: (4)                def fz_get_glyph_name2(self, glyph):
18852: (8)                    r"""
18853: (8)                    Class-aware wrapper for `::fz_get_glyph_name2()`.
18854: (8)                    C++ alternative to fz_get_glyph_name() that returns information in a
std::string.
18855: (8)                    """
18856: (8)                    return _mupdf.FzFont_fz_get_glyph_name2(self, glyph)
18857: (4)                def fz_glyph_cacheable(self, gid):
18858: (8)                    r"""
18859: (8)                    Class-aware wrapper for `::fz_glyph_cacheable()`.
18860: (12)                       Determine if a given glyph in a font
18861: (12)                       is cacheable. Certain glyphs in a type 3 font cannot safely
18862: (12)                       be cached, as their appearance depends on the enclosing
18863: (12)                       graphic state.
18864: (12)                       font: The font to look for the glyph in.
18865: (12)                       gif: The glyph to query.
18866: (12)                       Returns non-zero if cacheable, 0 if not.
18867: (8)                    """
18868: (8)                    return _mupdf.FzFont_fz_glyph_cacheable(self, gid)
18869: (4)                def fz_measure_string(self, trm, s, wmode, bidi_level, markup_dir,
language):
18870: (8)                    r"""
18871: (8)                    Class-aware wrapper for `::fz_measure_string()`.
18872: (12)                       Measure the advance width of a UTF8 string should it be added to a
text object.
18873: (12)                       This uses the same layout algorithms as fz_show_string, and can be
used
18874: (12)                       to calculate text alignment adjustments.
18875: (8)                    """
18876: (8)                    return _mupdf.FzFont_fz_measure_string(self, trm, s, wmode,
bidi_level, markup_dir, language)
18877: (4)                def fz_outline_glyph(self, gid, ctm):
18878: (8)                    r"""
18879: (8)                    Class-aware wrapper for `::fz_outline_glyph()`.
```

```
18880: (12)                     Look a glyph up from a font, and return the outline of the
18881: (12)                     glyph using the given transform.
18882: (12)                     The caller owns the returned path, and so is responsible for
18883: (12)                     ensuring that it eventually gets dropped.
18884: (8)                  """
18885: (8)                  return _mupdf.FzFont_fz_outline_glyph(self, gid, ctm)
18886: (4)              def fz_prepare_t3_glyph(self, gid):
18887: (8)                  r"""
18888: (8)                  Class-aware wrapper for `::fz_prepare_t3_glyph()`.
18889: (12)                     Force a type3 font to cache the displaylist for a given glyph
18890: (12)                     id.
18891: (12)                     This caching can involve reading the underlying file, so must
18892: (12)                     happen ahead of time, so we aren't suddenly forced to read the
18893: (12)                     file while playing a displaylist back.
18894: (8)                  """
18895: (8)                  return _mupdf.FzFont_fz_prepare_t3_glyph(self, gid)
18896: (4)              def fz_render_glyph_pixmap(self, gid, ctm, scissor, aa):
18897: (8)                  r"""
18898: (8)                  Class-aware wrapper for `::fz_render_glyph_pixmap()`.
18899: (12)                     Create a pixmap containing a rendered glyph.
18900: (12)                     Lookup gid from font, clip it with scissor, and rendering it
18901: (12)                     with aa bits of antialiasing into a new pixmap.
18902: (12)                     The caller takes ownership of the pixmap and so must free it.
18903: (12)                     Note: This function is no longer used for normal rendering
18904: (12)                     operations, and is kept around just because we use it in the
18905: (12)                     app. It should be considered "at risk" of removal from the API.
18906: (8)                  """
18907: (8)                  return _mupdf.FzFont_fz_render_glyph_pixmap(self, gid, ctm, scissor,
aa)
18908: (4)              def fz_run_t3_glyph(self, gid, trm, dev):
18909: (8)                  r"""
18910: (8)                  Class-aware wrapper for `::fz_run_t3_glyph()`.
18911: (12)                     Run a glyph from a Type3 font to
18912: (12)                     a given device.
18913: (12)                     font: The font to find the glyph in.
18914: (12)                     gid: The glyph to run.
18915: (12)                     trm: The transform to apply.
18916: (12)                     dev: The device to render onto.
18917: (8)                  """
18918: (8)                  return _mupdf.FzFont_fz_run_t3_glyph(self, gid, trm, dev)
18919: (4)              def fz_set_font_bbox(self, xmin, ymin, xmax, ymax):
18920: (8)                  r"""
18921: (8)                  Class-aware wrapper for `::fz_set_font_bbox()`.
18922: (12)                     Set the font bbox.
18923: (12)                     font: The font to set the bbox for.
18924: (12)                     xmin, ymin, xmax, ymax: The bounding box.
18925: (8)                  """
18926: (8)                  return _mupdf.FzFont_fz_set_font_bbox(self, xmin, ymin, xmax, ymax)
18927: (4)              def fz_set_font_embedding(self, embed):
18928: (8)                  r"""
18929: (8)                  Class-aware wrapper for `::fz_set_font_embedding()`.
18930: (12)                     Control whether a given font should be embedded or not when
writing.
18931: (8)                  """
18932: (8)                  return _mupdf.FzFont_fz_set_font_embedding(self, embed)
18933: (4)              def pdf_font_writing_supported(self):
18934: (8)                  r""" Class-aware wrapper for `::pdf_font_writing_supported()`."""
18935: (8)                  return _mupdf.FzFont_pdf_font_writing_supported(self)
18936: (4)              def pdf_layout_fit_text(self, lang, str, bounds):
18937: (8)                  r""" Class-aware wrapper for `::pdf_layout_fit_text()`."""
18938: (8)                  return _mupdf.FzFont_pdf_layout_fit_text(self, lang, str, bounds)
18939: (4)              def __init__(self, *args):
18940: (8)                  r"""
18941: (8)                  *Overload 1:*
18942: (9)                   == Constructors.  Constructor using `fz_new_base14_font()`.
18943: (16)                         Create a new font from one of the built-in fonts.
18944: (8)                  |
18945: (8)                  *Overload 2:*
18946: (9)                   Constructor using `fz_new_builtin_font()`.
```

```
18947: (8)                  |
18948: (8)                  *Overload 3:*
18949: (9)                   Constructor using `fz_new_cjk_font()`.
18950: (8)                  |
18951: (8)                  *Overload 4:*
18952: (9)                   Constructor using `fz_new_font_from_buffer()`.
18953: (16)                         Create a new font from a font file in a fz_buffer.
18954: (16)                         Fonts created in this way, will be eligible for embedding by
default.
18955: (16)                         name: Name of font (leave NULL to use name from font).
18956: (16)                         buffer: Buffer to load from.
18957: (16)                         index: Which font from the file to load (0 for default).
18958: (16)                         use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
18959: (16)                         Returns new font handle, or throws exception on error.
18960: (8)                  |
18961: (8)                  *Overload 5:*
18962: (9)                   Constructor using `fz_new_font_from_file()`.
18963: (16)                         Create a new font from a font file.
18964: (16)                         Fonts created in this way, will be eligible for embedding by
default.
18965: (16)                         name: Name of font (leave NULL to use name from font).
18966: (16)                         path: File path to load from.
18967: (16)                         index: Which font from the file to load (0 for default).
18968: (16)                         use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
18969: (16)                         Returns new font handle, or throws exception on error.
18970: (8)                  |
18971: (8)                  *Overload 6:*
18972: (9)                   Constructor using `fz_new_font_from_memory()`.
18973: (16)                         Create a new font from a font file in memory.
18974: (16)                         Fonts created in this way, will be eligible for embedding by
default.
18975: (16)                         name: Name of font (leave NULL to use name from font).
18976: (16)                         data: Pointer to the font file data.
18977: (16)                         len: Length of the font file data.
18978: (16)                         index: Which font from the file to load (0 for default).
18979: (16)                         use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
18980: (16)                         Returns new font handle, or throws exception on error.
18981: (8)                  |
18982: (8)                  *Overload 7:*
18983: (9)                   Constructor using `fz_new_type3_font()`.
18984: (16)                         Create a new (empty) type3 font.
18985: (16)                         name: Name of font (or NULL).
18986: (16)                         matrix: Font matrix.
18987: (16)                         Returns a new font handle, or throws exception on
18988: (16)                         allocation failure.
18989: (8)                  |
18990: (8)                  *Overload 8:*
18991: (9)                   Copy constructor using `fz_keep_font()`.
18992: (8)                  |
18993: (8)                  *Overload 9:*
18994: (9)                   Default constructor, sets `m_internal` to null.
18995: (8)                  |
18996: (8)                  *Overload 10:*
18997: (9)                   Constructor using raw copy of pre-existing `::fz_font`.
18998: (8)                  """
18999: (8)                  _mupdf.FzFont_swiginit(self, _mupdf.new_FzFont(*args))
19000: (4)              __swig_destroy__ = _mupdf.delete_FzFont
19001: (4)              def m_internal_value(self):
19002: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19003: (8)                  return _mupdf.FzFont_m_internal_value(self)
19004: (4)              m_internal = property(_mupdf.FzFont_m_internal_get,
_mupdf.FzFont_m_internal_set, doc=r""" Pointer to wrapped data.""")
19005: (4)              s_num_instances = property(_mupdf.FzFont_s_num_instances_get,
_mupdf.FzFont_s_num_instances_set)
19006: (0)          _mupdf.FzFont_swigregister(FzFont)
19007: (0)          class FzFontContext(object):
19008: (4)              r""" Wrapper class for struct `fz_font_context`. Not copyable or
assignable."""
```

```
19009: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19010: (4)              __repr__ = _swig_repr
19011: (4)              def __init__(self, *args):
19012: (8)                  r"""
19013: (8)                  *Overload 1:*
19014: (8)                  Default constructor, sets `m_internal` to null.
19015: (8)                  |
19016: (8)                  *Overload 2:*
19017: (8)                  Constructor using raw copy of pre-existing `::fz_font_context`.
19018: (8)                  """
19019: (8)                  _mupdf.FzFontContext_swiginit(self, _mupdf.new_FzFontContext(*args))
19020: (4)              __swig_destroy__ = _mupdf.delete_FzFontContext
19021: (4)              def m_internal_value(self):
19022: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19023: (8)                  return _mupdf.FzFontContext_m_internal_value(self)
19024: (4)              m_internal = property(_mupdf.FzFontContext_m_internal_get,
_mupdf.FzFontContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
19025: (4)              s_num_instances = property(_mupdf.FzFontContext_s_num_instances_get,
_mupdf.FzFontContext_s_num_instances_set)
19026: (0)          _mupdf.FzFontContext_swigregister(FzFontContext)
19027: (0)          class FzFontFlagsT(object):
19028: (4)              r"""
19029: (4)              Wrapper class for struct `fz_font_flags_t`. Not copyable or assignable.
19030: (4)              Every fz_font carries a set of flags
19031: (4)              within it, in a fz_font_flags_t structure.
19032: (4)              """
19033: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19034: (4)              __repr__ = _swig_repr
19035: (4)              def __init__(self, *args):
19036: (8)                  r"""
19037: (8)                  *Overload 1:*
19038: (8)                  Default constructor, sets `m_internal` to null.
19039: (8)                  |
19040: (8)                  *Overload 2:*
19041: (8)                  Constructor using raw copy of pre-existing `::fz_font_flags_t`.
19042: (8)                  """
19043: (8)                  _mupdf.FzFontFlagsT_swiginit(self, _mupdf.new_FzFontFlagsT(*args))
19044: (4)              __swig_destroy__ = _mupdf.delete_FzFontFlagsT
19045: (4)              def m_internal_value(self):
19046: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19047: (8)                  return _mupdf.FzFontFlagsT_m_internal_value(self)
19048: (4)              m_internal = property(_mupdf.FzFontFlagsT_m_internal_get,
_mupdf.FzFontFlagsT_m_internal_set, doc=r""" Pointer to wrapped data.""")
19049: (4)              s_num_instances = property(_mupdf.FzFontFlagsT_s_num_instances_get,
_mupdf.FzFontFlagsT_s_num_instances_set)
19050: (0)          _mupdf.FzFontFlagsT_swigregister(FzFontFlagsT)
19051: (0)          class FzFunction(object):
19052: (4)              r""" Wrapper class for struct `fz_function`."""
19053: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19054: (4)              __repr__ = _swig_repr
19055: (4)              def fz_eval_function(self, _in, inlen, out, outlen):
19056: (8)                  r"""
19057: (8)                  Class-aware wrapper for `::fz_eval_function()`.
19058: (8)                  This method has out-params. Python/C# wrappers look like:
19059: (12)                     `fz_eval_function(const float *in, int inlen, int outlen)` =>
float out
19060: (8)                  """
19061: (8)                  return _mupdf.FzFunction_fz_eval_function(self, _in, inlen, out,
outlen)
19062: (4)              def fz_function_size(self):
19063: (8)                  r""" Class-aware wrapper for `::fz_function_size()`."""
19064: (8)                  return _mupdf.FzFunction_fz_function_size(self)
19065: (4)              def __init__(self, *args):
19066: (8)                  r"""
```

```
19067: (8)                  *Overload 1:*
19068: (8)                  == Constructors.  Constructor using `fz_new_function_of_size()`.
19069: (8)                  |
19070: (8)                  *Overload 2:*
19071: (8)                  Copy constructor using `fz_keep_function()`.
19072: (8)                  |
19073: (8)                  *Overload 3:*
19074: (8)                  Default constructor, sets `m_internal` to null.
19075: (8)                  |
19076: (8)                  *Overload 4:*
19077: (8)                  Constructor using raw copy of pre-existing `::fz_function`.
19078: (8)                  """
19079: (8)                  _mupdf.FzFunction_swiginit(self, _mupdf.new_FzFunction(*args))
19080: (4)              __swig_destroy__ = _mupdf.delete_FzFunction
19081: (4)              def m_internal_value(self):
19082: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19083: (8)                  return _mupdf.FzFunction_m_internal_value(self)
19084: (4)              m_internal = property(_mupdf.FzFunction_m_internal_get,
_mupdf.FzFunction_m_internal_set, doc=r""" Pointer to wrapped data.""")
19085: (4)              s_num_instances = property(_mupdf.FzFunction_s_num_instances_get,
_mupdf.FzFunction_s_num_instances_set)
19086: (0)          _mupdf.FzFunction_swigregister(FzFunction)
19087: (0)          class FzGetoptLongOptions(object):
19088: (4)              r""" Wrapper class for struct `fz_getopt_long_options`. Not copyable or
assignable."""
19089: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19090: (4)              __repr__ = _swig_repr
19091: (4)              def __init__(self, *args):
19092: (8)                  r"""
19093: (8)                  *Overload 1:*
19094: (8)                  Default constructor, sets `m_internal` to null.
19095: (8)                  |
19096: (8)                  *Overload 2:*
19097: (8)                  Constructor using raw copy of pre-existing `::fz_getopt_long_options`.
19098: (8)                  """
19099: (8)                  _mupdf.FzGetoptLongOptions_swiginit(self,
_mupdf.new_FzGetoptLongOptions(*args))
19100: (4)              __swig_destroy__ = _mupdf.delete_FzGetoptLongOptions
19101: (4)              def m_internal_value(self):
19102: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19103: (8)                  return _mupdf.FzGetoptLongOptions_m_internal_value(self)
19104: (4)              m_internal = property(_mupdf.FzGetoptLongOptions_m_internal_get,
_mupdf.FzGetoptLongOptions_m_internal_set, doc=r""" Pointer to wrapped data.""")
19105: (4)              s_num_instances = property(_mupdf.FzGetoptLongOptions_s_num_instances_get,
_mupdf.FzGetoptLongOptions_s_num_instances_set)
19106: (0)          _mupdf.FzGetoptLongOptions_swigregister(FzGetoptLongOptions)
19107: (0)          class FzGlyph(object):
19108: (4)              r"""
19109: (4)              Wrapper class for struct `fz_glyph`.
19110: (4)              Glyphs represent a run length encoded set of pixels for a 2
19111: (4)              dimensional region of a plane.
19112: (4)              """
19113: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19114: (4)              __repr__ = _swig_repr
19115: (4)              def fz_glyph_bbox(self):
19116: (8)                  r"""
19117: (8)                  Class-aware wrapper for `::fz_glyph_bbox()`.
19118: (12)                     Return the bounding box of the glyph in pixels.
19119: (8)                  """
19120: (8)                  return _mupdf.FzGlyph_fz_glyph_bbox(self)
19121: (4)              def fz_glyph_bbox_no_ctx(self):
19122: (8)                  r""" Class-aware wrapper for `::fz_glyph_bbox_no_ctx()`."""
19123: (8)                  return _mupdf.FzGlyph_fz_glyph_bbox_no_ctx(self)
19124: (4)              def fz_glyph_height(self):
19125: (8)                  r"""
```

```
19126: (8)                  Class-aware wrapper for `::fz_glyph_height()`.
19127: (12)                     Return the height of the glyph in pixels.
19128: (8)                  """
19129: (8)                  return _mupdf.FzGlyph_fz_glyph_height(self)
19130: (4)              def fz_glyph_width(self):
19131: (8)                  r"""
19132: (8)                  Class-aware wrapper for `::fz_glyph_width()`.
19133: (12)                     Return the width of the glyph in pixels.
19134: (8)                  """
19135: (8)                  return _mupdf.FzGlyph_fz_glyph_width(self)
19136: (4)              def __init__(self, *args):
19137: (8)                  r"""
19138: (8)                  *Overload 1:*
19139: (8)                  Copy constructor using `fz_keep_glyph()`.
19140: (8)                  |
19141: (8)                  *Overload 2:*
19142: (8)                  Default constructor, sets `m_internal` to null.
19143: (8)                  |
19144: (8)                  *Overload 3:*
19145: (8)                  Constructor using raw copy of pre-existing `::fz_glyph`.
19146: (8)                  """
19147: (8)                  _mupdf.FzGlyph_swiginit(self, _mupdf.new_FzGlyph(*args))
19148: (4)              __swig_destroy__ = _mupdf.delete_FzGlyph
19149: (4)              def m_internal_value(self):
19150: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19151: (8)                  return _mupdf.FzGlyph_m_internal_value(self)
19152: (4)              m_internal = property(_mupdf.FzGlyph_m_internal_get,
_mupdf.FzGlyph_m_internal_set, doc=r""" Pointer to wrapped data.""")
19153: (4)              s_num_instances = property(_mupdf.FzGlyph_s_num_instances_get,
_mupdf.FzGlyph_s_num_instances_set)
19154: (0)          _mupdf.FzGlyph_swigregister(FzGlyph)
19155: (0)          class FzGlyphCache(object):
19156: (4)              r""" Wrapper class for struct `fz_glyph_cache`. Not copyable or
assignable."""
19157: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19158: (4)              __repr__ = _swig_repr
19159: (4)              def __init__(self, *args):
19160: (8)                  r"""
19161: (8)                  *Overload 1:*
19162: (8)                  Default constructor, sets `m_internal` to null.
19163: (8)                  |
19164: (8)                  *Overload 2:*
19165: (8)                  Constructor using raw copy of pre-existing `::fz_glyph_cache`.
19166: (8)                  """
19167: (8)                  _mupdf.FzGlyphCache_swiginit(self, _mupdf.new_FzGlyphCache(*args))
19168: (4)              __swig_destroy__ = _mupdf.delete_FzGlyphCache
19169: (4)              def m_internal_value(self):
19170: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19171: (8)                  return _mupdf.FzGlyphCache_m_internal_value(self)
19172: (4)              m_internal = property(_mupdf.FzGlyphCache_m_internal_get,
_mupdf.FzGlyphCache_m_internal_set, doc=r""" Pointer to wrapped data.""")
19173: (4)              s_num_instances = property(_mupdf.FzGlyphCache_s_num_instances_get,
_mupdf.FzGlyphCache_s_num_instances_set)
19174: (0)          _mupdf.FzGlyphCache_swigregister(FzGlyphCache)
19175: (0)          class FzHalftone(object):
19176: (4)              r"""
19177: (4)              Wrapper class for struct `fz_halftone`.
19178: (4)              A halftone is a set of threshold tiles, one per component. Each
19179: (4)              threshold tile is a pixmap, possibly of varying sizes and
19180: (4)              phases. Currently, we only provide one 'default' halftone tile
19181: (4)              for operating on 1 component plus alpha pixmaps (where the alpha
19182: (4)              is ignored). This is signified by a fz_halftone pointer to NULL.
19183: (4)              """
19184: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19185: (4)              __repr__ = _swig_repr
```

```
19186: (4)              def __init__(self, *args):
19187: (8)                  r"""
19188: (8)                  *Overload 1:*
19189: (8)                  Copy constructor using `fz_keep_halftone()`.
19190: (8)                  |
19191: (8)                  *Overload 2:*
19192: (8)                  Constructor using raw copy of pre-existing `::fz_halftone`.
19193: (8)                  |
19194: (8)                  *Overload 3:*
19195: (8)                  Constructor using raw copy of pre-existing `::fz_halftone`.
19196: (8)                  """
19197: (8)                  _mupdf.FzHalftone_swiginit(self, _mupdf.new_FzHalftone(*args))
19198: (4)              __swig_destroy__ = _mupdf.delete_FzHalftone
19199: (4)              def m_internal_value(self):
19200: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19201: (8)                  return _mupdf.FzHalftone_m_internal_value(self)
19202: (4)              m_internal = property(_mupdf.FzHalftone_m_internal_get,
_mupdf.FzHalftone_m_internal_set, doc=r""" Pointer to wrapped data.""")
19203: (4)              s_num_instances = property(_mupdf.FzHalftone_s_num_instances_get,
_mupdf.FzHalftone_s_num_instances_set)
19204: (0)          _mupdf.FzHalftone_swigregister(FzHalftone)
19205: (0)          class FzHashTable(object):
19206: (4)              r"""
19207: (4)              Wrapper class for struct `fz_hash_table`. Not copyable or assignable.
19208: (4)              Generic hash-table with fixed-length keys.
19209: (4)              The keys and values are NOT reference counted by the hash table.
19210: (4)              Callers are responsible for taking care the reference counts are
19211: (4)              correct. Inserting a duplicate entry will NOT overwrite the old
19212: (4)              value, and will return the old value.
19213: (4)              The drop_val callback function is only used to release values
19214: (4)              when the hash table is destroyed.
19215: (4)              """
19216: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19217: (4)              __repr__ = _swig_repr
19218: (4)              def fz_hash_filter(self, state, callback):
19219: (8)                  r"""
19220: (8)                  Class-aware wrapper for `::fz_hash_filter()`.
19221: (12)                     Iterate over the entries in a hash table, removing all the ones
where callback returns true.
19222: (12)                     Does NOT free the value of the entry, so the caller is expected to
take care of this.
19223: (8)                  """
19224: (8)                  return _mupdf.FzHashTable_fz_hash_filter(self, state, callback)
19225: (4)              def fz_hash_find(self, key):
19226: (8)                  r"""
19227: (8)                  Class-aware wrapper for `::fz_hash_find()`.
19228: (12)                     Search for a matching hash within the table, and return the
19229: (12)                     associated value.
19230: (8)                  """
19231: (8)                  return _mupdf.FzHashTable_fz_hash_find(self, key)
19232: (4)              def fz_hash_for_each(self, state, callback):
19233: (8)                  r"""
19234: (8)                  Class-aware wrapper for `::fz_hash_for_each()`.
19235: (12)                     Iterate over the entries in a hash table.
19236: (8)                  """
19237: (8)                  return _mupdf.FzHashTable_fz_hash_for_each(self, state, callback)
19238: (4)              def fz_hash_insert(self, key, val):
19239: (8)                  r"""
19240: (8)                  Class-aware wrapper for `::fz_hash_insert()`.
19241: (12)                     Insert a new key/value pair into the hash table.
19242: (12)                     If an existing entry with the same key is found, no change is
19243: (12)                     made to the hash table, and a pointer to the existing value is
19244: (12)                     returned.
19245: (12)                     If no existing entry with the same key is found, ownership of
19246: (12)                     val passes in, key is copied, and NULL is returned.
19247: (8)                  """
19248: (8)                  return _mupdf.FzHashTable_fz_hash_insert(self, key, val)
```

```
19249: (4)              def fz_hash_remove(self, key):
19250: (8)                  r"""
19251: (8)                  Class-aware wrapper for `::fz_hash_remove()`.
19252: (12)                     Remove the entry for a given key.
19253: (12)                     The value is NOT freed, so the caller is expected to take care
19254: (12)                     of this.
19255: (8)                  """
19256: (8)                  return _mupdf.FzHashTable_fz_hash_remove(self, key)
19257: (4)              def __init__(self, *args):
19258: (8)                  r"""
19259: (8)                  *Overload 1:*
19260: (9)                   == Constructors.  Constructor using `fz_new_hash_table()`.
19261: (16)                         Create a new hash table.
19262: (16)                         initialsize: The initial size of the hashtable. The hashtable
19263: (16)                         may grow (double in size) if it starts to get crowded (80%
19264: (16)                         full).
19265: (16)                         keylen: byte length for each key.
19266: (16)                         lock: -1 for no lock, otherwise the FZ_LOCK to use to protect
19267: (16)                         this table.
19268: (16)                         drop_val: Function to use to destroy values on table drop.
19269: (8)                  |
19270: (8)                  *Overload 2:*
19271: (9)                   Default constructor, sets `m_internal` to null.
19272: (8)                  |
19273: (8)                  *Overload 3:*
19274: (9)                   Constructor using raw copy of pre-existing `::fz_hash_table`.
19275: (8)                  """
19276: (8)                  _mupdf.FzHashTable_swiginit(self, _mupdf.new_FzHashTable(*args))
19277: (4)              __swig_destroy__ = _mupdf.delete_FzHashTable
19278: (4)              def m_internal_value(self):
19279: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19280: (8)                  return _mupdf.FzHashTable_m_internal_value(self)
19281: (4)              m_internal = property(_mupdf.FzHashTable_m_internal_get,
_mupdf.FzHashTable_m_internal_set, doc=r""" Pointer to wrapped data.""")
19282: (4)              s_num_instances = property(_mupdf.FzHashTable_s_num_instances_get,
_mupdf.FzHashTable_s_num_instances_set)
19283: (0)          _mupdf.FzHashTable_swigregister(FzHashTable)
19284: (0)          class FzIccProfile(object):
19285: (4)              r"""
19286: (4)              Wrapper class for struct `fz_icc_profile`. Not copyable or assignable.
19287: (4)              Opaque type for an ICC Profile.
19288: (4)              """
19289: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19290: (4)              __repr__ = _swig_repr
19291: (4)              def __init__(self, *args):
19292: (8)                  r"""
19293: (8)                  *Overload 1:*
19294: (8)                  Default constructor, sets `m_internal` to null.
19295: (8)                  |
19296: (8)                  *Overload 2:*
19297: (8)                  Constructor using raw copy of pre-existing `::fz_icc_profile`.
19298: (8)                  """
19299: (8)                  _mupdf.FzIccProfile_swiginit(self, _mupdf.new_FzIccProfile(*args))
19300: (4)              __swig_destroy__ = _mupdf.delete_FzIccProfile
19301: (4)              def m_internal_value(self):
19302: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19303: (8)                  return _mupdf.FzIccProfile_m_internal_value(self)
19304: (4)              m_internal = property(_mupdf.FzIccProfile_m_internal_get,
_mupdf.FzIccProfile_m_internal_set, doc=r""" Pointer to wrapped data.""")
19305: (4)              s_num_instances = property(_mupdf.FzIccProfile_s_num_instances_get,
_mupdf.FzIccProfile_s_num_instances_set)
19306: (0)          _mupdf.FzIccProfile_swigregister(FzIccProfile)
19307: (0)          class FzImage(object):
19308: (4)              r"""
19309: (4)              Wrapper class for struct `fz_image`.
19310: (4)              Images are storable objects from which we can obtain fz_pixmaps.
```

```
(4)             These may be implemented as simple wrappers around a pixmap, or
(4)             as more complex things that decode at different subsample
(4)             settings on demand.
(4)             """
(4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v), doc="The membership flag")
(4)             __repr__ = _swig_repr
(4)             def fz_compressed_image_buffer(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_compressed_image_buffer()`.
(12)                    Retrieve the underlying compressed data for an image.
(12)                    Returns a pointer to the underlying data buffer for an image,
(12)                    or NULL if this image is not based upon a compressed data
(12)                    buffer.
(12)                    This is not a reference counted structure, so no reference is
(12)                    returned. Lifespan is limited to that of the image itself.
(8)                 """
(8)                 return _mupdf.FzImage_fz_compressed_image_buffer(self)
(4)             def fz_compressed_image_type(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_compressed_image_type()`.
(12)                    Return the type of a compressed image.
(12)                    Any non-compressed image will have the type returned as UNKNOWN.
(8)                 """
(8)                 return _mupdf.FzImage_fz_compressed_image_type(self)
(4)             def fz_get_pixmap_from_image(self, subarea, ctm, w, h):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_get_pixmap_from_image()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_get_pixmap_from_image(const ::fz_irect *subarea, ::fz_matrix *ctm)` => `(fz_pixmap *, int w, int h)`
(12)                    Called to get a handle to a pixmap from an image.
(12)                    image: The image to retrieve a pixmap from.
(12)                    subarea: The subarea of the image that we actually care about
(12)                    (or NULL to indicate the whole image).
(12)                    ctm: Optional, unless subarea is given. If given, then on
(12)                    entry this is the transform that will be applied to the complete
(12)                    image. It should be updated on exit to the transform to apply to
(12)                    the given subarea of the image. This is used to calculate the
(12)                    desired width/height for subsampling.
(12)                    w: If non-NULL, a pointer to an int to be updated on exit to the
(12)                    width (in pixels) that the scaled output will cover.
(12)                    h: If non-NULL, a pointer to an int to be updated on exit to the
(12)                    height (in pixels) that the scaled output will cover.
(12)                    Returns a non NULL kept pixmap pointer. May throw exceptions.
(8)                 """
(8)                 return _mupdf.FzImage_fz_get_pixmap_from_image(self, subarea, ctm, w, h)
(4)             def fz_get_unscaled_pixmap_from_image(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_get_unscaled_pixmap_from_image()`.
(12)                    Calls fz_get_pixmap_from_image() with ctm, subarea, w and h all set to NULL.
(8)                 """
(8)                 return _mupdf.FzImage_fz_get_unscaled_pixmap_from_image(self)
(4)             def fz_image_orientation(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_image_orientation()`.
(12)                    Request the natural orientation of an image.
(12)                    This is for images (such as JPEG) that can contain internal
(12)                    specifications of rotation/flips. This is ignored by all the
(12)                    internal decode/rendering routines, but can be used by callers
(12)                    (such as the image document handler) to respect such
(12)                    specifications.
(12)                    The values used by MuPDF are as follows, with the equivalent
(12)                    Exif specifications given for information:
(12)                    0: Undefined
(12)                    1: 0 degree ccw rotation. (Exif = 1)
(12)                    2: 90 degree ccw rotation. (Exif = 8)
```

```
19376: (12)                     3: 180 degree ccw rotation. (Exif = 3)
19377: (12)                     4: 270 degree ccw rotation. (Exif = 6)
19378: (12)                     5: flip on X. (Exif = 2)
19379: (12)                     6: flip on X, then rotate ccw by 90 degrees. (Exif = 5)
19380: (12)                     7: flip on X, then rotate ccw by 180 degrees. (Exif = 4)
19381: (12)                     8: flip on X, then rotate ccw by 270 degrees. (Exif = 7)
19382: (8)                  """
19383: (8)                  return _mupdf.FzImage_fz_image_orientation(self)
19384: (4)              def fz_image_orientation_matrix(self):
19385: (8)                  r""" Class-aware wrapper for `::fz_image_orientation_matrix()`."""
19386: (8)                  return _mupdf.FzImage_fz_image_orientation_matrix(self)
19387: (4)              def fz_image_resolution(self, xres, yres):
19388: (8)                  r"""
19389: (8)                  Class-aware wrapper for `::fz_image_resolution()`.
19390: (8)                  This method has out-params. Python/C# wrappers look like:
19391: (12)                     `fz_image_resolution()` => `(int xres, int yres)`
19392: (12)                     Request the natural resolution
19393: (12)                     of an image.
19394: (12)                     xres, yres: Pointers to ints to be updated with the
19395: (12)                     natural resolution of an image (or a sensible default
19396: (12)                     if not encoded).
19397: (8)                  """
19398: (8)                  return _mupdf.FzImage_fz_image_resolution(self, xres, yres)
19399: (4)              def fz_image_size(self):
19400: (8)                  r"""
19401: (8)                  Class-aware wrapper for `::fz_image_size()`.
19402: (12)                     Return the size of the storage used by an image.
19403: (8)                  """
19404: (8)                  return _mupdf.FzImage_fz_image_size(self)
19405: (4)              def fz_new_buffer_from_image_as_jpeg(self, color_params, quality,
invert_cmyk):
19406: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_image_as_jpeg()`."""
19407: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_jpeg(self,
color_params, quality, invert_cmyk)
19408: (4)              def fz_new_buffer_from_image_as_jpx(self, color_params, quality):
19409: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_jpx()`."""
19410: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_jpx(self,
color_params, quality)
19411: (4)              def fz_new_buffer_from_image_as_pam(self, color_params):
19412: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pam()`."""
19413: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_pam(self,
color_params)
19414: (4)              def fz_new_buffer_from_image_as_png(self, color_params):
19415: (8)                  r"""
19416: (8)                  Class-aware wrapper for `::fz_new_buffer_from_image_as_png()`.
19417: (12)                     Reencode a given image as a PNG into a buffer.
19418: (12)                     Ownership of the buffer is returned.
19419: (8)                  """
19420: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_png(self,
color_params)
19421: (4)              def fz_new_buffer_from_image_as_pnm(self, color_params):
19422: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pnm()`."""
19423: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_pnm(self,
color_params)
19424: (4)              def fz_new_buffer_from_image_as_psd(self, color_params):
19425: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_psd()`."""
19426: (8)                  return _mupdf.FzImage_fz_new_buffer_from_image_as_psd(self,
color_params)
19427: (4)              def __init__(self, *args):
19428: (8)                  r"""
19429: (8)                  *Overload 1:*
19430: (9)                   == Constructors.  Constructor using `fz_new_image_from_buffer()`.
19431: (16)                         Create a new image from a
19432: (16)                         buffer of data, inferring its type from the format
19433: (16)                         of the data.
19434: (8)                  |
19435: (8)                  *Overload 2:*
19436: (9)                   Constructor using `fz_new_image_from_compressed_buffer()`.
```

```
19437: (16)                         Create an image based on
19438: (16)                         the data in the supplied compressed buffer.
19439: (16)                         w,h: Width and height of the created image.
19440: (16)                         bpc: Bits per component.
19441: (16)                         colorspace: The colorspace (determines the number of
components,
19442: (16)                         and any color conversions required while decoding).
19443: (16)                         xres, yres: The X and Y resolutions respectively.
19444: (16)                         interpolate: 1 if interpolation should be used when decoding
19445: (16)                         this image, 0 otherwise.
19446: (16)                         imagemask: 1 if this is an imagemask (i.e. transparency bitmap
19447: (16)                         mask), 0 otherwise.
19448: (16)                         decode: NULL, or a pointer to to a decode array. The default
19449: (16)                         decode array is [0 1] (repeated n times, for n color
components).
19450: (16)                         colorkey: NULL, or a pointer to a colorkey array. The default
19451: (16)                         colorkey array is [0 255] (repeated n times, for n color
19452: (16)                         components).
19453: (16)                         buffer: Buffer of compressed data and compression parameters.
19454: (16)                         Ownership of this reference is passed in.
19455: (16)                         mask: NULL, or another image to use as a mask for this one.
19456: (16)                         A new reference is taken to this image. Supplying a masked
19457: (16)                         image as a mask to another image is illegal!
19458: (8)                   |
19459: (8)                   *Overload 3:*
19460: (9)                    Constructor using `fz_new_image_from_compressed_buffer2()`.  Swig-
friendly wrapper for fz_new_image_from_compressed_buffer(),
19461: (12)                       uses specified `decode` and `colorkey` if they are not null (in
which
19462: (12)                       case we assert that they have size
`2*fz_colorspace_n(colorspace)`).
19463: (8)                   |
19464: (8)                   *Overload 4:*
19465: (9)                    Constructor using `fz_new_image_from_display_list()`.
19466: (16)                         Create a new image from a display list.
19467: (16)                         w, h: The conceptual width/height of the image.
19468: (16)                         transform: The matrix that needs to be applied to the given
19469: (16)                         list to make it render to the unit square.
19470: (16)                         list: The display list.
19471: (8)                   |
19472: (8)                   *Overload 5:*
19473: (9)                    Constructor using `fz_new_image_from_file()`.
19474: (16)                         Create a new image from the contents
19475: (16)                         of a file, inferring its type from the format of the
19476: (16)                         data.
19477: (8)                   |
19478: (8)                   *Overload 6:*
19479: (9)                    Constructor using `fz_new_image_from_pixmap()`.
19480: (16)                         Create an image from the given
19481: (16)                         pixmap.
19482: (16)                         pixmap: The pixmap to base the image upon. A new reference
19483: (16)                         to this is taken.
19484: (16)                         mask: NULL, or another image to use as a mask for this one.
19485: (16)                         A new reference is taken to this image. Supplying a masked
19486: (16)                         image as a mask to another image is illegal!
19487: (8)                   |
19488: (8)                   *Overload 7:*
19489: (9)                    Constructor using `fz_new_image_from_svg()`.
19490: (16)                         Create a scalable image from an SVG document.
19491: (8)                   |
19492: (8)                   *Overload 8:*
19493: (9)                    Constructor using `fz_new_image_from_svg_xml()`.
19494: (16)                         Create a scalable image from an SVG document.
19495: (8)                   |
19496: (8)                   *Overload 9:*
19497: (9)                    Constructor using `fz_new_image_of_size()`.
19498: (16)                         Internal function to make a new fz_image structure
19499: (16)                         for a derived class.
19500: (16)                         w,h: Width and height of the created image.
```

```
19501: (16)                         bpc: Bits per component.
19502: (16)                         colorspace: The colorspace (determines the number of
components,
19503: (16)                         and any color conversions required while decoding).
19504: (16)                         xres, yres: The X and Y resolutions respectively.
19505: (16)                         interpolate: 1 if interpolation should be used when decoding
19506: (16)                         this image, 0 otherwise.
19507: (16)                         imagemask: 1 if this is an imagemask (i.e. transparent), 0
19508: (16)                         otherwise.
19509: (16)                         decode: NULL, or a pointer to to a decode array. The default
19510: (16)                         decode array is [0 1] (repeated n times, for n color
components).
19511: (16)                         colorkey: NULL, or a pointer to a colorkey array. The default
19512: (16)                         colorkey array is [0 255] (repeated n times, for n color
19513: (16)                         components).
19514: (16)                         mask: NULL, or another image to use as a mask for this one.
19515: (16)                         A new reference is taken to this image. Supplying a masked
19516: (16)                         image as a mask to another image is illegal!
19517: (16)                         size: The size of the required allocated structure (the size
of
19518: (16)                         the derived structure).
19519: (16)                         get: The function to be called to obtain a decoded pixmap.
19520: (16)                         get_size: The function to be called to return the storage size
19521: (16)                         used by this image.
19522: (16)                         drop: The function to be called to dispose of this image once
19523: (16)                         the last reference is dropped.
19524: (16)                         Returns a pointer to an allocated structure of the required
size,
19525: (16)                         with the first sizeof(fz_image) bytes initialised as
appropriate
19526: (16)                         given the supplied parameters, and the other bytes set to
zero.
19527: (8)                      |
19528: (8)                      *Overload 10:*
19529: (9)                       Copy constructor using `fz_keep_image()`.
19530: (8)                      |
19531: (8)                      *Overload 11:*
19532: (9)                       Default constructor, sets `m_internal` to null.
19533: (8)                      |
19534: (8)                      *Overload 12:*
19535: (9)                       Constructor using raw copy of pre-existing `::fz_image`.
19536: (8)                      """
19537: (8)                      _mupdf.FzImage_swiginit(self, _mupdf.new_FzImage(*args))
19538: (4)                  def key_storable(self):
19539: (8)                      return _mupdf.FzImage_key_storable(self)
19540: (4)                  def w(self):
19541: (8)                      return _mupdf.FzImage_w(self)
19542: (4)                  def h(self):
19543: (8)                      return _mupdf.FzImage_h(self)
19544: (4)                  def n(self):
19545: (8)                      return _mupdf.FzImage_n(self)
19546: (4)                  def bpc(self):
19547: (8)                      return _mupdf.FzImage_bpc(self)
19548: (4)                  def imagemask(self):
19549: (8)                      return _mupdf.FzImage_imagemask(self)
19550: (4)                  def interpolate(self):
19551: (8)                      return _mupdf.FzImage_interpolate(self)
19552: (4)                  def use_colorkey(self):
19553: (8)                      return _mupdf.FzImage_use_colorkey(self)
19554: (4)                  def use_decode(self):
19555: (8)                      return _mupdf.FzImage_use_decode(self)
19556: (4)                  def decoded(self):
19557: (8)                      return _mupdf.FzImage_decoded(self)
19558: (4)                  def scalable(self):
19559: (8)                      return _mupdf.FzImage_scalable(self)
19560: (4)                  def orientation(self):
19561: (8)                      return _mupdf.FzImage_orientation(self)
19562: (4)                  def mask(self):
19563: (8)                      return _mupdf.FzImage_mask(self)
```

```
19564: (4)              def xres(self):
19565: (8)                  return _mupdf.FzImage_xres(self)
19566: (4)              def yres(self):
19567: (8)                  return _mupdf.FzImage_yres(self)
19568: (4)              def colorspace(self):
19569: (8)                  return _mupdf.FzImage_colorspace(self)
19570: (4)              def colorkey(self):
19571: (8)                  return _mupdf.FzImage_colorkey(self)
19572: (4)              def decode(self):
19573: (8)                  return _mupdf.FzImage_decode(self)
19574: (4)              __swig_destroy__ = _mupdf.delete_FzImage
19575: (4)              def m_internal_value(self):
19576: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19577: (8)                  return _mupdf.FzImage_m_internal_value(self)
19578: (4)              m_internal = property(_mupdf.FzImage_m_internal_get,
_mupdf.FzImage_m_internal_set, doc=r""" Pointer to wrapped data.""")
19579: (4)              s_num_instances = property(_mupdf.FzImage_s_num_instances_get,
_mupdf.FzImage_s_num_instances_set)
19580: (0)          _mupdf.FzImage_swigregister(FzImage)
19581: (0)          class FzInstallLoadSystemFontFuncsArgs(object):
19582: (4)              r"""
19583: (5)               Wrapper class for struct `fz_install_load_system_font_funcs_args`.
19584: (4)              Extra struct containing fz_install_load_system_font_funcs()'s args,
19585: (4)              which we wrap with virtual_fnptrs set to allow use from Python/C# via
19586: (4)              Swig Directors.
19587: (4)              """
19588: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19589: (4)              __repr__ = _swig_repr
19590: (4)              def fz_install_load_system_font_funcs2(self):
19591: (8)                  r"""
19592: (8)                  Class-aware wrapper for `::fz_install_load_system_font_funcs2()`.
19593: (8)                  Alternative to fz_install_load_system_font_funcs() that takes args in
a
19594: (8)                  struct, to allow use from Python/C# via Swig Directors.
19595: (8)                  """
19596: (8)                  return
_mupdf.FzInstallLoadSystemFontFuncsArgs_fz_install_load_system_font_funcs2(self)
19597: (4)              def __init__(self, *args):
19598: (8)                  r"""
19599: (8)                  *Overload 1:*
19600: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
19601: (8)                  |
19602: (8)                  *Overload 2:*
19603: (8)                  Constructor using raw copy of pre-existing
`::fz_install_load_system_font_funcs_args`.
19604: (8)                  """
19605: (8)                  _mupdf.FzInstallLoadSystemFontFuncsArgs_swiginit(self,
_mupdf.new_FzInstallLoadSystemFontFuncsArgs(*args))
19606: (4)              __swig_destroy__ = _mupdf.delete_FzInstallLoadSystemFontFuncsArgs
19607: (4)              m_internal =
property(_mupdf.FzInstallLoadSystemFontFuncsArgs_m_internal_get,
_mupdf.FzInstallLoadSystemFontFuncsArgs_m_internal_set)
19608: (4)              s_num_instances =
property(_mupdf.FzInstallLoadSystemFontFuncsArgs_s_num_instances_get,
_mupdf.FzInstallLoadSystemFontFuncsArgs_s_num_instances_set, doc=r""" Wrapped data is held by
value.""")
19609: (4)              def to_string(self):
19610: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
19611: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs_to_string(self)
19612: (4)              def __eq__(self, rhs):
19613: (8)                  r""" Comparison method."""
19614: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs___eq__(self, rhs)
19615: (4)              def __ne__(self, rhs):
19616: (8)                  r""" Comparison method."""
19617: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs___ne__(self, rhs)
```

```
19618: (0)
_mupdf.FzInstallLoadSystemFontFuncsArgs_swigregister(FzInstallLoadSystemFontFuncsArgs)
19619: (0)          class FzInstallLoadSystemFontFuncsArgs2(FzInstallLoadSystemFontFuncsArgs):
19620: (4)              r""" Wrapper class for struct fz_install_load_system_font_funcs_args with
virtual fns for each fnptr; this is for use as a SWIG Director class."""
19621: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19622: (4)              __repr__ = _swig_repr
19623: (4)              def __init__(self):
19624: (8)                  r""" == Constructor."""
19625: (8)                  if self.__class__ == FzInstallLoadSystemFontFuncsArgs2:
19626: (12)                     _self = None
19627: (8)                  else:
19628: (12)                     _self = self
19629: (8)                  _mupdf.FzInstallLoadSystemFontFuncsArgs2_swiginit(self,
_mupdf.new_FzInstallLoadSystemFontFuncsArgs2(_self, ))
19630: (4)              def use_virtual_f(self, use=True):
19631: (8)                  r"""
19632: (8)                  These methods set the function pointers in *m_internal
19633: (8)                  to point to internal callbacks that call our virtual methods.
19634: (8)                  """
19635: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs2_use_virtual_f(self,
use)
19636: (4)              def use_virtual_f_cjk(self, use=True):
19637: (8)                  return
_mupdf.FzInstallLoadSystemFontFuncsArgs2_use_virtual_f_cjk(self, use)
19638: (4)              def use_virtual_f_fallback(self, use=True):
19639: (8)                  return
_mupdf.FzInstallLoadSystemFontFuncsArgs2_use_virtual_f_fallback(self, use)
19640: (4)              def f(self, arg_0, arg_1, arg_2, arg_3, arg_4):
19641: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
19642: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs2_f(self, arg_0, arg_1,
arg_2, arg_3, arg_4)
19643: (4)              def f_cjk(self, arg_0, arg_1, arg_2, arg_3):
19644: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs2_f_cjk(self, arg_0,
arg_1, arg_2, arg_3)
19645: (4)              def f_fallback(self, arg_0, arg_1, arg_2, arg_3, arg_4, arg_5):
19646: (8)                  return _mupdf.FzInstallLoadSystemFontFuncsArgs2_f_fallback(self,
arg_0, arg_1, arg_2, arg_3, arg_4, arg_5)
19647: (4)              __swig_destroy__ = _mupdf.delete_FzInstallLoadSystemFontFuncsArgs2
19648: (4)              def __disown__(self):
19649: (8)                  self.this.disown()
19650: (8)                  _mupdf.disown_FzInstallLoadSystemFontFuncsArgs2(self)
19651: (8)                  return weakref.proxy(self)
19652: (0)
_mupdf.FzInstallLoadSystemFontFuncsArgs2_swigregister(FzInstallLoadSystemFontFuncsArgs2)
19653: (0)          class FzInt2(object):
19654: (4)              r""" Wrapper class for struct `fz_int2`. Not copyable or assignable."""
19655: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19656: (4)              __repr__ = _swig_repr
19657: (4)              def __init__(self, *args):
19658: (8)                  r"""
19659: (8)                  *Overload 1:*
19660: (8)                  Default constructor, sets `m_internal` to null.
19661: (8)                  |
19662: (8)                  *Overload 2:*
19663: (8)                  Constructor using raw copy of pre-existing `::fz_int2`.
19664: (8)                  """
19665: (8)                  _mupdf.FzInt2_swiginit(self, _mupdf.new_FzInt2(*args))
19666: (4)              __swig_destroy__ = _mupdf.delete_FzInt2
19667: (4)              def m_internal_value(self):
19668: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19669: (8)                  return _mupdf.FzInt2_m_internal_value(self)
19670: (4)              m_internal = property(_mupdf.FzInt2_m_internal_get,
_mupdf.FzInt2_m_internal_set, doc=r""" Pointer to wrapped data.""")
19671: (4)              s_num_instances = property(_mupdf.FzInt2_s_num_instances_get,
```

```
_mupdf.FzInt2_s_num_instances_set)
19672: (0)          _mupdf.FzInt2_swigregister(FzInt2)
19673: (0)          class FzInt2Heap(object):
19674: (4)              r""" Wrapper class for struct `fz_int2_heap`. Not copyable or
assignable."""
19675: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19676: (4)              __repr__ = _swig_repr
19677: (4)              def fz_int2_heap_insert(self, v):
19678: (8)                  r""" Class-aware wrapper for `::fz_int2_heap_insert()`."""
19679: (8)                  return _mupdf.FzInt2Heap_fz_int2_heap_insert(self, v)
19680: (4)              def fz_int2_heap_sort(self):
19681: (8)                  r""" Class-aware wrapper for `::fz_int2_heap_sort()`."""
19682: (8)                  return _mupdf.FzInt2Heap_fz_int2_heap_sort(self)
19683: (4)              def fz_int2_heap_uniq(self):
19684: (8)                  r""" Class-aware wrapper for `::fz_int2_heap_uniq()`."""
19685: (8)                  return _mupdf.FzInt2Heap_fz_int2_heap_uniq(self)
19686: (4)              def __init__(self, *args):
19687: (8)                  r"""
19688: (8)                  *Overload 1:*
19689: (8)                  Default constructor, sets `m_internal` to null.
19690: (8)                  |
19691: (8)                  *Overload 2:*
19692: (8)                  Constructor using raw copy of pre-existing `::fz_int2_heap`.
19693: (8)                  """
19694: (8)                  _mupdf.FzInt2Heap_swiginit(self, _mupdf.new_FzInt2Heap(*args))
19695: (4)              __swig_destroy__ = _mupdf.delete_FzInt2Heap
19696: (4)              def m_internal_value(self):
19697: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19698: (8)                  return _mupdf.FzInt2Heap_m_internal_value(self)
19699: (4)              m_internal = property(_mupdf.FzInt2Heap_m_internal_get,
_mupdf.FzInt2Heap_m_internal_set, doc=r""" Pointer to wrapped data.""")
19700: (4)              s_num_instances = property(_mupdf.FzInt2Heap_s_num_instances_get,
_mupdf.FzInt2Heap_s_num_instances_set)
19701: (0)          _mupdf.FzInt2Heap_swigregister(FzInt2Heap)
19702: (0)          class FzIntHeap(object):
19703: (4)              r""" Wrapper class for struct `fz_int_heap`. Not copyable or
assignable."""
19704: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19705: (4)              __repr__ = _swig_repr
19706: (4)              def fz_int_heap_insert(self, v):
19707: (8)                  r""" Class-aware wrapper for `::fz_int_heap_insert()`."""
19708: (8)                  return _mupdf.FzIntHeap_fz_int_heap_insert(self, v)
19709: (4)              def fz_int_heap_sort(self):
19710: (8)                  r""" Class-aware wrapper for `::fz_int_heap_sort()`."""
19711: (8)                  return _mupdf.FzIntHeap_fz_int_heap_sort(self)
19712: (4)              def fz_int_heap_uniq(self):
19713: (8)                  r""" Class-aware wrapper for `::fz_int_heap_uniq()`."""
19714: (8)                  return _mupdf.FzIntHeap_fz_int_heap_uniq(self)
19715: (4)              def __init__(self, *args):
19716: (8)                  r"""
19717: (8)                  *Overload 1:*
19718: (8)                  Default constructor, sets `m_internal` to null.
19719: (8)                  |
19720: (8)                  *Overload 2:*
19721: (8)                  Constructor using raw copy of pre-existing `::fz_int_heap`.
19722: (8)                  """
19723: (8)                  _mupdf.FzIntHeap_swiginit(self, _mupdf.new_FzIntHeap(*args))
19724: (4)              __swig_destroy__ = _mupdf.delete_FzIntHeap
19725: (4)              def m_internal_value(self):
19726: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19727: (8)                  return _mupdf.FzIntHeap_m_internal_value(self)
19728: (4)              m_internal = property(_mupdf.FzIntHeap_m_internal_get,
_mupdf.FzIntHeap_m_internal_set, doc=r""" Pointer to wrapped data.""")
19729: (4)              s_num_instances = property(_mupdf.FzIntHeap_s_num_instances_get,
_mupdf.FzIntHeap_s_num_instances_set)
```

```
19730: (0)          _mupdf.FzIntHeap_swigregister(FzIntHeap)
19731: (0)          class FzIntptr(object):
19732: (4)              r""" Wrapper class for struct `fz_intptr`. Not copyable or assignable."""
19733: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19734: (4)              __repr__ = _swig_repr
19735: (4)              def __init__(self, *args):
19736: (8)                  r"""
19737: (8)                  *Overload 1:*
19738: (8)                  Default constructor, sets `m_internal` to null.
19739: (8)                  |
19740: (8)                  *Overload 2:*
19741: (8)                  Constructor using raw copy of pre-existing `::fz_intptr`.
19742: (8)                  """
19743: (8)                  _mupdf.FzIntptr_swiginit(self, _mupdf.new_FzIntptr(*args))
19744: (4)              __swig_destroy__ = _mupdf.delete_FzIntptr
19745: (4)              def m_internal_value(self):
19746: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19747: (8)                  return _mupdf.FzIntptr_m_internal_value(self)
19748: (4)              m_internal = property(_mupdf.FzIntptr_m_internal_get,
_mupdf.FzIntptr_m_internal_set, doc=r""" Pointer to wrapped data.""")
19749: (4)              s_num_instances = property(_mupdf.FzIntptr_s_num_instances_get,
_mupdf.FzIntptr_s_num_instances_set)
19750: (0)          _mupdf.FzIntptr_swigregister(FzIntptr)
19751: (0)          class FzIntptrHeap(object):
19752: (4)              r""" Wrapper class for struct `fz_intptr_heap`. Not copyable or
assignable."""
19753: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19754: (4)              __repr__ = _swig_repr
19755: (4)              def fz_intptr_heap_insert(self, v):
19756: (8)                  r""" Class-aware wrapper for `::fz_intptr_heap_insert()`."""
19757: (8)                  return _mupdf.FzIntptrHeap_fz_intptr_heap_insert(self, v)
19758: (4)              def fz_intptr_heap_sort(self):
19759: (8)                  r""" Class-aware wrapper for `::fz_intptr_heap_sort()`."""
19760: (8)                  return _mupdf.FzIntptrHeap_fz_intptr_heap_sort(self)
19761: (4)              def fz_intptr_heap_uniq(self):
19762: (8)                  r""" Class-aware wrapper for `::fz_intptr_heap_uniq()`."""
19763: (8)                  return _mupdf.FzIntptrHeap_fz_intptr_heap_uniq(self)
19764: (4)              def __init__(self, *args):
19765: (8)                  r"""
19766: (8)                  *Overload 1:*
19767: (8)                  Default constructor, sets `m_internal` to null.
19768: (8)                  |
19769: (8)                  *Overload 2:*
19770: (8)                  Constructor using raw copy of pre-existing `::fz_intptr_heap`.
19771: (8)                  """
19772: (8)                  _mupdf.FzIntptrHeap_swiginit(self, _mupdf.new_FzIntptrHeap(*args))
19773: (4)              __swig_destroy__ = _mupdf.delete_FzIntptrHeap
19774: (4)              def m_internal_value(self):
19775: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19776: (8)                  return _mupdf.FzIntptrHeap_m_internal_value(self)
19777: (4)              m_internal = property(_mupdf.FzIntptrHeap_m_internal_get,
_mupdf.FzIntptrHeap_m_internal_set, doc=r""" Pointer to wrapped data.""")
19778: (4)              s_num_instances = property(_mupdf.FzIntptrHeap_s_num_instances_get,
_mupdf.FzIntptrHeap_s_num_instances_set)
19779: (0)          _mupdf.FzIntptrHeap_swigregister(FzIntptrHeap)
19780: (0)          class FzIrect(object):
19781: (4)              r"""
19782: (4)              Wrapper class for struct `fz_irect`.
19783: (4)              fz_irect is a rectangle using integers instead of floats.
19784: (4)              It's used in the draw device and for pixmap dimensions.
19785: (4)              """
19786: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19787: (4)              __repr__ = _swig_repr
19788: (4)              def fz_expand_irect(self, expand):
```

```
19789: (8)                  r""" Class-aware wrapper for `::fz_expand_irect()`."""
19790: (8)                  return _mupdf.FzIrect_fz_expand_irect(self, expand)
19791: (4)              def fz_intersect_irect(self, b):
19792: (8)                  r"""
19793: (8)                  Class-aware wrapper for `::fz_intersect_irect()`.
19794: (12)                     Compute intersection of two bounding boxes.
19795: (12)                     Similar to fz_intersect_rect but operates on two bounding
19796: (12)                     boxes instead of two rectangles.
19797: (8)                  """
19798: (8)                  return _mupdf.FzIrect_fz_intersect_irect(self, b)
19799: (4)              def fz_irect_height(self):
19800: (8)                  r"""
19801: (8)                  Class-aware wrapper for `::fz_irect_height()`.
19802: (12)                     Return the height of an irect. Invalid irects return 0.
19803: (8)                  """
19804: (8)                  return _mupdf.FzIrect_fz_irect_height(self)
19805: (4)              def fz_irect_width(self):
19806: (8)                  r"""
19807: (8)                  Class-aware wrapper for `::fz_irect_width()`.
19808: (12)                     Return the width of an irect. Invalid irects return 0.
19809: (8)                  """
19810: (8)                  return _mupdf.FzIrect_fz_irect_width(self)
19811: (4)              def fz_is_empty_irect(self):
19812: (8)                  r""" Class-aware wrapper for `::fz_is_empty_irect()`."""
19813: (8)                  return _mupdf.FzIrect_fz_is_empty_irect(self)
19814: (4)              def fz_is_infinite_irect(self):
19815: (8)                  r"""
19816: (8)                  Class-aware wrapper for `::fz_is_infinite_irect()`.
19817: (12)                     Check if an integer rectangle
19818: (12)                     is infinite.
19819: (8)                  """
19820: (8)                  return _mupdf.FzIrect_fz_is_infinite_irect(self)
19821: (4)              def fz_is_valid_irect(self):
19822: (8)                  r"""
19823: (8)                  Class-aware wrapper for `::fz_is_valid_irect()`.
19824: (12)                     Check if an integer rectangle is valid.
19825: (8)                  """
19826: (8)                  return _mupdf.FzIrect_fz_is_valid_irect(self)
19827: (4)              def fz_rect_from_irect(self):
19828: (8)                  r"""
19829: (8)                  Class-aware wrapper for `::fz_rect_from_irect()`.
19830: (12)                     Convert a bbox into a rect.
19831: (12)                     For our purposes, a rect can represent all the values we meet in
19832: (12)                     a bbox, so nothing can go wrong.
19833: (12)                     rect: A place to store the generated rectangle.
19834: (12)                     bbox: The bbox to convert.
19835: (12)                     Returns rect (updated).
19836: (8)                  """
19837: (8)                  return _mupdf.FzIrect_fz_rect_from_irect(self)
19838: (4)              def fz_translate_irect(self, xoff, yoff):
19839: (8)                  r""" Class-aware wrapper for `::fz_translate_irect()`."""
19840: (8)                  return _mupdf.FzIrect_fz_translate_irect(self, xoff, yoff)
19841: (4)              def __init__(self, *args):
19842: (8)                  r"""
19843: (8)                  *Overload 1:*
19844: (9)                   == Constructors.  Constructor using `fz_irect_from_rect()`.
19845: (16)                         Convert a rect into the minimal bounding box
19846: (16)                         that covers the rectangle.
19847: (16)                         Coordinates in a bounding box are integers, so rounding of the
19848: (16)                         rects coordinates takes place. The top left corner is rounded
19849: (16)                         upwards and left while the bottom right corner is rounded
19850: (16)                         downwards and to the right.
19851: (8)                  |
19852: (8)                  *Overload 2:*
19853: (9)                   Constructor using `fz_make_irect()`.
19854: (8)                  |
19855: (8)                  *Overload 3:*
19856: (9)                   We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
```

```
19857: (8)                  |
19858: (8)                  *Overload 4:*
19859: (9)                   Constructor using raw copy of pre-existing `::fz_irect`.
19860: (8)                  |
19861: (8)                  *Overload 5:*
19862: (9)                   Constructor using raw copy of pre-existing `::fz_irect`.
19863: (8)                  """
19864: (8)                  _mupdf.FzIrect_swiginit(self, _mupdf.new_FzIrect(*args))
19865: (4)              def internal(self, *args):
19866: (8)                  r"""
19867: (8)                  *Overload 1:*
19868: (8)                  Access as underlying struct.
19869: (8)                  |
19870: (8)                  *Overload 2:*
19871: (8)                  Access as underlying struct.
19872: (8)                  """
19873: (8)                  return _mupdf.FzIrect_internal(self, *args)
19874: (4)              __swig_destroy__ = _mupdf.delete_FzIrect
19875: (4)              x0 = property(_mupdf.FzIrect_x0_get, _mupdf.FzIrect_x0_set)
19876: (4)              y0 = property(_mupdf.FzIrect_y0_get, _mupdf.FzIrect_y0_set)
19877: (4)              x1 = property(_mupdf.FzIrect_x1_get, _mupdf.FzIrect_x1_set)
19878: (4)              y1 = property(_mupdf.FzIrect_y1_get, _mupdf.FzIrect_y1_set)
19879: (4)              s_num_instances = property(_mupdf.FzIrect_s_num_instances_get,
_mupdf.FzIrect_s_num_instances_set)
19880: (4)              def to_string(self):
19881: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
19882: (8)                  return _mupdf.FzIrect_to_string(self)
19883: (4)              def __eq__(self, rhs):
19884: (8)                  r""" Comparison method."""
19885: (8)                  return _mupdf.FzIrect___eq__(self, rhs)
19886: (4)              def __ne__(self, rhs):
19887: (8)                  r""" Comparison method."""
19888: (8)                  return _mupdf.FzIrect___ne__(self, rhs)
19889: (0)          _mupdf.FzIrect_swigregister(FzIrect)
19890: (0)          class FzJbig2Globals(object):
19891: (4)              r""" Wrapper class for struct `fz_jbig2_globals`."""
19892: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19893: (4)              __repr__ = _swig_repr
19894: (4)              def fz_jbig2_globals_data(self):
19895: (8)                  r"""
19896: (8)                  Class-aware wrapper for `::fz_jbig2_globals_data()`.
19897: (12)                     Return buffer containing jbig2 globals data stream.
19898: (8)                  """
19899: (8)                  return _mupdf.FzJbig2Globals_fz_jbig2_globals_data(self)
19900: (4)              def __init__(self, *args):
19901: (8)                  r"""
19902: (8)                  *Overload 1:*
19903: (8)                  Copy constructor using `fz_keep_jbig2_globals()`.
19904: (8)                  |
19905: (8)                  *Overload 2:*
19906: (8)                  Default constructor, sets `m_internal` to null.
19907: (8)                  |
19908: (8)                  *Overload 3:*
19909: (8)                  Constructor using raw copy of pre-existing `::fz_jbig2_globals`.
19910: (8)                  """
19911: (8)                  _mupdf.FzJbig2Globals_swiginit(self, _mupdf.new_FzJbig2Globals(*args))
19912: (4)              __swig_destroy__ = _mupdf.delete_FzJbig2Globals
19913: (4)              def m_internal_value(self):
19914: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19915: (8)                  return _mupdf.FzJbig2Globals_m_internal_value(self)
19916: (4)              m_internal = property(_mupdf.FzJbig2Globals_m_internal_get,
_mupdf.FzJbig2Globals_m_internal_set, doc=r""" Pointer to wrapped data.""")
19917: (4)              s_num_instances = property(_mupdf.FzJbig2Globals_s_num_instances_get,
_mupdf.FzJbig2Globals_s_num_instances_set)
19918: (0)          _mupdf.FzJbig2Globals_swigregister(FzJbig2Globals)
19919: (0)          class FzKeyStorable(object):
```

```
19920: (4)              r"""
19921: (4)              Wrapper class for struct `fz_key_storable`.
19922: (4)              Any storable object that can appear in the key of another
19923: (4)              storable object should include an fz_key_storable structure
19924: (4)              at the start (by convention at least) of their structure.
19925: (4)              """
19926: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19927: (4)              __repr__ = _swig_repr
19928: (4)              def __init__(self, *args):
19929: (8)                  r"""
19930: (8)                  *Overload 1:*
19931: (8)                  Copy constructor using `fz_keep_key_storable()`.
19932: (8)                  |
19933: (8)                  *Overload 2:*
19934: (8)                  Default constructor, sets `m_internal` to null.
19935: (8)                  |
19936: (8)                  *Overload 3:*
19937: (8)                  Constructor using raw copy of pre-existing `::fz_key_storable`.
19938: (8)                  """
19939: (8)                  _mupdf.FzKeyStorable_swiginit(self, _mupdf.new_FzKeyStorable(*args))
19940: (4)              __swig_destroy__ = _mupdf.delete_FzKeyStorable
19941: (4)              def m_internal_value(self):
19942: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19943: (8)                  return _mupdf.FzKeyStorable_m_internal_value(self)
19944: (4)              m_internal = property(_mupdf.FzKeyStorable_m_internal_get,
_mupdf.FzKeyStorable_m_internal_set, doc=r""" Pointer to wrapped data.""")
19945: (4)              s_num_instances = property(_mupdf.FzKeyStorable_s_num_instances_get,
_mupdf.FzKeyStorable_s_num_instances_set)
19946: (0)          _mupdf.FzKeyStorable_swigregister(FzKeyStorable)
19947: (0)          class FzLayoutBlock(object):
19948: (4)              r""" Wrapper class for struct `fz_layout_block`. Not copyable or
assignable."""
19949: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19950: (4)              __repr__ = _swig_repr
19951: (4)              def fz_add_layout_char(self, x, w, p):
19952: (8)                  r"""
19953: (8)                  Class-aware wrapper for `::fz_add_layout_char()`.
19954: (12)                     Add a new char to the line at the end of the layout block.
19955: (8)                  """
19956: (8)                  return _mupdf.FzLayoutBlock_fz_add_layout_char(self, x, w, p)
19957: (4)              def fz_add_layout_line(self, x, y, h, p):
19958: (8)                  r"""
19959: (8)                  Class-aware wrapper for `::fz_add_layout_line()`.
19960: (12)                     Add a new line to the end of the layout block.
19961: (8)                  """
19962: (8)                  return _mupdf.FzLayoutBlock_fz_add_layout_line(self, x, y, h, p)
19963: (4)              def __init__(self, *args):
19964: (8)                  r"""
19965: (8)                  *Overload 1:*
19966: (9)                   == Constructors.  Constructor using `fz_new_layout()`.
19967: (16)                         Create a new layout block, with new allocation pool, zero
19968: (16)                         matrices, and initialise linked pointers.
19969: (8)                  |
19970: (8)                  *Overload 2:*
19971: (9)                   Constructor using raw copy of pre-existing `::fz_layout_block`.
19972: (8)                  """
19973: (8)                  _mupdf.FzLayoutBlock_swiginit(self, _mupdf.new_FzLayoutBlock(*args))
19974: (4)              __swig_destroy__ = _mupdf.delete_FzLayoutBlock
19975: (4)              def m_internal_value(self):
19976: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
19977: (8)                  return _mupdf.FzLayoutBlock_m_internal_value(self)
19978: (4)              m_internal = property(_mupdf.FzLayoutBlock_m_internal_get,
_mupdf.FzLayoutBlock_m_internal_set, doc=r""" Pointer to wrapped data.""")
19979: (4)              s_num_instances = property(_mupdf.FzLayoutBlock_s_num_instances_get,
_mupdf.FzLayoutBlock_s_num_instances_set)
```

```
19980: (0)          _mupdf.FzLayoutBlock_swigregister(FzLayoutBlock)
19981: (0)          class FzLayoutChar(object):
19982: (4)              r"""
19983: (4)              Wrapper class for struct `fz_layout_char`. Not copyable or assignable.
19984: (4)              Simple text layout (for use with annotation editing primarily).
19985: (4)              """
19986: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
19987: (4)              __repr__ = _swig_repr
19988: (4)              def __init__(self, *args):
19989: (8)                  r"""
19990: (8)                  *Overload 1:*
19991: (8)                  Default constructor, sets `m_internal` to null.
19992: (8)                  |
19993: (8)                  *Overload 2:*
19994: (8)                  Constructor using raw copy of pre-existing `::fz_layout_char`.
19995: (8)                  """
19996: (8)                  _mupdf.FzLayoutChar_swiginit(self, _mupdf.new_FzLayoutChar(*args))
19997: (4)              __swig_destroy__ = _mupdf.delete_FzLayoutChar
19998: (4)              def m_internal_value(self):
19999: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20000: (8)                  return _mupdf.FzLayoutChar_m_internal_value(self)
20001: (4)              m_internal = property(_mupdf.FzLayoutChar_m_internal_get,
_mupdf.FzLayoutChar_m_internal_set, doc=r""" Pointer to wrapped data.""")
20002: (4)              s_num_instances = property(_mupdf.FzLayoutChar_s_num_instances_get,
_mupdf.FzLayoutChar_s_num_instances_set)
20003: (0)          _mupdf.FzLayoutChar_swigregister(FzLayoutChar)
20004: (0)          class FzLayoutLine(object):
20005: (4)              r""" Wrapper class for struct `fz_layout_line`. Not copyable or
assignable."""
20006: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20007: (4)              __repr__ = _swig_repr
20008: (4)              def __init__(self, *args):
20009: (8)                  r"""
20010: (8)                  *Overload 1:*
20011: (8)                  Default constructor, sets `m_internal` to null.
20012: (8)                  |
20013: (8)                  *Overload 2:*
20014: (8)                  Constructor using raw copy of pre-existing `::fz_layout_line`.
20015: (8)                  """
20016: (8)                  _mupdf.FzLayoutLine_swiginit(self, _mupdf.new_FzLayoutLine(*args))
20017: (4)              __swig_destroy__ = _mupdf.delete_FzLayoutLine
20018: (4)              def m_internal_value(self):
20019: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20020: (8)                  return _mupdf.FzLayoutLine_m_internal_value(self)
20021: (4)              m_internal = property(_mupdf.FzLayoutLine_m_internal_get,
_mupdf.FzLayoutLine_m_internal_set, doc=r""" Pointer to wrapped data.""")
20022: (4)              s_num_instances = property(_mupdf.FzLayoutLine_s_num_instances_get,
_mupdf.FzLayoutLine_s_num_instances_set)
20023: (0)          _mupdf.FzLayoutLine_swigregister(FzLayoutLine)
20024: (0)          class FzLink(object):
20025: (4)              r"""
20026: (4)              Wrapper class for struct `fz_link`.
20027: (4)              fz_link is a list of interactive links on a page.
20028: (4)              There is no relation between the order of the links in the
20029: (4)              list and the order they appear on the page. The list of links
20030: (4)              for a given page can be obtained from fz_load_links.
20031: (4)              A link is reference counted. Dropping a reference to a link is
20032: (4)              done by calling fz_drop_link.
20033: (4)              rect: The hot zone. The area that can be clicked in
20034: (4)              untransformed coordinates.
20035: (4)              uri: Link destinations come in two forms: internal and external.
20036: (4)              Internal links refer to other pages in the same document.
20037: (4)              External links are URLs to other documents.
20038: (4)              next: A pointer to the next link on the same page.
20039: (4)              """
```

```
20040: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20041: (4)              __repr__ = _swig_repr
20042: (4)              def fz_set_link_rect(self, rect):
20043: (8)                  r""" Class-aware wrapper for `::fz_set_link_rect()`."""
20044: (8)                  return _mupdf.FzLink_fz_set_link_rect(self, rect)
20045: (4)              def fz_set_link_uri(self, uri):
20046: (8)                  r""" Class-aware wrapper for `::fz_set_link_uri()`."""
20047: (8)                  return _mupdf.FzLink_fz_set_link_uri(self, uri)
20048: (4)              def begin(self):
20049: (8)                  r""" Used for iteration over linked list of FzLink items starting at
fz_link::."""
20050: (8)                  return _mupdf.FzLink_begin(self)
20051: (4)              def end(self):
20052: (8)                  r""" Used for iteration over linked list of FzLink items starting at
fz_link::."""
20053: (8)                  return _mupdf.FzLink_end(self)
20054: (4)              def __init__(self, *args):
20055: (8)                  r"""
20056: (8)                  *Overload 1:*
20057: (9)                   == Constructors.  Constructor using `fz_new_link_of_size()`.
20058: (16)                         Create a new link record.
20059: (16)                         next is set to NULL with the expectation that the caller will
20060: (16)                         handle the linked list setup. Internal function.
20061: (16)                         Different document types will be implemented by deriving from
20062: (16)                         fz_link. This macro allocates such derived structures, and
20063: (16)                         initialises the base sections.
20064: (8)                  |
20065: (8)                  *Overload 2:*
20066: (9)                   Constructor using `pdf_new_link()`.
20067: (8)                  |
20068: (8)                  *Overload 3:*
20069: (9)                   Construct by calling fz_new_link_of_size() with size=sizeof(fz_link).
20070: (8)                  |
20071: (8)                  *Overload 4:*
20072: (9)                   Copy constructor using `fz_keep_link()`.
20073: (8)                  |
20074: (8)                  *Overload 5:*
20075: (9)                   Constructor using raw copy of pre-existing `::fz_link`.
20076: (8)                  |
20077: (8)                  *Overload 6:*
20078: (9)                   Constructor using raw copy of pre-existing `::fz_link`.
20079: (8)                  """
20080: (8)                  _mupdf.FzLink_swiginit(self, _mupdf.new_FzLink(*args))
20081: (4)              def refs(self):
20082: (8)                  return _mupdf.FzLink_refs(self)
20083: (4)              def next(self):
20084: (8)                  return _mupdf.FzLink_next(self)
20085: (4)              def rect(self):
20086: (8)                  return _mupdf.FzLink_rect(self)
20087: (4)              def uri(self):
20088: (8)                  return _mupdf.FzLink_uri(self)
20089: (4)              __swig_destroy__ = _mupdf.delete_FzLink
20090: (4)              def m_internal_value(self):
20091: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20092: (8)                  return _mupdf.FzLink_m_internal_value(self)
20093: (4)              m_internal = property(_mupdf.FzLink_m_internal_get,
_mupdf.FzLink_m_internal_set, doc=r""" Pointer to wrapped data.""")
20094: (4)              s_num_instances = property(_mupdf.FzLink_s_num_instances_get,
_mupdf.FzLink_s_num_instances_set)
20095: (0)          _mupdf.FzLink_swigregister(FzLink)
20096: (0)          class FzLinkDest(object):
20097: (4)              r""" Wrapper class for struct `fz_link_dest`. Not copyable or
assignable."""
20098: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20099: (4)              __repr__ = _swig_repr
20100: (4)              def pdf_new_uri_from_explicit_dest(self):
```

```
20101: (8)                  r""" Class-aware wrapper for `::pdf_new_uri_from_explicit_dest()`."""
20102: (8)                  return _mupdf.FzLinkDest_pdf_new_uri_from_explicit_dest(self)
20103: (4)              def __init__(self, *args):
20104: (8)                  r"""
20105: (8)                  *Overload 1:*
20106: (8)                  Default constructor, sets `m_internal` to null.
20107: (8)                  |
20108: (8)                  *Overload 2:*
20109: (8)                  Constructor using raw copy of pre-existing `::fz_link_dest`.
20110: (8)                  """
20111: (8)                  _mupdf.FzLinkDest_swiginit(self, _mupdf.new_FzLinkDest(*args))
20112: (4)              __swig_destroy__ = _mupdf.delete_FzLinkDest
20113: (4)              def m_internal_value(self):
20114: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20115: (8)                  return _mupdf.FzLinkDest_m_internal_value(self)
20116: (4)              m_internal = property(_mupdf.FzLinkDest_m_internal_get,
_mupdf.FzLinkDest_m_internal_set, doc=r""" Pointer to wrapped data.""")
20117: (4)              s_num_instances = property(_mupdf.FzLinkDest_s_num_instances_get,
_mupdf.FzLinkDest_s_num_instances_set)
20118: (0)          _mupdf.FzLinkDest_swigregister(FzLinkDest)
20119: (0)          class FzLocation(object):
20120: (4)              r"""
20121: (4)              Wrapper class for struct `fz_location`.
20122: (4)              Locations within the document are referred to in terms of
20123: (4)              chapter and page, rather than just a page number. For some
20124: (4)              documents (such as epub documents with large numbers of pages
20125: (4)              broken into many chapters) this can make navigation much faster
20126: (4)              as only the required chapter needs to be decoded at a time.
20127: (4)              """
20128: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20129: (4)              __repr__ = _swig_repr
20130: (4)              def __init__(self, *args):
20131: (8)                  r"""
20132: (8)                  *Overload 1:*
20133: (9)                   == Constructors.  Constructor using `fz_make_location()`.
20134: (16)                         Simple constructor for fz_locations.
20135: (8)                  |
20136: (8)                  *Overload 2:*
20137: (9)                   We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
20138: (8)                  |
20139: (8)                  *Overload 3:*
20140: (9)                   Constructor using raw copy of pre-existing `::fz_location`.
20141: (8)                  |
20142: (8)                  *Overload 4:*
20143: (9)                   Constructor using raw copy of pre-existing `::fz_location`.
20144: (8)                  """
20145: (8)                  _mupdf.FzLocation_swiginit(self, _mupdf.new_FzLocation(*args))
20146: (4)              def internal(self, *args):
20147: (8)                  r"""
20148: (8)                  *Overload 1:*
20149: (8)                  Access as underlying struct.
20150: (8)                  |
20151: (8)                  *Overload 2:*
20152: (8)                  Access as underlying struct.
20153: (8)                  """
20154: (8)                  return _mupdf.FzLocation_internal(self, *args)
20155: (4)              __swig_destroy__ = _mupdf.delete_FzLocation
20156: (4)              chapter = property(_mupdf.FzLocation_chapter_get,
_mupdf.FzLocation_chapter_set)
20157: (4)              page = property(_mupdf.FzLocation_page_get, _mupdf.FzLocation_page_set)
20158: (4)              s_num_instances = property(_mupdf.FzLocation_s_num_instances_get,
_mupdf.FzLocation_s_num_instances_set)
20159: (4)              def to_string(self):
20160: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
20161: (8)                  return _mupdf.FzLocation_to_string(self)
```

```
20162: (4)              def __eq__(self, rhs):
20163: (8)                  r""" Comparison method."""
20164: (8)                  return _mupdf.FzLocation___eq__(self, rhs)
20165: (4)              def __ne__(self, rhs):
20166: (8)                  r""" Comparison method."""
20167: (8)                  return _mupdf.FzLocation___ne__(self, rhs)
20168: (0)          _mupdf.FzLocation_swigregister(FzLocation)
20169: (0)          class FzLocksContext(object):
20170: (4)              r"""
20171: (4)              Wrapper class for struct `fz_locks_context`. Not copyable or assignable.
20172: (4)              Locking functions
20173: (4)              MuPDF is kept deliberately free of any knowledge of particular
20174: (4)              threading systems. As such, in order for safe multi-threaded
20175: (4)              operation, we rely on callbacks to client provided functions.
20176: (4)              A client is expected to provide FZ_LOCK_MAX number of mutexes,
20177: (4)              and a function to lock/unlock each of them. These may be
20178: (4)              recursive mutexes, but do not have to be.
20179: (4)              If a client does not intend to use multiple threads, then it
20180: (4)              may pass NULL instead of a lock structure.
20181: (4)              In order to avoid deadlocks, we have one simple rule
20182: (4)              internally as to how we use locks: We can never take lock n
20183: (4)              when we already hold any lock i, where 0 <= i <= n. In order
20184: (4)              to verify this, we have some debugging code, that can be
20185: (4)              enabled by defining FITZ_DEBUG_LOCKING.
20186: (4)              """
20187: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20188: (4)              __repr__ = _swig_repr
20189: (4)              def __init__(self, *args):
20190: (8)                  r"""
20191: (8)                  *Overload 1:*
20192: (8)                  Default constructor, sets `m_internal` to null.
20193: (8)                  |
20194: (8)                  *Overload 2:*
20195: (8)                  Constructor using raw copy of pre-existing `::fz_locks_context`.
20196: (8)                  """
20197: (8)                  _mupdf.FzLocksContext_swiginit(self, _mupdf.new_FzLocksContext(*args))
20198: (4)              __swig_destroy__ = _mupdf.delete_FzLocksContext
20199: (4)              def m_internal_value(self):
20200: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20201: (8)                  return _mupdf.FzLocksContext_m_internal_value(self)
20202: (4)              m_internal = property(_mupdf.FzLocksContext_m_internal_get,
_mupdf.FzLocksContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
20203: (4)              s_num_instances = property(_mupdf.FzLocksContext_s_num_instances_get,
_mupdf.FzLocksContext_s_num_instances_set)
20204: (0)          _mupdf.FzLocksContext_swigregister(FzLocksContext)
20205: (0)          class FzMatrix(object):
20206: (4)              r"""
20207: (5)               Wrapper class for struct `fz_matrix`.
20208: (8)                  fz_matrix is a row-major 3x3 matrix used for representing
20209: (8)                  transformations of coordinates throughout MuPDF.
20210: (8)                  Since all points reside in a two-dimensional space, one vector
20211: (8)                  is always a constant unit vector; hence only some elements may
20212: (8)                  vary in a matrix. Below is how the elements map between
20213: (8)                  different representations.
20214: (4)              a b 0
20215: (8)                  | c d 0 | normally represented as [ a b c d e f ].
20216: (8)                  \ e f 1 /
20217: (4)              """
20218: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20219: (4)              __repr__ = _swig_repr
20220: (4)              @staticmethod
20221: (4)              def fz_scale(sx, sy):
20222: (8)                  r"""
20223: (8)                  Class-aware wrapper for `::fz_scale()`.
20224: (12)                     Create a scaling matrix.
20225: (12)                     The returned matrix is of the form [ sx 0 0 sy 0 0 ].
```

```
20226: (12)                     m: Pointer to the matrix to populate
20227: (12)                     sx, sy: Scaling factors along the X- and Y-axes. A scaling
20228: (12)                     factor of 1.0 will not cause any scaling along the relevant
20229: (12)                     axis.
20230: (12)                     Returns m.
20231: (8)                  """
20232: (8)                  return _mupdf.FzMatrix_fz_scale(sx, sy)
20233: (4)              @staticmethod
20234: (4)              def fz_shear(sx, sy):
20235: (8)                  r"""
20236: (8)                  Class-aware wrapper for `::fz_shear()`.
20237: (12)                     Create a shearing matrix.
20238: (12)                     The returned matrix is of the form [ 1 sy sx 1 0 0 ].
20239: (12)                     m: pointer to place to store returned matrix
20240: (12)                     sx, sy: Shearing factors. A shearing factor of 0.0 will not
20241: (12)                     cause any shearing along the relevant axis.
20242: (12)                     Returns m.
20243: (8)                  """
20244: (8)                  return _mupdf.FzMatrix_fz_shear(sx, sy)
20245: (4)              @staticmethod
20246: (4)              def fz_rotate(degrees):
20247: (8)                  r"""
20248: (8)                  Class-aware wrapper for `::fz_rotate()`.
20249: (12)                     Create a rotation matrix.
20250: (12)                     The returned matrix is of the form
20251: (12)                     [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
20252: (12)                     m: Pointer to place to store matrix
20253: (12)                     degrees: Degrees of counter clockwise rotation. Values less
20254: (12)                     than zero and greater than 360 are handled as expected.
20255: (12)                     Returns m.
20256: (8)                  """
20257: (8)                  return _mupdf.FzMatrix_fz_rotate(degrees)
20258: (4)              @staticmethod
20259: (4)              def fz_translate(tx, ty):
20260: (8)                  r"""
20261: (8)                  Class-aware wrapper for `::fz_translate()`.
20262: (12)                     Create a translation matrix.
20263: (12)                     The returned matrix is of the form [ 1 0 0 1 tx ty ].
20264: (12)                     m: A place to store the created matrix.
20265: (12)                     tx, ty: Translation distances along the X- and Y-axes. A
20266: (12)                     translation of 0 will not cause any translation along the
20267: (12)                     relevant axis.
20268: (12)                     Returns m.
20269: (8)                  """
20270: (8)                  return _mupdf.FzMatrix_fz_translate(tx, ty)
20271: (4)              @staticmethod
20272: (4)              def fz_transform_page(mediabox, resolution, rotate):
20273: (8)                  r"""
20274: (8)                  Class-aware wrapper for `::fz_transform_page()`.
20275: (12)                     Create transform matrix to draw page
20276: (12)                     at a given resolution and rotation. Adjusts the scaling
20277: (12)                     factors so that the page covers whole number of
20278: (12)                     pixels and adjust the page origin to be at 0,0.
20279: (8)                  """
20280: (8)                  return _mupdf.FzMatrix_fz_transform_page(mediabox, resolution, rotate)
20281: (4)              def fz_concat(self, *args):
20282: (8)                  r"""
20283: (8)                  *Overload 1:*
20284: (9)                   We use default copy constructor and operator=.  Class-aware wrapper
for `::fz_concat()`.
20285: (16)                         Multiply two matrices.
20286: (16)                         The order of the two matrices are important since matrix
20287: (16)                         multiplication is not commutative.
20288: (16)                         Returns result.
20289: (8)                  |
20290: (8)                  *Overload 2:*
20291: (9)                   Class-aware wrapper for `::fz_concat()`.
20292: (16)                         Multiply two matrices.
20293: (16)                         The order of the two matrices are important since matrix
```

```
(16)                        multiplication is not commutative.
(16)                        Returns result.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_concat(self, *args)
(4)             def fz_invert_matrix(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_invert_matrix()`.
(12)                    Create an inverse matrix.
(12)                    inverse: Place to store inverse matrix.
(12)                    matrix: Matrix to invert. A degenerate matrix, where the
(12)                    determinant is equal to zero, can not be inverted and the
(12)                    original matrix is returned instead.
(12)                    Returns inverse.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_invert_matrix(self)
(4)             def fz_is_identity(self):
(8)                 r""" Class-aware wrapper for `::fz_is_identity()`."""
(8)                 return _mupdf.FzMatrix_fz_is_identity(self)
(4)             def fz_is_rectilinear(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_rectilinear()`.
(12)                    Check if a transformation is rectilinear.
(12)                    Rectilinear means that no shearing is present and that any
(12)                    rotations present are a multiple of 90 degrees. Usually this
(12)                    is used to make sure that axis-aligned rectangles before the
(12)                    transformation are still axis-aligned rectangles afterwards.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_is_rectilinear(self)
(4)             def fz_matrix_expansion(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_matrix_expansion()`.
(12)                    Calculate average scaling factor of matrix.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_matrix_expansion(self)
(4)             def fz_matrix_max_expansion(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_matrix_max_expansion()`.
(12)                    Find the largest expansion performed by this matrix.
(12)                    (i.e. max(abs(m.a),abs(m.b),abs(m.c),abs(m.d))
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_matrix_max_expansion(self)
(4)             def fz_post_scale(self, sx, sy):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_post_scale()`.
(12)                    Scale a matrix by postmultiplication.
(12)                    m: Pointer to the matrix to scale
(12)                    sx, sy: Scaling factors along the X- and Y-axes. A scaling
(12)                    factor of 1.0 will not cause any scaling along the relevant
(12)                    axis.
(12)                    Returns m (updated).
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_post_scale(self, sx, sy)
(4)             def fz_pre_rotate(self, degrees):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pre_rotate()`.
(12)                    Rotate a transformation by premultiplying.
(12)                    The premultiplied matrix is of the form
(12)                    [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
(12)                    m: Pointer to matrix to premultiply.
(12)                    degrees: Degrees of counter clockwise rotation. Values less
(12)                    than zero and greater than 360 are handled as expected.
(12)                    Returns m (updated).
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_pre_rotate(self, degrees)
(4)             def fz_pre_scale(self, sx, sy):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pre_scale()`.
(12)                    Scale a matrix by premultiplication.
(12)                    m: Pointer to the matrix to scale
```

```
(12)                    sx, sy: Scaling factors along the X- and Y-axes. A scaling
(12)                    factor of 1.0 will not cause any scaling along the relevant
(12)                    axis.
(12)                    Returns m (updated).
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_pre_scale(self, sx, sy)
(4)             def fz_pre_shear(self, sx, sy):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pre_shear()`.
(12)                    Premultiply a matrix with a shearing matrix.
(12)                    The shearing matrix is of the form [ 1 sy sx 1 0 0 ].
(12)                    m: pointer to matrix to premultiply
(12)                    sx, sy: Shearing factors. A shearing factor of 0.0 will not
(12)                    cause any shearing along the relevant axis.
(12)                    Returns m (updated).
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_pre_shear(self, sx, sy)
(4)             def fz_pre_translate(self, tx, ty):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pre_translate()`.
(12)                    Translate a matrix by premultiplication.
(12)                    m: The matrix to translate
(12)                    tx, ty: Translation distances along the X- and Y-axes. A
(12)                    translation of 0 will not cause any translation along the
(12)                    relevant axis.
(12)                    Returns m.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_pre_translate(self, tx, ty)
(4)             def fz_subpixel_adjust(self, subpix_ctm, qe, qf):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_subpixel_adjust()`.
(12)                    Perform subpixel quantisation and adjustment on a glyph matrix.
(12)                    ctm: On entry, the desired 'ideal' transformation for a glyph.
(12)                    On exit, adjusted to a (very similar) transformation quantised
(12)                    for subpixel caching.
(12)                    subpix_ctm: Initialised by the routine to the transform that
(12)                    should be used to render the glyph.
(12)                    qe, qf: which subpixel position we quantised to.
(12)                    Returns: the size of the glyph.
(12)                    Note: This is currently only exposed for use in our app. It
(12)                    should be considered "at risk" of removal from the API.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_subpixel_adjust(self, subpix_ctm, qe, qf)
(4)             def fz_try_invert_matrix(self, src):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_try_invert_matrix()`.
(12)                    Attempt to create an inverse matrix.
(12)                    inverse: Place to store inverse matrix.
(12)                    matrix: Matrix to invert. A degenerate matrix, where the
(12)                    determinant is equal to zero, can not be inverted.
(12)                    Returns 1 if matrix is degenerate (singular), or 0 otherwise.
(8)                 """
(8)                 return _mupdf.FzMatrix_fz_try_invert_matrix(self, src)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(8)                 == Constructors.  Constructor using `fz_make_matrix()`.
(8)                 |
(8)                 *Overload 2:*
(8)                 Constructs identity matrix (like fz_identity).
(8)                 |
(8)                 *Overload 3:*
(8)                 Constructor using raw copy of pre-existing `::fz_matrix`.
(8)                 |
(8)                 *Overload 4:*
(8)                 Constructor using raw copy of pre-existing `::fz_matrix`.
(8)                 """
(8)                 _mupdf.FzMatrix_swiginit(self, _mupdf.new_FzMatrix(*args))
(4)             def internal(self, *args):
```

```
20432: (8)                  r"""
20433: (8)                  *Overload 1:*
20434: (8)                  Access as underlying struct.
20435: (8)                  |
20436: (8)                  *Overload 2:*
20437: (8)                  Access as underlying struct.
20438: (8)                  """
20439: (8)                  return _mupdf.FzMatrix_internal(self, *args)
20440: (4)              __swig_destroy__ = _mupdf.delete_FzMatrix
20441: (4)              a = property(_mupdf.FzMatrix_a_get, _mupdf.FzMatrix_a_set)
20442: (4)              b = property(_mupdf.FzMatrix_b_get, _mupdf.FzMatrix_b_set)
20443: (4)              c = property(_mupdf.FzMatrix_c_get, _mupdf.FzMatrix_c_set)
20444: (4)              d = property(_mupdf.FzMatrix_d_get, _mupdf.FzMatrix_d_set)
20445: (4)              e = property(_mupdf.FzMatrix_e_get, _mupdf.FzMatrix_e_set)
20446: (4)              f = property(_mupdf.FzMatrix_f_get, _mupdf.FzMatrix_f_set)
20447: (4)              s_num_instances = property(_mupdf.FzMatrix_s_num_instances_get,
_mupdf.FzMatrix_s_num_instances_set)
20448: (4)              def to_string(self):
20449: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
20450: (8)                  return _mupdf.FzMatrix_to_string(self)
20451: (4)              def __eq__(self, rhs):
20452: (8)                  r""" Comparison method."""
20453: (8)                  return _mupdf.FzMatrix___eq__(self, rhs)
20454: (4)              def __ne__(self, rhs):
20455: (8)                  r""" Comparison method."""
20456: (8)                  return _mupdf.FzMatrix___ne__(self, rhs)
20457: (0)          _mupdf.FzMatrix_swigregister(FzMatrix)
20458: (0)          class FzMd5(object):
20459: (4)              r"""
20460: (4)              Wrapper class for struct `fz_md5`.
20461: (4)              Structure definition is public to enable stack
20462: (4)              based allocation. Do not access the members directly.
20463: (4)              """
20464: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20465: (4)              __repr__ = _swig_repr
20466: (4)              def fz_md5_final(self, digest):
20467: (8)                  r"""
20468: (8)                  We use default copy constructor and operator=.  Class-aware wrapper
for `::fz_md5_final()`.
20469: (12)                     MD5 finalization. Ends an MD5 message-digest operation, writing
20470: (12)                     the message digest and zeroizing the context.
20471: (12)                     Never throws an exception.
20472: (8)                  """
20473: (8)                  return _mupdf.FzMd5_fz_md5_final(self, digest)
20474: (4)              def fz_md5_final2(self):
20475: (8)                  r"""
20476: (8)                  Class-aware wrapper for `::fz_md5_final2()`.
20477: (8)                  C++ alternative to fz_md5_final() that returns the digest by value.
20478: (8)                  """
20479: (8)                  return _mupdf.FzMd5_fz_md5_final2(self)
20480: (4)              def fz_md5_init(self):
20481: (8)                  r"""
20482: (8)                  Class-aware wrapper for `::fz_md5_init()`.
20483: (12)                     MD5 initialization. Begins an MD5 operation, writing a new
20484: (12)                     context.
20485: (12)                     Never throws an exception.
20486: (8)                  """
20487: (8)                  return _mupdf.FzMd5_fz_md5_init(self)
20488: (4)              def fz_md5_update(self, input, inlen):
20489: (8)                  r"""
20490: (8)                  Class-aware wrapper for `::fz_md5_update()`.
20491: (12)                     MD5 block update operation. Continues an MD5 message-digest
20492: (12)                     operation, processing another message block, and updating the
20493: (12)                     context.
20494: (12)                     Never throws an exception.
20495: (8)                  """
20496: (8)                  return _mupdf.FzMd5_fz_md5_update(self, input, inlen)
```

```
20497: (4)              def fz_md5_update_int64(self, i):
20498: (8)                  r"""
20499: (8)                  Class-aware wrapper for `::fz_md5_update_int64()`.
20500: (12)                     MD5 block update operation. Continues an MD5 message-digest
20501: (12)                     operation, processing an int64, and updating the context.
20502: (12)                     Never throws an exception.
20503: (8)                  """
20504: (8)                  return _mupdf.FzMd5_fz_md5_update_int64(self, i)
20505: (4)              def __init__(self, *args):
20506: (8)                  r"""
20507: (8)                  *Overload 1:*
20508: (8)                  Default constructor calls md5_init().
20509: (8)                  |
20510: (8)                  *Overload 2:*
20511: (8)                  Constructor using raw copy of pre-existing `::fz_md5`.
20512: (8)                  """
20513: (8)                  _mupdf.FzMd5_swiginit(self, _mupdf.new_FzMd5(*args))
20514: (4)              def lo(self):
20515: (8)                  return _mupdf.FzMd5_lo(self)
20516: (4)              def hi(self):
20517: (8)                  return _mupdf.FzMd5_hi(self)
20518: (4)              def a(self):
20519: (8)                  return _mupdf.FzMd5_a(self)
20520: (4)              def b(self):
20521: (8)                  return _mupdf.FzMd5_b(self)
20522: (4)              def c(self):
20523: (8)                  return _mupdf.FzMd5_c(self)
20524: (4)              def d(self):
20525: (8)                  return _mupdf.FzMd5_d(self)
20526: (4)              def buffer(self):
20527: (8)                  return _mupdf.FzMd5_buffer(self)
20528: (4)              __swig_destroy__ = _mupdf.delete_FzMd5
20529: (4)              m_internal = property(_mupdf.FzMd5_m_internal_get,
_mupdf.FzMd5_m_internal_set)
20530: (4)              s_num_instances = property(_mupdf.FzMd5_s_num_instances_get,
_mupdf.FzMd5_s_num_instances_set, doc=r""" Wrapped data is held by value.""")
20531: (4)              def to_string(self):
20532: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
20533: (8)                  return _mupdf.FzMd5_to_string(self)
20534: (4)              def __eq__(self, rhs):
20535: (8)                  r""" Comparison method."""
20536: (8)                  return _mupdf.FzMd5___eq__(self, rhs)
20537: (4)              def __ne__(self, rhs):
20538: (8)                  r""" Comparison method."""
20539: (8)                  return _mupdf.FzMd5___ne__(self, rhs)
20540: (0)          _mupdf.FzMd5_swigregister(FzMd5)
20541: (0)          class FzOutline(object):
20542: (4)              r"""
20543: (4)              Wrapper class for struct `fz_outline`.
20544: (4)              fz_outline is a tree of the outline of a document (also known
20545: (4)              as table of contents).
20546: (4)              title: Title of outline item using UTF-8 encoding. May be NULL
20547: (4)              if the outline item has no text string.
20548: (4)              uri: Destination in the document to be displayed when this
20549: (4)              outline item is activated. May be an internal or external
20550: (4)              link, or NULL if the outline item does not have a destination.
20551: (4)              page: The page number of an internal link, or -1 for external
20552: (4)              links or links with no destination.
20553: (4)              next: The next outline item at the same level as this outline
20554: (4)              item. May be NULL if no more outline items exist at this level.
20555: (4)              down: The outline items immediate children in the hierarchy.
20556: (4)              May be NULL if no children exist.
20557: (4)              """
20558: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20559: (4)              __repr__ = _swig_repr
20560: (4)              def __init__(self, *args):
20561: (8)                  r"""
```

```
20562: (8)                  *Overload 1:*
20563: (9)                   == Constructors.  Constructor using `fz_load_outline()`.
20564: (16)                         Load the hierarchical document outline.
20565: (16)                         Should be freed by fz_drop_outline.
20566: (8)                  |
20567: (8)                  *Overload 2:*
20568: (9)                   Constructor using `fz_load_outline_from_iterator()`.
20569: (16)                         Routine to implement the old Structure based API from an
iterator.
20570: (8)                  |
20571: (8)                  *Overload 3:*
20572: (9)                   Constructor using `fz_new_outline()`.
20573: (16)                         Create a new outline entry with zeroed fields for the caller
20574: (16)                         to fill in.
20575: (8)                  |
20576: (8)                  *Overload 4:*
20577: (9)                   Copy constructor using `fz_keep_outline()`.
20578: (8)                  |
20579: (8)                  *Overload 5:*
20580: (9)                   Constructor using raw copy of pre-existing `::fz_outline`.
20581: (8)                  """
20582: (8)                  _mupdf.FzOutline_swiginit(self, _mupdf.new_FzOutline(*args))
20583: (4)              def refs(self):
20584: (8)                  return _mupdf.FzOutline_refs(self)
20585: (4)              def title(self):
20586: (8)                  return _mupdf.FzOutline_title(self)
20587: (4)              def uri(self):
20588: (8)                  return _mupdf.FzOutline_uri(self)
20589: (4)              def page(self):
20590: (8)                  return _mupdf.FzOutline_page(self)
20591: (4)              def x(self):
20592: (8)                  return _mupdf.FzOutline_x(self)
20593: (4)              def y(self):
20594: (8)                  return _mupdf.FzOutline_y(self)
20595: (4)              def next(self):
20596: (8)                  return _mupdf.FzOutline_next(self)
20597: (4)              def down(self):
20598: (8)                  return _mupdf.FzOutline_down(self)
20599: (4)              def is_open(self):
20600: (8)                  return _mupdf.FzOutline_is_open(self)
20601: (4)              __swig_destroy__ = _mupdf.delete_FzOutline
20602: (4)              def m_internal_value(self):
20603: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20604: (8)                  return _mupdf.FzOutline_m_internal_value(self)
20605: (4)              m_internal = property(_mupdf.FzOutline_m_internal_get,
_mupdf.FzOutline_m_internal_set, doc=r""" Pointer to wrapped data.""")
20606: (4)              s_num_instances = property(_mupdf.FzOutline_s_num_instances_get,
_mupdf.FzOutline_s_num_instances_set)
20607: (0)          _mupdf.FzOutline_swigregister(FzOutline)
20608: (0)          class FzOutlineItem(object):
20609: (4)              r""" Wrapper class for struct `fz_outline_item`."""
20610: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20611: (4)              __repr__ = _swig_repr
20612: (4)              def valid(self):
20613: (8)                  return _mupdf.FzOutlineItem_valid(self)
20614: (4)              def title(self):
20615: (8)                  return _mupdf.FzOutlineItem_title(self)
20616: (4)              def uri(self):
20617: (8)                  return _mupdf.FzOutlineItem_uri(self)
20618: (4)              def is_open(self):
20619: (8)                  return _mupdf.FzOutlineItem_is_open(self)
20620: (4)              def __init__(self, *args):
20621: (8)                  r"""
20622: (8)                  *Overload 1:*
20623: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
20624: (8)                  |
```

```
20625: (8)                  *Overload 2:*
20626: (8)                  Constructor using raw copy of pre-existing `::fz_outline_item`.
20627: (8)                  """
20628: (8)                  _mupdf.FzOutlineItem_swiginit(self, _mupdf.new_FzOutlineItem(*args))
20629: (4)              __swig_destroy__ = _mupdf.delete_FzOutlineItem
20630: (4)              s_num_instances = property(_mupdf.FzOutlineItem_s_num_instances_get,
_mupdf.FzOutlineItem_s_num_instances_set)
20631: (0)          _mupdf.FzOutlineItem_swigregister(FzOutlineItem)
20632: (0)          class FzOutlineIterator(object):
20633: (4)              r""" Wrapper class for struct `fz_outline_iterator`. Not copyable or
assignable."""
20634: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20635: (4)              __repr__ = _swig_repr
20636: (4)              def fz_load_outline_from_iterator(self):
20637: (8)                  r"""
20638: (8)                  Class-aware wrapper for `::fz_load_outline_from_iterator()`.
20639: (12)                     Routine to implement the old Structure based API from an iterator.
20640: (8)                  """
20641: (8)                  return _mupdf.FzOutlineIterator_fz_load_outline_from_iterator(self)
20642: (4)              def fz_outline_iterator_delete(self):
20643: (8)                  r"""
20644: (8)                  Class-aware wrapper for `::fz_outline_iterator_delete()`.
20645: (12)                     Delete the current item.
20646: (12)                     This implicitly moves us to the 'next' item, and the return code
is as for fz_outline_iterator_next.
20647: (8)                  """
20648: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_delete(self)
20649: (4)              def fz_outline_iterator_down(self):
20650: (8)                  r""" Class-aware wrapper for `::fz_outline_iterator_down()`."""
20651: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_down(self)
20652: (4)              def fz_outline_iterator_item(self):
20653: (8)                  r"""
20654: (8)                  Class-aware wrapper for `::fz_outline_iterator_item()`.
20655: (12)                     Call to get the current outline item.
20656: (12)                     Can return NULL. The item is only valid until the next call.
20657: (8)                  """
20658: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_item(self)
20659: (4)              def fz_outline_iterator_next(self):
20660: (8)                  r"""
20661: (8)                  Class-aware wrapper for `::fz_outline_iterator_next()`.
20662: (12)                     Calls to move the iterator position.
20663: (12)                     A negative return value means we could not move as requested.
Otherwise:
20664: (12)                     0 = the final position has a valid item.
20665: (12)                     1 = not a valid item, but we can insert an item here.
20666: (8)                  """
20667: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_next(self)
20668: (4)              def fz_outline_iterator_prev(self):
20669: (8)                  r""" Class-aware wrapper for `::fz_outline_iterator_prev()`."""
20670: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_prev(self)
20671: (4)              def fz_outline_iterator_up(self):
20672: (8)                  r""" Class-aware wrapper for `::fz_outline_iterator_up()`."""
20673: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_up(self)
20674: (4)              def fz_outline_iterator_insert(self, item):
20675: (8)                  r""" Custom wrapper for fz_outline_iterator_insert()."""
20676: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_insert(self, item)
20677: (4)              def fz_outline_iterator_update(self, item):
20678: (8)                  r""" Custom wrapper for fz_outline_iterator_update()."""
20679: (8)                  return _mupdf.FzOutlineIterator_fz_outline_iterator_update(self, item)
20680: (4)              def __init__(self, *args):
20681: (8)                  r"""
20682: (8)                  *Overload 1:*
20683: (9)                   == Constructors.  Constructor using `fz_new_outline_iterator()`.
20684: (16)                         Get an iterator for the document outline.
20685: (16)                         Should be freed by fz_drop_outline_iterator.
20686: (8)                  |
20687: (8)                  *Overload 2:*
20688: (9)                   Constructor using `fz_new_outline_iterator_of_size()`.
```

```
20689: (8)                  |
20690: (8)                  *Overload 3:*
20691: (9)                   Constructor using `pdf_new_outline_iterator()`.
20692: (8)                  |
20693: (8)                  *Overload 4:*
20694: (9)                   Default constructor, sets `m_internal` to null.
20695: (8)                  |
20696: (8)                  *Overload 5:*
20697: (9)                   Constructor using raw copy of pre-existing `::fz_outline_iterator`.
20698: (8)                  """
20699: (8)                  _mupdf.FzOutlineIterator_swiginit(self,
_mupdf.new_FzOutlineIterator(*args))
20700: (4)              __swig_destroy__ = _mupdf.delete_FzOutlineIterator
20701: (4)              def m_internal_value(self):
20702: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
20703: (8)                  return _mupdf.FzOutlineIterator_m_internal_value(self)
20704: (4)              m_internal = property(_mupdf.FzOutlineIterator_m_internal_get,
_mupdf.FzOutlineIterator_m_internal_set, doc=r""" Pointer to wrapped data.""")
20705: (4)              s_num_instances = property(_mupdf.FzOutlineIterator_s_num_instances_get,
_mupdf.FzOutlineIterator_s_num_instances_set)
20706: (0)          _mupdf.FzOutlineIterator_swigregister(FzOutlineIterator)
20707: (0)          class FzOutput(object):
20708: (4)              r""" Wrapper class for struct `fz_output`. Not copyable or assignable."""
20709: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
20710: (4)              __repr__ = _swig_repr
20711: (4)              Fixed_STDOUT = _mupdf.FzOutput_Fixed_STDOUT
20712: (4)              Fixed_STDERR = _mupdf.FzOutput_Fixed_STDERR
20713: (4)              Filter_HEX = _mupdf.FzOutput_Filter_HEX
20714: (4)              Filter_85 = _mupdf.FzOutput_Filter_85
20715: (4)              Filter_RLE = _mupdf.FzOutput_Filter_RLE
20716: (4)              def fz_close_output(self):
20717: (8)                  r"""
20718: (8)                  Class-aware wrapper for `::fz_close_output()`.
20719: (12)                     Flush pending output and close an output stream.
20720: (8)                  """
20721: (8)                  return _mupdf.FzOutput_fz_close_output(self)
20722: (4)              def fz_debug_store(self):
20723: (8)                  r"""
20724: (8)                  Class-aware wrapper for `::fz_debug_store()`.
20725: (12)                     Output debugging information for the current state of the store
20726: (12)                     to the given output channel.
20727: (8)                  """
20728: (8)                  return _mupdf.FzOutput_fz_debug_store(self)
20729: (4)              def fz_dump_glyph_cache_stats(self):
20730: (8)                  r"""
20731: (8)                  Class-aware wrapper for `::fz_dump_glyph_cache_stats()`.
20732: (12)                     Dump debug statistics for the glyph cache.
20733: (8)                  """
20734: (8)                  return _mupdf.FzOutput_fz_dump_glyph_cache_stats(self)
20735: (4)              def fz_flush_output(self):
20736: (8)                  r"""
20737: (8)                  Class-aware wrapper for `::fz_flush_output()`.
20738: (12)                     Flush unwritten data.
20739: (8)                  """
20740: (8)                  return _mupdf.FzOutput_fz_flush_output(self)
20741: (4)              def fz_new_svg_device(self, page_width, page_height, text_format,
reuse_images):
20742: (8)                  r"""
20743: (8)                  Class-aware wrapper for `::fz_new_svg_device()`.
20744: (12)                     Create a device that outputs (single page) SVG files to
20745: (12)                     the given output stream.
20746: (12)                     Equivalent to fz_new_svg_device_with_id passing id = NULL.
20747: (8)                  """
20748: (8)                  return _mupdf.FzOutput_fz_new_svg_device(self, page_width,
page_height, text_format, reuse_images)
20749: (4)              def fz_new_svg_device_with_id(self, page_width, page_height, text_format,
reuse_images, id):
```

```
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_svg_device_with_id()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_new_svg_device_with_id(float page_width, float page_height,
int text_format, int reuse_images)` => `(fz_device *, int id)`
(12)                    Create a device that outputs (single page) SVG files to
(12)                    the given output stream.
(12)                    output: The output stream to send the constructed SVG page to.
(12)                    page_width, page_height: The page dimensions to use (in points).
(12)                    text_format: How to emit text. One of the following values:
(16)                        FZ_SVG_TEXT_AS_TEXT: As <text> elements with possible
(16)                        layout errors and mismatching fonts.
(16)                        FZ_SVG_TEXT_AS_PATH: As <path> elements with exact
(16)                        visual appearance.
(12)                    reuse_images: Share image resources using <symbol> definitions.
(12)                    id: ID parameter to keep generated IDs unique across SVG files.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_new_svg_device_with_id(self, page_width,
page_height, text_format, reuse_images, id)
(4)             def fz_new_trace_device(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_trace_device()`.
(12)                    Create a device to print a debug trace of all device calls.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_new_trace_device(self)
(4)             def fz_new_xmltext_device(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_xmltext_device()`.
(12)                    Create a device to output raw information.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_new_xmltext_device(self)
(4)             def fz_output_supports_stream(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_output_supports_stream()`.
(12)                    Query whether a given fz_output supports fz_stream_from_output.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_output_supports_stream(self)
(4)             def fz_output_xml(self, item, level):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_output_xml()`.
(12)                    Pretty-print an XML tree to given output.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_output_xml(self, item, level)
(4)             def fz_print_stext_header_as_html(self):
(8)                 r""" Class-aware wrapper for `::fz_print_stext_header_as_html()`."""
(8)                 return _mupdf.FzOutput_fz_print_stext_header_as_html(self)
(4)             def fz_print_stext_header_as_xhtml(self):
(8)                 r""" Class-aware wrapper for `::fz_print_stext_header_as_xhtml()`."""
(8)                 return _mupdf.FzOutput_fz_print_stext_header_as_xhtml(self)
(4)             def fz_print_stext_page_as_html(self, page, id):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_print_stext_page_as_html()`.
(12)                    Output structured text to a file in HTML (visual) format.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_print_stext_page_as_html(self, page, id)
(4)             def fz_print_stext_page_as_json(self, page, scale):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_print_stext_page_as_json()`.
(12)                    Output structured text to a file in JSON format.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_print_stext_page_as_json(self, page, scale)
(4)             def fz_print_stext_page_as_text(self, page):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_print_stext_page_as_text()`.
(12)                    Output structured text to a file in plain-text UTF-8 format.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_print_stext_page_as_text(self, page)
(4)             def fz_print_stext_page_as_xhtml(self, page, id):
(8)                 r"""
```

```
(8)                 Class-aware wrapper for `::fz_print_stext_page_as_xhtml()`.
(12)                    Output structured text to a file in XHTML (semantic) format.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_print_stext_page_as_xhtml(self, page, id)
(4)             def fz_print_stext_page_as_xml(self, page, id):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_print_stext_page_as_xml()`.
(12)                    Output structured text to a file in XML format.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_print_stext_page_as_xml(self, page, id)
(4)             def fz_print_stext_trailer_as_html(self):
(8)                 r""" Class-aware wrapper for `::fz_print_stext_trailer_as_html()`."""
(8)                 return _mupdf.FzOutput_fz_print_stext_trailer_as_html(self)
(4)             def fz_print_stext_trailer_as_xhtml(self):
(8)                 r""" Class-aware wrapper for `::fz_print_stext_trailer_as_xhtml()`."""
(8)                 return _mupdf.FzOutput_fz_print_stext_trailer_as_xhtml(self)
(4)             def fz_reset_output(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_reset_output()`.
(12)                    Reset a closed output stream. Returns state to
(12)                    (broadly) that which it was in when opened. Not
(12)                    all outputs can be reset, so this may throw an
(12)                    exception.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_reset_output(self)
(4)             def fz_seek_output(self, off, whence):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_seek_output()`.
(12)                    Seek to the specified file position.
(12)                    See fseek for arguments.
(12)                    Throw an error on unseekable outputs.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_seek_output(self, off, whence)
(4)             def fz_set_stddbg(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_set_stddbg()`.
(12)                    Set the output stream to be used for fz_stddbg. Set to NULL to
(12)                    reset to default (stderr).
(8)                 """
(8)                 return _mupdf.FzOutput_fz_set_stddbg(self)
(4)             def fz_stream_from_output(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_stream_from_output()`.
(12)                    Obtain the fz_output in the form of a fz_stream.
(12)                    This allows data to be read back from some forms of fz_output
(12)                    object. When finished reading, the fz_stream should be released
(12)                    by calling fz_drop_stream. Until the fz_stream is dropped, no
(12)                    further operations should be performed on the fz_output object.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_stream_from_output(self)
(4)             def fz_tell_output(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_tell_output()`.
(12)                    Return the current file position.
(12)                    Throw an error on untellable outputs.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_tell_output(self)
(4)             def fz_truncate_output(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_truncate_output()`.
(12)                    Truncate the output at the current position.
(12)                    This allows output streams which have seeked back from the end
(12)                    of their storage to be truncated at the current point.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_truncate_output(self)
(4)             def fz_write_base64(self, data, size, newline):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_base64()`.
(12)                    Write a base64 encoded data block, optionally with periodic
```

```
(12)                    newlines.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_base64(self, data, size, newline)
(4)             def fz_write_base64_buffer(self, data, newline):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_base64_buffer()`.
(12)                    Write a base64 encoded fz_buffer, optionally with periodic
(12)                    newlines.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_base64_buffer(self, data, newline)
(4)             def fz_write_bitmap_as_pbm(self, bitmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bitmap_as_pbm()`.
(12)                    Write a bitmap as a pbm.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bitmap_as_pbm(self, bitmap)
(4)             def fz_write_bitmap_as_pcl(self, bitmap, pcl):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bitmap_as_pcl()`.
(12)                    Write a bitmap as mono PCL.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bitmap_as_pcl(self, bitmap, pcl)
(4)             def fz_write_bitmap_as_pkm(self, bitmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bitmap_as_pkm()`.
(12)                    Write a CMYK bitmap as a pkm.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bitmap_as_pkm(self, bitmap)
(4)             def fz_write_bitmap_as_pwg(self, bitmap, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bitmap_as_pwg()`.
(12)                    Write a bitmap as a PWG.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bitmap_as_pwg(self, bitmap, pwg)
(4)             def fz_write_bitmap_as_pwg_page(self, bitmap, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bitmap_as_pwg_page()`.
(12)                    Write a bitmap as a PWG page.
(12)                    Caller should provide a file header by calling
(12)                    fz_write_pwg_file_header, but can then write several pages to
(12)                    the same file.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bitmap_as_pwg_page(self, bitmap, pwg)
(4)             def fz_write_bits(self, data, num_bits):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bits()`.
(12)                    Write num_bits of data to the end of the output stream, assumed to
be packed
(12)                    most significant bits first.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bits(self, data, num_bits)
(4)             def fz_write_bits_sync(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_bits_sync()`.
(12)                    Sync to byte boundary after writing bits.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_bits_sync(self)
(4)             def fz_write_buffer(self, data):
(8)                 r""" Class-aware wrapper for `::fz_write_buffer()`."""
(8)                 return _mupdf.FzOutput_fz_write_buffer(self, data)
(4)             def fz_write_byte(self, x):
(8)                 r""" Class-aware wrapper for `::fz_write_byte()`."""
(8)                 return _mupdf.FzOutput_fz_write_byte(self, x)
(4)             def fz_write_char(self, x):
(8)                 r""" Class-aware wrapper for `::fz_write_char()`."""
(8)                 return _mupdf.FzOutput_fz_write_char(self, x)
(4)             def fz_write_data(self, data, size):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_data()`.
```

```
(12)                    Write data to output.
(12)                    data: Pointer to data to write.
(12)                    size: Size of data to write in bytes.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_data(self, data, size)
(4)             def fz_write_float_be(self, f):
(8)                 r""" Class-aware wrapper for `::fz_write_float_be()`."""
(8)                 return _mupdf.FzOutput_fz_write_float_be(self, f)
(4)             def fz_write_float_le(self, f):
(8)                 r""" Class-aware wrapper for `::fz_write_float_le()`."""
(8)                 return _mupdf.FzOutput_fz_write_float_le(self, f)
(4)             def fz_write_image_as_data_uri(self, image):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_image_as_data_uri()`.
(12)                    Write image as a data URI (for HTML and SVG output).
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_image_as_data_uri(self, image)
(4)             def fz_write_int16_be(self, x):
(8)                 r""" Class-aware wrapper for `::fz_write_int16_be()`."""
(8)                 return _mupdf.FzOutput_fz_write_int16_be(self, x)
(4)             def fz_write_int16_le(self, x):
(8)                 r""" Class-aware wrapper for `::fz_write_int16_le()`."""
(8)                 return _mupdf.FzOutput_fz_write_int16_le(self, x)
(4)             def fz_write_int32_be(self, x):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_int32_be()`.
(12)                    Write different sized data to an output stream.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_int32_be(self, x)
(4)             def fz_write_int32_le(self, x):
(8)                 r""" Class-aware wrapper for `::fz_write_int32_le()`."""
(8)                 return _mupdf.FzOutput_fz_write_int32_le(self, x)
(4)             def fz_write_pixmap_as_data_uri(self, pixmap):
(8)                 r""" Class-aware wrapper for `::fz_write_pixmap_as_data_uri()`."""
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_data_uri(self, pixmap)
(4)             def fz_write_pixmap_as_jpeg(self, pix, quality, invert_cmyk):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_jpeg()`.
(12)                    Write a pixmap as a JPEG.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_jpeg(self, pix, quality,
invert_cmyk)
(4)             def fz_write_pixmap_as_jpx(self, pix, quality):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_jpx()`.
(12)                    Pixmap data as JP2K with no subsampling.
(12)                    quality = 100 = lossless
(12)                    otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_jpx(self, pix, quality)
(4)             def fz_write_pixmap_as_pam(self, pixmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pam()`.
(12)                    Write a pixmap as a pnm (greyscale, rgb or cmyk, with or without
(12)                    alpha).
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pam(self, pixmap)
(4)             def fz_write_pixmap_as_pcl(self, pixmap, pcl):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pcl()`.
(12)                    Write an (RGB) pixmap as color PCL.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pcl(self, pixmap, pcl)
(4)             def fz_write_pixmap_as_pclm(self, pixmap, options):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pclm()`.
(12)                    Write a (Greyscale or RGB) pixmap as pclm.
(8)                 """
```

```
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pclm(self, pixmap, options)
(4)             def fz_write_pixmap_as_pdfocr(self, pixmap, options):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pdfocr()`.
(12)                    Write a (Greyscale or RGB) pixmap as pdfocr.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pdfocr(self, pixmap, options)
(4)             def fz_write_pixmap_as_png(self, pixmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_png()`.
(12)                    Write a (Greyscale or RGB) pixmap as a png.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_png(self, pixmap)
(4)             def fz_write_pixmap_as_pnm(self, pixmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pnm()`.
(12)                    Write a pixmap as a pnm (greyscale or rgb, no alpha).
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pnm(self, pixmap)
(4)             def fz_write_pixmap_as_ps(self, pixmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_ps()`.
(12)                    Write a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_ps(self, pixmap)
(4)             def fz_write_pixmap_as_psd(self, pixmap):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_psd()`.
(12)                    Write a pixmap as a PSD file.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_psd(self, pixmap)
(4)             def fz_write_pixmap_as_pwg(self, pixmap, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pwg()`.
(12)                    Write a pixmap as a PWG.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pwg(self, pixmap, pwg)
(4)             def fz_write_pixmap_as_pwg_page(self, pixmap, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pixmap_as_pwg_page()`.
(12)                    Write a pixmap as a PWG page.
(12)                    Caller should provide a file header by calling
(12)                    fz_write_pwg_file_header, but can then write several pages to
(12)                    the same file.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pixmap_as_pwg_page(self, pixmap, pwg)
(4)             def fz_write_ps_file_header(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_ps_file_header()`.
(12)                    Write the file level header for ps band writer output.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_ps_file_header(self)
(4)             def fz_write_ps_file_trailer(self, pages):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_ps_file_trailer()`.
(12)                    Write the file level trailer for ps band writer output.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_ps_file_trailer(self, pages)
(4)             def fz_write_pwg_file_header(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_pwg_file_header()`.
(12)                    Output the file header to a pwg stream, ready for pages to follow it.
(8)                 """
(8)                 return _mupdf.FzOutput_fz_write_pwg_file_header(self)
(4)             def fz_write_rune(self, rune):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_write_rune()`.
```

```
21088: (12)                     Write a UTF-8 encoded unicode character.
21089: (8)                  """
21090: (8)                  return _mupdf.FzOutput_fz_write_rune(self, rune)
21091: (4)              def fz_write_stream(self, _in):
21092: (8)                  r"""
21093: (8)                  Class-aware wrapper for `::fz_write_stream()`.
21094: (12)                     Copy the stream contents to the output.
21095: (8)                  """
21096: (8)                  return _mupdf.FzOutput_fz_write_stream(self, _in)
21097: (4)              def fz_write_string(self, s):
21098: (8)                  r"""
21099: (8)                  Class-aware wrapper for `::fz_write_string()`.
21100: (12)                     Write a string. Does not write zero terminator.
21101: (8)                  """
21102: (8)                  return _mupdf.FzOutput_fz_write_string(self, s)
21103: (4)              def fz_write_uint16_be(self, x):
21104: (8)                  r""" Class-aware wrapper for `::fz_write_uint16_be()`."""
21105: (8)                  return _mupdf.FzOutput_fz_write_uint16_be(self, x)
21106: (4)              def fz_write_uint16_le(self, x):
21107: (8)                  r""" Class-aware wrapper for `::fz_write_uint16_le()`."""
21108: (8)                  return _mupdf.FzOutput_fz_write_uint16_le(self, x)
21109: (4)              def fz_write_uint32_be(self, x):
21110: (8)                  r""" Class-aware wrapper for `::fz_write_uint32_be()`."""
21111: (8)                  return _mupdf.FzOutput_fz_write_uint32_be(self, x)
21112: (4)              def fz_write_uint32_le(self, x):
21113: (8)                  r""" Class-aware wrapper for `::fz_write_uint32_le()`."""
21114: (8)                  return _mupdf.FzOutput_fz_write_uint32_le(self, x)
21115: (4)              def pdf_new_output_processor(self, ahxencode, newlines):
21116: (8)                  r""" Class-aware wrapper for `::pdf_new_output_processor()`."""
21117: (8)                  return _mupdf.FzOutput_pdf_new_output_processor(self, ahxencode,
newlines)
21118: (4)              def pdf_print_crypt(self, crypt):
21119: (8)                  r""" Class-aware wrapper for `::pdf_print_crypt()`."""
21120: (8)                  return _mupdf.FzOutput_pdf_print_crypt(self, crypt)
21121: (4)              def pdf_print_encrypted_obj(self, obj, tight, ascii, crypt, num, gen,
sep):
21122: (8)                  r"""
21123: (8)                  Class-aware wrapper for `::pdf_print_encrypted_obj()`.
21124: (8)                  This method has out-params. Python/C# wrappers look like:
21125: (12)                     `pdf_print_encrypted_obj(::pdf_obj *obj, int tight, int ascii,
::pdf_crypt *crypt, int num, int gen)` => int sep
21126: (8)                  """
21127: (8)                  return _mupdf.FzOutput_pdf_print_encrypted_obj(self, obj, tight,
ascii, crypt, num, gen, sep)
21128: (4)              def pdf_print_font(self, fontdesc):
21129: (8)                  r""" Class-aware wrapper for `::pdf_print_font()`."""
21130: (8)                  return _mupdf.FzOutput_pdf_print_font(self, fontdesc)
21131: (4)              def pdf_print_obj(self, obj, tight, ascii):
21132: (8)                  r""" Class-aware wrapper for `::pdf_print_obj()`."""
21133: (8)                  return _mupdf.FzOutput_pdf_print_obj(self, obj, tight, ascii)
21134: (4)              def pdf_write_digest(self, byte_range, field, digest_offset,
digest_length, signer):
21135: (8)                  r""" Class-aware wrapper for `::pdf_write_digest()`."""
21136: (8)                  return _mupdf.FzOutput_pdf_write_digest(self, byte_range, field,
digest_offset, digest_length, signer)
21137: (4)              def __init__(self, *args):
21138: (8)                  r"""
21139: (8)                  *Overload 1:*
21140: (9)                   == Constructors.  Constructor using `fz_new_arc4_output()`.
21141: (8)                  |
21142: (8)                  *Overload 2:*
21143: (9)                   Constructor using `fz_new_deflate_output()`.
21144: (8)                  |
21145: (8)                  *Overload 3:*
21146: (9)                   Constructor using `fz_new_log_for_module()`.
21147: (16)                         Internal function to actually do the opening of the logfile.
21148: (16)                         Caller should close/drop the output when finished with it.
21149: (8)                  |
21150: (8)                  *Overload 4:*
```

```
21151: (9)                   Constructor using `fz_new_output()`.
21152: (16)                         Create a new output object with the given
21153: (16)                         internal state and function pointers.
21154: (16)                         state: Internal state (opaque to everything but
implementation).
21155: (16)                         write: Function to output a given buffer.
21156: (16)                         close: Cleanup function to destroy state when output closed.
21157: (16)                         May permissibly be null.
21158: (8)                   |
21159: (8)                   *Overload 5:*
21160: (9)                   Constructor using `fz_new_output_with_buffer()`.
21161: (16)                         Open an output stream that appends
21162: (16)                         to a buffer.
21163: (16)                         buf: The buffer to append to.
21164: (8)                   |
21165: (8)                   *Overload 6:*
21166: (9)                   Constructor using `fz_new_output_with_file_ptr()`.
21167: (16)                         Open an output stream that writes to a
21168: (16)                         given FILE *.
21169: (16)                         file: The file pointers to write to. NULL is interpreted as
effectively
21170: (16)                         meaning /dev/null or similar.
21171: (8)                   |
21172: (8)                   *Overload 7:*
21173: (9)                   Constructor using `fz_new_output_with_path()`.
21174: (16)                         Open an output stream that writes to a
21175: (16)                         given path.
21176: (16)                         filename: The filename to write to (specified in UTF-8).
21177: (16)                         append: non-zero if we should append to the file, rather than
21178: (16)                         overwriting it.
21179: (8)                   |
21180: (8)                   *Overload 8:*
21181: (9)                   Uses fz_stdout() or fz_stderr().
21182: (8)                   |
21183: (8)                   *Overload 9:*
21184: (9)                   Calls one of: fz_new_asciihex_output(), fz_new_ascii85_output(),
fz_new_rle_output().
21185: (8)                   |
21186: (8)                   *Overload 10:*
21187: (9)                   Constructor using raw copy of pre-existing `::fz_output`.
21188: (8)                   |
21189: (8)                   *Overload 11:*
21190: (9)                   Constructor using raw copy of pre-existing `::fz_output`.
21191: (8)                   """
21192: (8)                   _mupdf.FzOutput_swiginit(self, _mupdf.new_FzOutput(*args))
21193: (4)               __swig_destroy__ = _mupdf.delete_FzOutput
21194: (4)               def m_internal_value(self):
21195: (8)                   r""" Return numerical value of .m_internal; helps with Python
debugging."""
21196: (8)                   return _mupdf.FzOutput_m_internal_value(self)
21197: (4)               m_internal = property(_mupdf.FzOutput_m_internal_get,
_mupdf.FzOutput_m_internal_set, doc=r""" Pointer to wrapped data.""")
21198: (4)               s_num_instances = property(_mupdf.FzOutput_s_num_instances_get,
_mupdf.FzOutput_s_num_instances_set)
21199: (0)           _mupdf.FzOutput_swigregister(FzOutput)
21200: (0)           class FzOutput2(FzOutput):
21201: (4)               r""" Wrapper class for struct fz_output with virtual fns for each fnptr;
this is for use as a SWIG Director class."""
21202: (4)               thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21203: (4)               __repr__ = _swig_repr
21204: (4)               def __init__(self):
21205: (8)                   r""" == Constructor."""
21206: (8)                   if self.__class__ == FzOutput2:
21207: (12)                      _self = None
21208: (8)                   else:
21209: (12)                      _self = self
21210: (8)                   _mupdf.FzOutput2_swiginit(self, _mupdf.new_FzOutput2(_self, ))
21211: (4)               def use_virtual_write(self, use=True):
```

```
21212: (8)                  r"""
21213: (8)                  These methods set the function pointers in *m_internal
21214: (8)                  to point to internal callbacks that call our virtual methods.
21215: (8)                  """
21216: (8)                  return _mupdf.FzOutput2_use_virtual_write(self, use)
21217: (4)              def use_virtual_seek(self, use=True):
21218: (8)                  return _mupdf.FzOutput2_use_virtual_seek(self, use)
21219: (4)              def use_virtual_tell(self, use=True):
21220: (8)                  return _mupdf.FzOutput2_use_virtual_tell(self, use)
21221: (4)              def use_virtual_close(self, use=True):
21222: (8)                  return _mupdf.FzOutput2_use_virtual_close(self, use)
21223: (4)              def use_virtual_drop(self, use=True):
21224: (8)                  return _mupdf.FzOutput2_use_virtual_drop(self, use)
21225: (4)              def use_virtual_reset(self, use=True):
21226: (8)                  return _mupdf.FzOutput2_use_virtual_reset(self, use)
21227: (4)              def use_virtual_as_stream(self, use=True):
21228: (8)                  return _mupdf.FzOutput2_use_virtual_as_stream(self, use)
21229: (4)              def use_virtual_truncate(self, use=True):
21230: (8)                  return _mupdf.FzOutput2_use_virtual_truncate(self, use)
21231: (4)              def write(self, arg_0, arg_2, arg_3):
21232: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
21233: (8)                  return _mupdf.FzOutput2_write(self, arg_0, arg_2, arg_3)
21234: (4)              def seek(self, arg_0, arg_2, arg_3):
21235: (8)                  return _mupdf.FzOutput2_seek(self, arg_0, arg_2, arg_3)
21236: (4)              def tell(self, arg_0):
21237: (8)                  return _mupdf.FzOutput2_tell(self, arg_0)
21238: (4)              def close(self, arg_0):
21239: (8)                  return _mupdf.FzOutput2_close(self, arg_0)
21240: (4)              def drop(self, arg_0):
21241: (8)                  return _mupdf.FzOutput2_drop(self, arg_0)
21242: (4)              def reset(self, arg_0):
21243: (8)                  return _mupdf.FzOutput2_reset(self, arg_0)
21244: (4)              def as_stream(self, arg_0):
21245: (8)                  return _mupdf.FzOutput2_as_stream(self, arg_0)
21246: (4)              def truncate(self, arg_0):
21247: (8)                  return _mupdf.FzOutput2_truncate(self, arg_0)
21248: (4)              __swig_destroy__ = _mupdf.delete_FzOutput2
21249: (4)              def __disown__(self):
21250: (8)                  self.this.disown()
21251: (8)                  _mupdf.disown_FzOutput2(self)
21252: (8)                  return weakref.proxy(self)
21253: (0)          _mupdf.FzOutput2_swigregister(FzOutput2)
21254: (0)          class FzOverprint(object):
21255: (4)              r"""
21256: (4)              Wrapper class for struct `fz_overprint`. Not copyable or assignable.
21257: (4)              Pixmaps represent a set of pixels for a 2 dimensional region of
21258: (4)              a plane. Each pixel has n components per pixel. The components
21259: (4)              are in the order process-components, spot-colors, alpha, where
21260: (4)              there can be 0 of any of those types. The data is in
21261: (4)              premultiplied alpha when rendering, but non-premultiplied for
21262: (4)              colorspace conversions and rescaling.
21263: (4)              """
21264: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21265: (4)              __repr__ = _swig_repr
21266: (4)              def __init__(self, *args):
21267: (8)                  r"""
21268: (8)                  *Overload 1:*
21269: (8)                  Default constructor, sets `m_internal` to null.
21270: (8)                  |
21271: (8)                  *Overload 2:*
21272: (8)                  Constructor using raw copy of pre-existing `::fz_overprint`.
21273: (8)                  """
21274: (8)                  _mupdf.FzOverprint_swiginit(self, _mupdf.new_FzOverprint(*args))
21275: (4)              __swig_destroy__ = _mupdf.delete_FzOverprint
21276: (4)              def m_internal_value(self):
21277: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
```

```
21278: (8)                  return _mupdf.FzOverprint_m_internal_value(self)
21279: (4)              m_internal = property(_mupdf.FzOverprint_m_internal_get,
_mupdf.FzOverprint_m_internal_set, doc=r""" Pointer to wrapped data.""")
21280: (4)              s_num_instances = property(_mupdf.FzOverprint_s_num_instances_get,
_mupdf.FzOverprint_s_num_instances_set)
21281: (0)          _mupdf.FzOverprint_swigregister(FzOverprint)
21282: (0)          class FzPage(object):
21283: (4)              r""" Wrapper class for struct `fz_page`."""
21284: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21285: (4)              __repr__ = _swig_repr
21286: (4)              def fz_bound_page(self):
21287: (8)                  r"""
21288: (8)                  Class-aware wrapper for `::fz_bound_page()`.
21289: (12)                     Determine the size of a page at 72 dpi.
21290: (8)                  """
21291: (8)                  return _mupdf.FzPage_fz_bound_page(self)
21292: (4)              def fz_bound_page_box(self, box):
21293: (8)                  r""" Class-aware wrapper for `::fz_bound_page_box()`."""
21294: (8)                  return _mupdf.FzPage_fz_bound_page_box(self, box)
21295: (4)              def fz_create_link(self, bbox, uri):
21296: (8)                  r"""
21297: (8)                  Class-aware wrapper for `::fz_create_link()`.
21298: (12)                     Create a new link on a page.
21299: (8)                  """
21300: (8)                  return _mupdf.FzPage_fz_create_link(self, bbox, uri)
21301: (4)              def fz_delete_link(self, link):
21302: (8)                  r"""
21303: (8)                  Class-aware wrapper for `::fz_delete_link()`.
21304: (12)                     Delete an existing link on a page.
21305: (8)                  """
21306: (8)                  return _mupdf.FzPage_fz_delete_link(self, link)
21307: (4)              def fz_load_links(self):
21308: (8)                  r"""
21309: (8)                  Class-aware wrapper for `::fz_load_links()`.
21310: (12)                     Load the list of links for a page.
21311: (12)                     Returns a linked list of all the links on the page, each with
21312: (12)                     its clickable region and link destination. Each link is
21313: (12)                     reference counted so drop and free the list of links by
21314: (12)                     calling fz_drop_link on the pointer return from fz_load_links.
21315: (12)                     page: Page obtained from fz_load_page.
21316: (8)                  """
21317: (8)                  return _mupdf.FzPage_fz_load_links(self)
21318: (4)              def fz_new_buffer_from_page(self, options):
21319: (8)                  r""" Class-aware wrapper for `::fz_new_buffer_from_page()`."""
21320: (8)                  return _mupdf.FzPage_fz_new_buffer_from_page(self, options)
21321: (4)              def fz_new_buffer_from_page_with_format(self, format, options, transform,
cookie):
21322: (8)                  r"""
21323: (8)                  Class-aware wrapper for `::fz_new_buffer_from_page_with_format()`.
21324: (12)                     Returns an fz_buffer containing a page after conversion to
specified format.
21325: (12)                     page: The page to convert.
21326: (12)                     format, options: Passed to fz_new_document_writer_with_output()
internally.
21327: (12)                     transform, cookie: Passed to fz_run_page() internally.
21328: (8)                  """
21329: (8)                  return _mupdf.FzPage_fz_new_buffer_from_page_with_format(self, format,
options, transform, cookie)
21330: (4)              def fz_new_display_list_from_page(self):
21331: (8)                  r"""
21332: (8)                  Class-aware wrapper for `::fz_new_display_list_from_page()`.
21333: (12)                     Create a display list.
21334: (12)                     Ownership of the display list is returned to the caller.
21335: (8)                  """
21336: (8)                  return _mupdf.FzPage_fz_new_display_list_from_page(self)
21337: (4)              def fz_new_display_list_from_page_contents(self):
21338: (8)                  r"""
21339: (8)                  Class-aware wrapper for `::fz_new_display_list_from_page_contents()`.
```

```
21340: (12)                     Create a display list from page contents (no annotations).
21341: (12)                     Ownership of the display list is returned to the caller.
21342: (8)                  """
21343: (8)                  return _mupdf.FzPage_fz_new_display_list_from_page_contents(self)
21344: (4)              def fz_new_pixmap_from_page(self, ctm, cs, alpha):
21345: (8)                  r""" Class-aware wrapper for `::fz_new_pixmap_from_page()`."""
21346: (8)                  return _mupdf.FzPage_fz_new_pixmap_from_page(self, ctm, cs, alpha)
21347: (4)              def fz_new_pixmap_from_page_contents(self, ctm, cs, alpha):
21348: (8)                  r"""
21349: (8)                  Class-aware wrapper for `::fz_new_pixmap_from_page_contents()`.
21350: (12)                     Render the page contents without annotations.
21351: (12)                     Ownership of the pixmap is returned to the caller.
21352: (8)                  """
21353: (8)                  return _mupdf.FzPage_fz_new_pixmap_from_page_contents(self, ctm, cs,
alpha)
21354: (4)              def fz_new_pixmap_from_page_contents_with_separations(self, ctm, cs, seps,
alpha):
21355: (8)                  r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_contents_with_separations()`."""
21356: (8)                  return
_mupdf.FzPage_fz_new_pixmap_from_page_contents_with_separations(self, ctm, cs, seps, alpha)
21357: (4)              def fz_new_pixmap_from_page_with_separations(self, ctm, cs, seps, alpha):
21358: (8)                  r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_with_separations()`."""
21359: (8)                  return _mupdf.FzPage_fz_new_pixmap_from_page_with_separations(self,
ctm, cs, seps, alpha)
21360: (4)              def fz_page_label(self, buf, size):
21361: (8)                  r"""
21362: (8)                  Class-aware wrapper for `::fz_page_label()`.
21363: (12)                     Get page label for a given page.
21364: (8)                  """
21365: (8)                  return _mupdf.FzPage_fz_page_label(self, buf, size)
21366: (4)              def fz_page_presentation(self, transition, duration):
21367: (8)                  r"""
21368: (8)                  Class-aware wrapper for `::fz_page_presentation()`.
21369: (8)                  This method has out-params. Python/C# wrappers look like:
21370: (12)                     `fz_page_presentation(::fz_transition *transition)` =>
`(fz_transition *, float duration)`
21371: (12)                     Get the presentation details for a given page.
21372: (12)                     transition: A pointer to a transition struct to fill out.
21373: (12)                     duration: A pointer to a place to set the page duration in
21374: (12)                     seconds. Will be set to 0 if no transition is specified for the
21375: (12)                     page.
21376: (12)                     Returns: a pointer to the transition structure, or NULL if there
21377: (12)                     is no transition specified for the page.
21378: (8)                  """
21379: (8)                  return _mupdf.FzPage_fz_page_presentation(self, transition, duration)
21380: (4)              def fz_page_separations(self):
21381: (8)                  r"""
21382: (8)                  Class-aware wrapper for `::fz_page_separations()`.
21383: (12)                     Get the separations details for a page.
21384: (12)                     This will be NULL, unless the format specifically supports
21385: (12)                     separations (such as PDF files). May be NULL even
21386: (12)                     so, if there are no separations on a page.
21387: (12)                     Returns a reference that must be dropped.
21388: (8)                  """
21389: (8)                  return _mupdf.FzPage_fz_page_separations(self)
21390: (4)              def fz_page_uses_overprint(self):
21391: (8)                  r"""
21392: (8)                  Class-aware wrapper for `::fz_page_uses_overprint()`.
21393: (12)                     Query if a given page requires overprint.
21394: (8)                  """
21395: (8)                  return _mupdf.FzPage_fz_page_uses_overprint(self)
21396: (4)              def fz_run_page(self, dev, transform, cookie):
21397: (8)                  r"""
21398: (8)                  Class-aware wrapper for `::fz_run_page()`.
21399: (12)                     Run a page through a device.
21400: (12)                     page: Page obtained from fz_load_page.
21401: (12)                     dev: Device obtained from fz_new_*_device.
```

```
21402: (12)                     transform: Transform to apply to page. May include for example
21403: (12)                     scaling and rotation, see fz_scale, fz_rotate and fz_concat.
21404: (12)                     Set to fz_identity if no transformation is desired.
21405: (12)                     cookie: Communication mechanism between caller and library
21406: (12)                     rendering the page. Intended for multi-threaded applications,
21407: (12)                     while single-threaded applications set cookie to NULL. The
21408: (12)                     caller may abort an ongoing rendering of a page. Cookie also
21409: (12)                     communicates progress information back to the caller. The
21410: (12)                     fields inside cookie are continually updated while the page is
21411: (12)                     rendering.
21412: (8)                  """
21413: (8)                  return _mupdf.FzPage_fz_run_page(self, dev, transform, cookie)
21414: (4)              def fz_run_page_annots(self, dev, transform, cookie):
21415: (8)                  r"""
21416: (8)                  Class-aware wrapper for `::fz_run_page_annots()`.
21417: (12)                     Run the annotations on a page through a device.
21418: (8)                  """
21419: (8)                  return _mupdf.FzPage_fz_run_page_annots(self, dev, transform, cookie)
21420: (4)              def fz_run_page_contents(self, dev, transform, cookie):
21421: (8)                  r"""
21422: (8)                  Class-aware wrapper for `::fz_run_page_contents()`.
21423: (12)                     Run a page through a device. Just the main
21424: (12)                     page content, without the annotations, if any.
21425: (12)                     page: Page obtained from fz_load_page.
21426: (12)                     dev: Device obtained from fz_new_*_device.
21427: (12)                     transform: Transform to apply to page. May include for example
21428: (12)                     scaling and rotation, see fz_scale, fz_rotate and fz_concat.
21429: (12)                     Set to fz_identity if no transformation is desired.
21430: (12)                     cookie: Communication mechanism between caller and library
21431: (12)                     rendering the page. Intended for multi-threaded applications,
21432: (12)                     while single-threaded applications set cookie to NULL. The
21433: (12)                     caller may abort an ongoing rendering of a page. Cookie also
21434: (12)                     communicates progress information back to the caller. The
21435: (12)                     fields inside cookie are continually updated while the page is
21436: (12)                     rendering.
21437: (8)                  """
21438: (8)                  return _mupdf.FzPage_fz_run_page_contents(self, dev, transform,
cookie)
21439: (4)              def fz_run_page_widgets(self, dev, transform, cookie):
21440: (8)                  r"""
21441: (8)                  Class-aware wrapper for `::fz_run_page_widgets()`.
21442: (12)                     Run the widgets on a page through a device.
21443: (8)                  """
21444: (8)                  return _mupdf.FzPage_fz_run_page_widgets(self, dev, transform, cookie)
21445: (4)              def fz_search_page(self, needle, hit_mark, hit_bbox, hit_max):
21446: (8)                  r"""
21447: (8)                  Class-aware wrapper for `::fz_search_page()`.
21448: (8)                  This method has out-params. Python/C# wrappers look like:
21449: (12)                     `fz_search_page(const char *needle, ::fz_quad *hit_bbox, int
hit_max)` => `(int, int hit_mark)`
21450: (12)                     Search for the 'needle' text on the page.
21451: (12)                     Record the hits in the hit_bbox array and return the number of
21452: (12)                     hits. Will stop looking once it has filled hit_max rectangles.
21453: (8)                  """
21454: (8)                  return _mupdf.FzPage_fz_search_page(self, needle, hit_mark, hit_bbox,
hit_max)
21455: (4)              def pdf_page_from_fz_page(self):
21456: (8)                  r""" Class-aware wrapper for `::pdf_page_from_fz_page()`."""
21457: (8)                  return _mupdf.FzPage_pdf_page_from_fz_page(self)
21458: (4)              def doc(self):
21459: (8)                  r""" Returns wrapper for .doc member."""
21460: (8)                  return _mupdf.FzPage_doc(self)
21461: (4)              def __init__(self, *args):
21462: (8)                  r"""
21463: (8)                  *Overload 1:*
21464: (9)                   == Constructors.  Constructor using `fz_load_chapter_page()`.
21465: (16)                         Load a page.
21466: (16)                         After fz_load_page is it possible to retrieve the size of the
21467: (16)                         page using fz_bound_page, or to render the page using
```

```
21468: (16)                         fz_run_page_*. Free the page by calling fz_drop_page.
21469: (16)                         chapter: chapter number, 0 is the first chapter of the
document.
21470: (16)                         number: page number, 0 is the first page of the chapter.
21471: (8)                  |
21472: (8)                  *Overload 2:*
21473: (9)                   Constructor using `fz_load_page()`.
21474: (16)                         Load a given page number from a document. This may be much
less
21475: (16)                         efficient than loading by location (chapter+page) for some
21476: (16)                         document types.
21477: (8)                  |
21478: (8)                  *Overload 3:*
21479: (9)                   Constructor using `fz_new_page_of_size()`.
21480: (16)                         Different document types will be implemented by deriving from
21481: (16)                         fz_page. This macro allocates such derived structures, and
21482: (16)                         initialises the base sections.
21483: (8)                  |
21484: (8)                  *Overload 4:*
21485: (9)                   Return FzPage for pdfpage.m_internal.super.
21486: (8)                  |
21487: (8)                  *Overload 5:*
21488: (9)                   Copy constructor using `fz_keep_page()`.
21489: (8)                  |
21490: (8)                  *Overload 6:*
21491: (9)                   Default constructor, sets `m_internal` to null.
21492: (8)                  |
21493: (8)                  *Overload 7:*
21494: (9)                   Constructor using raw copy of pre-existing `::fz_page`.
21495: (8)                  """
21496: (8)                  _mupdf.FzPage_swiginit(self, _mupdf.new_FzPage(*args))
21497: (4)              __swig_destroy__ = _mupdf.delete_FzPage
21498: (4)              def m_internal_value(self):
21499: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
21500: (8)                  return _mupdf.FzPage_m_internal_value(self)
21501: (4)              m_internal = property(_mupdf.FzPage_m_internal_get,
_mupdf.FzPage_m_internal_set, doc=r""" Pointer to wrapped data.""")
21502: (4)              s_num_instances = property(_mupdf.FzPage_s_num_instances_get,
_mupdf.FzPage_s_num_instances_set)
21503: (0)          _mupdf.FzPage_swigregister(FzPage)
21504: (0)          class FzPath(object):
21505: (4)              r"""
21506: (5)               Wrapper class for struct `fz_path`.
21507: (4)              Vector path buffer.
21508: (4)              It can be stroked and dashed, or be filled.
21509: (4)              It has a fill rule (nonzero or even_odd).
21510: (4)              When rendering, they are flattened, stroked and dashed straight
21511: (4)              into the Global Edge List.
21512: (4)              """
21513: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21514: (4)              __repr__ = _swig_repr
21515: (4)              def fz_bound_path(self, stroke, ctm):
21516: (8)                  r"""
21517: (8)                  Class-aware wrapper for `::fz_bound_path()`.
21518: (12)                     Return a bounding rectangle for a path.
21519: (12)                     path: The path to bound.
21520: (12)                     stroke: If NULL, the bounding rectangle given is for
21521: (12)                     the filled path. If non-NULL the bounding rectangle
21522: (12)                     given is for the path stroked with the given attributes.
21523: (12)                     ctm: The matrix to apply to the path during stroking.
21524: (12)                     r: Pointer to a fz_rect which will be used to hold
21525: (12)                     the result.
21526: (12)                     Returns r, updated to contain the bounding rectangle.
21527: (8)                  """
21528: (8)                  return _mupdf.FzPath_fz_bound_path(self, stroke, ctm)
21529: (4)              def fz_clone_path(self):
21530: (8)                  r"""
```

```
(8)                 Class-aware wrapper for `::fz_clone_path()`.
(12)                    Clone the data for a path.
(12)                    This is used in preference to fz_keep_path when a whole
(12)                    new copy of a path is required, rather than just a shared
(12)                    pointer. This probably indicates that the path is about to
(12)                    be modified.
(12)                    path: path to clone.
(12)                    Throws exceptions on failure to allocate.
(8)                 """
(8)                 return _mupdf.FzPath_fz_clone_path(self)
(4)             def fz_closepath(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_closepath()`.
(12)                    Close the current subpath.
(12)                    path: The path to modify.
(12)                    Throws exceptions on failure to allocate, attempting to modify
(12)                    a packed path, and illegal path closes (i.e. closing a non open
(12)                    path).
(8)                 """
(8)                 return _mupdf.FzPath_fz_closepath(self)
(4)             def fz_currentpoint(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_currentpoint()`.
(12)                    Return the current point that a path has
(12)                    reached or (0,0) if empty.
(12)                    path: path to return the current point of.
(8)                 """
(8)                 return _mupdf.FzPath_fz_currentpoint(self)
(4)             def fz_curveto(self, x0, y0, x1, y1, x2, y2):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_curveto()`.
(12)                    Append a 'curveto' command to an open path. (For a
(12)                    cubic bezier).
(12)                    path: The path to modify.
(12)                    x0, y0: The coordinates of the first control point for the
(12)                    curve.
(12)                    x1, y1: The coordinates of the second control point for the
(12)                    curve.
(12)                    x2, y2: The end coordinates for the curve.
(12)                    Throws exceptions on failure to allocate, or attempting to
(12)                    modify a packed path.
(8)                 """
(8)                 return _mupdf.FzPath_fz_curveto(self, x0, y0, x1, y1, x2, y2)
(4)             def fz_curvetov(self, x1, y1, x2, y2):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_curvetov()`.
(12)                    Append a 'curvetov' command to an open path. (For a
(12)                    cubic bezier with the first control coordinate equal to
(12)                    the start point).
(12)                    path: The path to modify.
(12)                    x1, y1: The coordinates of the second control point for the
(12)                    curve.
(12)                    x2, y2: The end coordinates for the curve.
(12)                    Throws exceptions on failure to allocate, or attempting to
(12)                    modify a packed path.
(8)                 """
(8)                 return _mupdf.FzPath_fz_curvetov(self, x1, y1, x2, y2)
(4)             def fz_curvetoy(self, x0, y0, x2, y2):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_curvetoy()`.
(12)                    Append a 'curvetoy' command to an open path. (For a
(12)                    cubic bezier with the second control coordinate equal to
(12)                    the end point).
(12)                    path: The path to modify.
(12)                    x0, y0: The coordinates of the first control point for the
(12)                    curve.
(12)                    x2, y2: The end coordinates for the curve (and the second
(12)                    control coordinate).
(12)                    Throws exceptions on failure to allocate, or attempting to
```

```
21600: (12)                 modify a packed path.
21601: (8)              """
21602: (8)              return _mupdf.FzPath_fz_curvetoy(self, x0, y0, x2, y2)
21603: (4)          def fz_lineto(self, x, y):
21604: (8)              r"""
21605: (8)              Class-aware wrapper for `::fz_lineto()`.
21606: (12)                 Append a 'lineto' command to an open path.
21607: (12)                 path: The path to modify.
21608: (12)                 x, y: The coordinate to line to.
21609: (12)                 Throws exceptions on failure to allocate, or attempting to
21610: (12)                 modify a packed path.
21611: (8)              """
21612: (8)              return _mupdf.FzPath_fz_lineto(self, x, y)
21613: (4)          def fz_moveto(self, x, y):
21614: (8)              r"""
21615: (8)              Class-aware wrapper for `::fz_moveto()`.
21616: (12)                 Append a 'moveto' command to a path.
21617: (12)                 This 'opens' a path.
21618: (12)                 path: The path to modify.
21619: (12)                 x, y: The coordinate to move to.
21620: (12)                 Throws exceptions on failure to allocate, or attempting to
21621: (12)                 modify a packed path.
21622: (8)              """
21623: (8)              return _mupdf.FzPath_fz_moveto(self, x, y)
21624: (4)          def fz_packed_path_size(self):
21625: (8)              r"""
21626: (8)              Class-aware wrapper for `::fz_packed_path_size()`.
21627: (12)                 Return the number of bytes required to pack a path.
21628: (8)              """
21629: (8)              return _mupdf.FzPath_fz_packed_path_size(self)
21630: (4)          def fz_quadto(self, x0, y0, x1, y1):
21631: (8)              r"""
21632: (8)              Class-aware wrapper for `::fz_quadto()`.
21633: (12)                 Append a 'quadto' command to an open path. (For a
21634: (12)                 quadratic bezier).
21635: (12)                 path: The path to modify.
21636: (12)                 x0, y0: The control coordinates for the quadratic curve.
21637: (12)                 x1, y1: The end coordinates for the quadratic curve.
21638: (12)                 Throws exceptions on failure to allocate, or attempting to
21639: (12)                 modify a packed path.
21640: (8)              """
21641: (8)              return _mupdf.FzPath_fz_quadto(self, x0, y0, x1, y1)
21642: (4)          def fz_rectto(self, x0, y0, x1, y1):
21643: (8)              r"""
21644: (8)              Class-aware wrapper for `::fz_rectto()`.
21645: (12)                 Append a 'rectto' command to an open path.
21646: (12)                 The rectangle is equivalent to:
21647: (16)                     moveto x0 y0
21648: (16)                     lineto x1 y0
21649: (16)                     lineto x1 y1
21650: (16)                     lineto x0 y1
21651: (16)                     closepath
21652: (12)                 path: The path to modify.
21653: (12)                 x0, y0: First corner of the rectangle.
21654: (12)                 x1, y1: Second corner of the rectangle.
21655: (12)                 Throws exceptions on failure to allocate, or attempting to
21656: (12)                 modify a packed path.
21657: (8)              """
21658: (8)              return _mupdf.FzPath_fz_rectto(self, x0, y0, x1, y1)
21659: (4)          def fz_transform_path(self, transform):
21660: (8)              r"""
21661: (8)              Class-aware wrapper for `::fz_transform_path()`.
21662: (12)                 Transform a path by a given
21663: (12)                 matrix.
21664: (12)                 path: The path to modify (must not be a packed path).
21665: (12)                 transform: The transform to apply.
21666: (12)                 Throws exceptions if the path is packed, or on failure
21667: (12)                 to allocate.
21668: (8)              """
```

```
21669: (8)                  return _mupdf.FzPath_fz_transform_path(self, transform)
21670: (4)              def fz_trim_path(self):
21671: (8)                  r"""
21672: (8)                  Class-aware wrapper for `::fz_trim_path()`.
21673: (12)                     Minimise the internal storage used by a path.
21674: (12)                     As paths are constructed, the internal buffers
21675: (12)                     grow. To avoid repeated reallocations they
21676: (12)                     grow with some spare space. Once a path has
21677: (12)                     been fully constructed, this call allows the
21678: (12)                     excess space to be trimmed.
21679: (8)                  """
21680: (8)                  return _mupdf.FzPath_fz_trim_path(self)
21681: (4)              def fz_walk_path(self, walker, arg):
21682: (8)                  r"""
21683: (8)                  Class-aware wrapper for `::fz_walk_path()`.
21684: (12)                     Walk the segments of a path, calling the
21685: (12)                     appropriate callback function from a given set for each
21686: (12)                     segment of the path.
21687: (12)                     path: The path to walk.
21688: (12)                     walker: The set of callback functions to use. The first
21689: (12)                     4 callback pointers in the set must be non-NULL. The
21690: (12)                     subsequent ones can either be supplied, or can be left
21691: (12)                     as NULL, in which case the top 4 functions will be
21692: (12)                     called as appropriate to simulate them.
21693: (12)                     arg: An opaque argument passed in to each callback.
21694: (12)                     Exceptions will only be thrown if the underlying callback
21695: (12)                     functions throw them.
21696: (8)                  """
21697: (8)                  return _mupdf.FzPath_fz_walk_path(self, walker, arg)
21698: (4)              def __init__(self, *args):
21699: (8)                  r"""
21700: (8)                  *Overload 1:*
21701: (9)                   == Constructors.  Constructor using `fz_new_path()`.
21702: (16)                         Create a new (empty) path structure.
21703: (8)                  |
21704: (8)                  *Overload 2:*
21705: (9)                   Copy constructor using `fz_keep_path()`.
21706: (8)                  |
21707: (8)                  *Overload 3:*
21708: (9)                   Constructor using raw copy of pre-existing `::fz_path`.
21709: (8)                  """
21710: (8)                  _mupdf.FzPath_swiginit(self, _mupdf.new_FzPath(*args))
21711: (4)              __swig_destroy__ = _mupdf.delete_FzPath
21712: (4)              def m_internal_value(self):
21713: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
21714: (8)                  return _mupdf.FzPath_m_internal_value(self)
21715: (4)              m_internal = property(_mupdf.FzPath_m_internal_get,
_mupdf.FzPath_m_internal_set, doc=r""" Pointer to wrapped data.""")
21716: (4)              s_num_instances = property(_mupdf.FzPath_s_num_instances_get,
_mupdf.FzPath_s_num_instances_set)
21717: (0)          _mupdf.FzPath_swigregister(FzPath)
21718: (0)          class FzPathWalker(object):
21719: (4)              r""" Wrapper class for struct `fz_path_walker`. Not copyable or
assignable."""
21720: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21721: (4)              __repr__ = _swig_repr
21722: (4)              def __init__(self, internal=None):
21723: (8)                  r""" Constructor using raw copy of pre-existing `::fz_path_walker`."""
21724: (8)                  _mupdf.FzPathWalker_swiginit(self, _mupdf.new_FzPathWalker(internal))
21725: (4)              __swig_destroy__ = _mupdf.delete_FzPathWalker
21726: (4)              def m_internal_value(self):
21727: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
21728: (8)                  return _mupdf.FzPathWalker_m_internal_value(self)
21729: (4)              m_internal = property(_mupdf.FzPathWalker_m_internal_get,
_mupdf.FzPathWalker_m_internal_set, doc=r""" Pointer to wrapped data.""")
21730: (4)              s_num_instances = property(_mupdf.FzPathWalker_s_num_instances_get,
```

```
_mupdf.FzPathWalker_s_num_instances_set)
21731: (0)          _mupdf.FzPathWalker_swigregister(FzPathWalker)
21732: (0)          class FzPathWalker2(FzPathWalker):
21733: (4)              r""" Wrapper class for struct fz_path_walker with virtual fns for each
fnptr; this is for use as a SWIG Director class."""
21734: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21735: (4)              __repr__ = _swig_repr
21736: (4)              def __init__(self):
21737: (8)                  r""" == Constructor."""
21738: (8)                  if self.__class__ == FzPathWalker2:
21739: (12)                     _self = None
21740: (8)                  else:
21741: (12)                     _self = self
21742: (8)                  _mupdf.FzPathWalker2_swiginit(self, _mupdf.new_FzPathWalker2(_self, ))
21743: (4)              __swig_destroy__ = _mupdf.delete_FzPathWalker2
21744: (4)              def use_virtual_moveto(self, use=True):
21745: (8)                  r"""
21746: (8)                  These methods set the function pointers in *m_internal
21747: (8)                  to point to internal callbacks that call our virtual methods.
21748: (8)                  """
21749: (8)                  return _mupdf.FzPathWalker2_use_virtual_moveto(self, use)
21750: (4)              def use_virtual_lineto(self, use=True):
21751: (8)                  return _mupdf.FzPathWalker2_use_virtual_lineto(self, use)
21752: (4)              def use_virtual_curveto(self, use=True):
21753: (8)                  return _mupdf.FzPathWalker2_use_virtual_curveto(self, use)
21754: (4)              def use_virtual_closepath(self, use=True):
21755: (8)                  return _mupdf.FzPathWalker2_use_virtual_closepath(self, use)
21756: (4)              def use_virtual_quadto(self, use=True):
21757: (8)                  return _mupdf.FzPathWalker2_use_virtual_quadto(self, use)
21758: (4)              def use_virtual_curvetov(self, use=True):
21759: (8)                  return _mupdf.FzPathWalker2_use_virtual_curvetov(self, use)
21760: (4)              def use_virtual_curvetoy(self, use=True):
21761: (8)                  return _mupdf.FzPathWalker2_use_virtual_curvetoy(self, use)
21762: (4)              def use_virtual_rectto(self, use=True):
21763: (8)                  return _mupdf.FzPathWalker2_use_virtual_rectto(self, use)
21764: (4)              def moveto(self, arg_0, arg_2, arg_3):
21765: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
21766: (8)                  return _mupdf.FzPathWalker2_moveto(self, arg_0, arg_2, arg_3)
21767: (4)              def lineto(self, arg_0, arg_2, arg_3):
21768: (8)                  return _mupdf.FzPathWalker2_lineto(self, arg_0, arg_2, arg_3)
21769: (4)              def curveto(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
21770: (8)                  return _mupdf.FzPathWalker2_curveto(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
21771: (4)              def closepath(self, arg_0):
21772: (8)                  return _mupdf.FzPathWalker2_closepath(self, arg_0)
21773: (4)              def quadto(self, arg_0, arg_2, arg_3, arg_4, arg_5):
21774: (8)                  return _mupdf.FzPathWalker2_quadto(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
21775: (4)              def curvetov(self, arg_0, arg_2, arg_3, arg_4, arg_5):
21776: (8)                  return _mupdf.FzPathWalker2_curvetov(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
21777: (4)              def curvetoy(self, arg_0, arg_2, arg_3, arg_4, arg_5):
21778: (8)                  return _mupdf.FzPathWalker2_curvetoy(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
21779: (4)              def rectto(self, arg_0, arg_2, arg_3, arg_4, arg_5):
21780: (8)                  return _mupdf.FzPathWalker2_rectto(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
21781: (4)              def __disown__(self):
21782: (8)                  self.this.disown()
21783: (8)                  _mupdf.disown_FzPathWalker2(self)
21784: (8)                  return weakref.proxy(self)
21785: (0)          _mupdf.FzPathWalker2_swigregister(FzPathWalker2)
21786: (0)          class FzPclOptions(object):
21787: (4)              r"""
21788: (4)              Wrapper class for struct `fz_pcl_options`. Not copyable or assignable.
21789: (4)              PCL output
21790: (4)              """
```

```
21791: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21792: (4)              __repr__ = _swig_repr
21793: (4)              def fz_pcl_preset(self, preset):
21794: (8)                  r"""
21795: (8)                  Class-aware wrapper for `::fz_pcl_preset()`.
21796: (12)                     Initialize PCL option struct for a given preset.
21797: (12)                     Currently defined presets include:
21798: (16)                         generic    Generic PCL printer
21799: (16)                         ljet4    HP DeskJet
21800: (16)                         dj500    HP DeskJet 500
21801: (16)                         fs600    Kyocera FS-600
21802: (16)                         lj    HP LaserJet, HP LaserJet Plus
21803: (16)                         lj2    HP LaserJet IIp, HP LaserJet IId
21804: (16)                         lj3    HP LaserJet III
21805: (16)                         lj3d    HP LaserJet IIId
21806: (16)                         lj4    HP LaserJet 4
21807: (16)                         lj4pl    HP LaserJet 4 PL
21808: (16)                         lj4d    HP LaserJet 4d
21809: (16)                         lp2563b    HP 2563B line printer
21810: (16)                         oce9050    Oce 9050 Line printer
21811: (8)                  """
21812: (8)                  return _mupdf.FzPclOptions_fz_pcl_preset(self, preset)
21813: (4)              def __init__(self, *args):
21814: (8)                  r"""
21815: (8)                  *Overload 1:*
21816: (9)                   == Constructors.  Constructor using `fz_parse_pcl_options()`.
21817: (16)                         Parse PCL options.
21818: (16)                         Currently defined options and values are as follows:
21819: (20)                             preset=X    Either "generic" or one of the presets as for
fz_pcl_preset.
21820: (20)                             spacing=0    No vertical spacing capability
21821: (20)                             spacing=1    PCL 3 spacing (<ESC>*p+<n>Y)
21822: (20)                             spacing=2    PCL 4 spacing (<ESC>*b<n>Y)
21823: (20)                             spacing=3    PCL 5 spacing (<ESC>*b<n>Y and clear seed
row)
21824: (20)                             mode2        Disable/Enable mode 2 graphics compression
21825: (20)                             mode3        Disable/Enable mode 3 graphics compression
21826: (20)                             eog_reset    End of graphics (<ESC>*rB) resets all
parameters
21827: (20)                             has_duplex    Duplex supported (<ESC>&l<duplex>S)
21828: (20)                             has_papersize    Papersize setting supported
(<ESC>&l<sizecode>A)
21829: (20)                             has_copies    Number of copies supported
(<ESC>&l<copies>X)
21830: (20)                             is_ljet4pjl    Disable/Enable HP 4PJL model-specific
output
21831: (20)                             is_oce9050    Disable/Enable Oce 9050 model-specific
output
21832: (8)                  |
21833: (8)                  *Overload 2:*
21834: (9)                   Default constructor, sets `m_internal` to null.
21835: (8)                  |
21836: (8)                  *Overload 3:*
21837: (9)                   Constructor using raw copy of pre-existing `::fz_pcl_options`.
21838: (8)                  """
21839: (8)                  _mupdf.FzPclOptions_swiginit(self, _mupdf.new_FzPclOptions(*args))
21840: (4)              __swig_destroy__ = _mupdf.delete_FzPclOptions
21841: (4)              def m_internal_value(self):
21842: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
21843: (8)                  return _mupdf.FzPclOptions_m_internal_value(self)
21844: (4)              m_internal = property(_mupdf.FzPclOptions_m_internal_get,
_mupdf.FzPclOptions_m_internal_set, doc=r""" Pointer to wrapped data.""")
21845: (4)              s_num_instances = property(_mupdf.FzPclOptions_s_num_instances_get,
_mupdf.FzPclOptions_s_num_instances_set)
21846: (0)          _mupdf.FzPclOptions_swigregister(FzPclOptions)
21847: (0)          class FzPclmOptions(object):
21848: (4)              r"""
```

```
21849: (4)              Wrapper class for struct `fz_pclm_options`. Not copyable or assignable.
21850: (4)              PCLm output
21851: (4)              """
21852: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21853: (4)              __repr__ = _swig_repr
21854: (4)              def __init__(self, *args):
21855: (8)                  r"""
21856: (8)                  *Overload 1:*
21857: (9)                   == Constructors.  Constructor using `fz_parse_pclm_options()`.
21858: (16)                         Parse PCLm options.
21859: (16)                         Currently defined options and values are as follows:
21860: (20)                             compression=none: No compression
21861: (20)                             compression=flate: Flate compression
21862: (20)                             strip-height=n: Strip height (default 16)
21863: (8)                  |
21864: (8)                  *Overload 2:*
21865: (9)                   Construct using fz_parse_pclm_options().
21866: (8)                  |
21867: (8)                  *Overload 3:*
21868: (9)                   Default constructor, sets `m_internal` to null.
21869: (8)                  |
21870: (8)                  *Overload 4:*
21871: (9)                   Constructor using raw copy of pre-existing `::fz_pclm_options`.
21872: (8)                  """
21873: (8)                  _mupdf.FzPclmOptions_swiginit(self, _mupdf.new_FzPclmOptions(*args))
21874: (4)              __swig_destroy__ = _mupdf.delete_FzPclmOptions
21875: (4)              def m_internal_value(self):
21876: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
21877: (8)                  return _mupdf.FzPclmOptions_m_internal_value(self)
21878: (4)              m_internal = property(_mupdf.FzPclmOptions_m_internal_get,
_mupdf.FzPclmOptions_m_internal_set, doc=r""" Pointer to wrapped data.""")
21879: (4)              s_num_instances = property(_mupdf.FzPclmOptions_s_num_instances_get,
_mupdf.FzPclmOptions_s_num_instances_set)
21880: (0)          _mupdf.FzPclmOptions_swigregister(FzPclmOptions)
21881: (0)          class FzPdfocrOptions(object):
21882: (4)              r"""
21883: (4)              Wrapper class for struct `fz_pdfocr_options`.
21884: (4)              PDFOCR output
21885: (4)              """
21886: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21887: (4)              __repr__ = _swig_repr
21888: (4)              def fz_parse_pdfocr_options(self, args):
21889: (8)                  r"""
21890: (8)                  Class-aware wrapper for `::fz_parse_pdfocr_options()`.
21891: (12)                     Parse PDFOCR options.
21892: (12)                     Currently defined options and values are as follows:
21893: (16)                         compression=none: No compression
21894: (16)                         compression=flate: Flate compression
21895: (16)                         strip-height=n: Strip height (default 16)
21896: (16)                         ocr-language=<lang>: OCR Language (default eng)
21897: (16)                         ocr-datadir=<datadir>: OCR data path (default rely on
TESSDATA_PREFIX)
21898: (8)                  """
21899: (8)                  return _mupdf.FzPdfocrOptions_fz_parse_pdfocr_options(self, args)
21900: (4)              def language_set2(self, language):
21901: (8)                  r""" Copies <language> into this->language, truncating if
necessary."""
21902: (8)                  return _mupdf.FzPdfocrOptions_language_set2(self, language)
21903: (4)              def datadir_set2(self, datadir):
21904: (8)                  r""" Copies <datadir> into this->datadir, truncating if necessary."""
21905: (8)                  return _mupdf.FzPdfocrOptions_datadir_set2(self, datadir)
21906: (4)              def __init__(self, *args):
21907: (8)                  r"""
21908: (8)                  *Overload 1:*
21909: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
```

```
21910: (8)                  |
21911: (8)                  *Overload 2:*
21912: (8)                  Constructor using raw copy of pre-existing `::fz_pdfocr_options`.
21913: (8)                  |
21914: (8)                  *Overload 3:*
21915: (8)                  Constructor using raw copy of pre-existing `::fz_pdfocr_options`.
21916: (8)                  """
21917: (8)                  _mupdf.FzPdfocrOptions_swiginit(self,
_mupdf.new_FzPdfocrOptions(*args))
21918: (4)              def internal(self, *args):
21919: (8)                  r"""
21920: (8)                  *Overload 1:*
21921: (8)                  Access as underlying struct.
21922: (8)                  |
21923: (8)                  *Overload 2:*
21924: (8)                  Access as underlying struct.
21925: (8)                  """
21926: (8)                  return _mupdf.FzPdfocrOptions_internal(self, *args)
21927: (4)              __swig_destroy__ = _mupdf.delete_FzPdfocrOptions
21928: (4)              compress = property(_mupdf.FzPdfocrOptions_compress_get,
_mupdf.FzPdfocrOptions_compress_set)
21929: (4)              strip_height = property(_mupdf.FzPdfocrOptions_strip_height_get,
_mupdf.FzPdfocrOptions_strip_height_set)
21930: (4)              language = property(_mupdf.FzPdfocrOptions_language_get,
_mupdf.FzPdfocrOptions_language_set)
21931: (4)              datadir = property(_mupdf.FzPdfocrOptions_datadir_get,
_mupdf.FzPdfocrOptions_datadir_set)
21932: (4)              page_count = property(_mupdf.FzPdfocrOptions_page_count_get,
_mupdf.FzPdfocrOptions_page_count_set)
21933: (4)              s_num_instances = property(_mupdf.FzPdfocrOptions_s_num_instances_get,
_mupdf.FzPdfocrOptions_s_num_instances_set)
21934: (4)              def to_string(self):
21935: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
21936: (8)                  return _mupdf.FzPdfocrOptions_to_string(self)
21937: (4)              def __eq__(self, rhs):
21938: (8)                  r""" Comparison method."""
21939: (8)                  return _mupdf.FzPdfocrOptions___eq__(self, rhs)
21940: (4)              def __ne__(self, rhs):
21941: (8)                  r""" Comparison method."""
21942: (8)                  return _mupdf.FzPdfocrOptions___ne__(self, rhs)
21943: (0)          _mupdf.FzPdfocrOptions_swigregister(FzPdfocrOptions)
21944: (0)          class FzPixmap(object):
21945: (4)              r"""
21946: (4)              Wrapper class for struct `fz_pixmap`.
21947: (4)              Pixmaps represent a set of pixels for a 2 dimensional region of
21948: (4)              a plane. Each pixel has n components per pixel. The components
21949: (4)              are in the order process-components, spot-colors, alpha, where
21950: (4)              there can be 0 of any of those types. The data is in
21951: (4)              premultiplied alpha when rendering, but non-premultiplied for
21952: (4)              colorspace conversions and rescaling.
21953: (4)              """
21954: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
21955: (4)              __repr__ = _swig_repr
21956: (4)              @staticmethod
21957: (4)              def pdf_new_pixmap_from_page_with_usage(page, ctm, cs, alpha, usage, box):
21958: (8)                  r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_with_usage()`."""
21959: (8)                  return _mupdf.FzPixmap_pdf_new_pixmap_from_page_with_usage(page, ctm,
cs, alpha, usage, box)
21960: (4)              @staticmethod
21961: (4)              def pdf_new_pixmap_from_page_with_separations_and_usage(page, ctm, cs,
seps, alpha, usage, box):
21962: (8)                  r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_with_separations_and_usage()`."""
21963: (8)                  return
_mupdf.FzPixmap_pdf_new_pixmap_from_page_with_separations_and_usage(page, ctm, cs, seps, alpha,
usage, box)
```

```
(4)             @staticmethod
(4)             def fz_new_pixmap_from_page_contents(page, ctm, cs, alpha):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_pixmap_from_page_contents()`.
(12)                    Render the page contents without annotations.
(12)                    Ownership of the pixmap is returned to the caller.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_new_pixmap_from_page_contents(page, ctm, cs, alpha)
(4)             @staticmethod
(4)             def fz_new_pixmap_from_page_contents_with_separations(page, ctm, cs, seps, alpha):
(8)                 r""" Class-aware wrapper for `::fz_new_pixmap_from_page_contents_with_separations()`."""
(8)                 return _mupdf.FzPixmap_fz_new_pixmap_from_page_contents_with_separations(page, ctm, cs, seps, alpha)
(4)             def fz_alpha_from_gray(self):
(8)                 r""" Class-aware wrapper for `::fz_alpha_from_gray()`."""
(8)                 return _mupdf.FzPixmap_fz_alpha_from_gray(self)
(4)             def fz_clear_pixmap(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clear_pixmap()`.
(12)                    Sets all components (including alpha) of
(12)                    all pixels in a pixmap to 0.
(12)                    pix: The pixmap to clear.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_clear_pixmap(self)
(4)             def fz_clear_pixmap_rect_with_value(self, value, r):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clear_pixmap_rect_with_value()`.
(12)                    Clears a subrect of a pixmap with the given value.
(12)                    pix: The pixmap to clear.
(12)                    value: Values in the range 0 to 255 are valid. Each component
(12)                    sample for each pixel in the pixmap will be set to this value,
(12)                    while alpha will always be set to 255 (non-transparent).
(12)                    r: the rectangle.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_clear_pixmap_rect_with_value(self, value, r)
(4)             def fz_clear_pixmap_with_value(self, value):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clear_pixmap_with_value()`.
(12)                    Clears a pixmap with the given value.
(12)                    pix: The pixmap to clear.
(12)                    value: Values in the range 0 to 255 are valid. Each component
(12)                    sample for each pixel in the pixmap will be set to this value,
(12)                    while alpha will always be set to 255 (non-transparent).
(12)                    This function is horrible, and should be removed from the
(12)                    API and replaced with a less magic one.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_clear_pixmap_with_value(self, value)
(4)             def fz_clone_pixmap(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_clone_pixmap()`.
(12)                    Clone a pixmap, copying the pixels and associated data to new
(12)                    storage.
(12)                    The reference count of 'old' is unchanged.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_clone_pixmap(self)
(4)             def fz_clone_pixmap_area_with_different_seps(self, bbox, dcs, seps, color_params, default_cs):
(8)                 r""" Class-aware wrapper for `::fz_clone_pixmap_area_with_different_seps()`."""
(8)                 return _mupdf.FzPixmap_fz_clone_pixmap_area_with_different_seps(self, bbox, dcs, seps, color_params, default_cs)
(4)             def fz_convert_indexed_pixmap_to_base(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_convert_indexed_pixmap_to_base()`.
(12)                    Convert pixmap from indexed to base colorspace.
(12)                    This creates a new bitmap containing the converted pixmap data.
```

```
22026: (8)                    """
22027: (8)                    return _mupdf.FzPixmap_fz_convert_indexed_pixmap_to_base(self)
22028: (4)                def fz_convert_pixmap(self, cs_des, prf, default_cs, color_params,
keep_alpha):
22029: (8)                    r"""
22030: (8)                    Class-aware wrapper for `::fz_convert_pixmap()`.
22031: (12)                       Convert an existing pixmap to a desired
22032: (12)                       colorspace. Other properties of the pixmap, such as resolution
22033: (12)                       and position are copied to the converted pixmap.
22034: (12)                       pix: The pixmap to convert.
22035: (12)                       default_cs: If NULL pix->colorspace is used. It is possible that
22036: (12)                       the data may need to be interpreted as one of the color spaces
22037: (12)                       in default_cs.
22038: (12)                       cs_des: Desired colorspace, may be NULL to denote alpha-only.
22039: (12)                       prf: Proofing color space through which we need to convert.
22040: (12)                       color_params: Parameters that may be used in conversion (e.g.
22041: (12)                       ri).
22042: (12)                       keep_alpha: If 0 any alpha component is removed, otherwise
22043: (12)                       alpha is kept if present in the pixmap.
22044: (8)                    """
22045: (8)                    return _mupdf.FzPixmap_fz_convert_pixmap(self, cs_des, prf,
default_cs, color_params, keep_alpha)
22046: (4)                def fz_convert_separation_pixmap_to_base(self):
22047: (8)                    r"""
22048: (8)                    Class-aware wrapper for `::fz_convert_separation_pixmap_to_base()`.
22049: (12)                       Convert pixmap from DeviceN/Separation to base colorspace.
22050: (12)                       This creates a new bitmap containing the converted pixmap data.
22051: (8)                    """
22052: (8)                    return _mupdf.FzPixmap_fz_convert_separation_pixmap_to_base(self)
22053: (4)                def fz_copy_pixmap_rect(self, src, r, default_cs):
22054: (8)                    r""" Class-aware wrapper for `::fz_copy_pixmap_rect()`."""
22055: (8)                    return _mupdf.FzPixmap_fz_copy_pixmap_rect(self, src, r, default_cs)
22056: (4)                def fz_decode_tile(self, decode):
22057: (8)                    r""" Class-aware wrapper for `::fz_decode_tile()`."""
22058: (8)                    return _mupdf.FzPixmap_fz_decode_tile(self, decode)
22059: (4)                def fz_fill_pixmap_with_color(self, colorspace, color, color_params):
22060: (8)                    r"""
22061: (8)                    Class-aware wrapper for `::fz_fill_pixmap_with_color()`.
22062: (8)                    This method has out-params. Python/C# wrappers look like:
22063: (12)                       `fz_fill_pixmap_with_color(::fz_colorspace *colorspace,
::fz_color_params color_params)` => float color
22064: (12)                       Fill pixmap with solid color.
22065: (8)                    """
22066: (8)                    return _mupdf.FzPixmap_fz_fill_pixmap_with_color(self, colorspace,
color, color_params)
22067: (4)                def fz_gamma_pixmap(self, gamma):
22068: (8)                    r"""
22069: (8)                    Class-aware wrapper for `::fz_gamma_pixmap()`.
22070: (12)                       Apply gamma correction to a pixmap. All components
22071: (12)                       of all pixels are modified (except alpha, which is unchanged).
22072: (12)                       gamma: The gamma value to apply; 1.0 for no change.
22073: (8)                    """
22074: (8)                    return _mupdf.FzPixmap_fz_gamma_pixmap(self, gamma)
22075: (4)                def fz_generate_transition(self, opix, npix, time, trans):
22076: (8)                    r"""
22077: (8)                    Class-aware wrapper for `::fz_generate_transition()`.
22078: (12)                       Generate a frame of a transition.
22079: (12)                       tpix: Target pixmap
22080: (12)                       opix: Old pixmap
22081: (12)                       npix: New pixmap
22082: (12)                       time: Position within the transition (0 to 256)
22083: (12)                       trans: Transition details
22084: (12)                       Returns 1 if successfully generated a frame.
22085: (12)                       Note: Pixmaps must include alpha.
22086: (8)                    """
22087: (8)                    return _mupdf.FzPixmap_fz_generate_transition(self, opix, npix, time,
trans)
22088: (4)                def fz_invert_pixmap(self):
22089: (8)                    r"""
```

```
(8)                 Class-aware wrapper for `::fz_invert_pixmap()`.
(12)                    Invert all the pixels in a pixmap. All components (process and
(12)                    spots) of all pixels are inverted (except alpha, which is
(12)                    unchanged).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_invert_pixmap(self)
(4)             def fz_invert_pixmap_alpha(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_invert_pixmap_alpha()`.
(12)                    Invert the alpha fo all the pixels in a pixmap.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_invert_pixmap_alpha(self)
(4)             def fz_invert_pixmap_luminance(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_invert_pixmap_luminance()`.
(12)                    Transform the pixels in a pixmap so that luminance of each
(12)                    pixel is inverted, and the chrominance remains unchanged (as
(12)                    much as accuracy allows).
(12)                    All components of all pixels are inverted (except alpha, which
(12)                    is unchanged). Only supports Grey and RGB bitmaps.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_invert_pixmap_luminance(self)
(4)             def fz_invert_pixmap_raw(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_invert_pixmap_raw()`.
(12)                    Invert all the pixels in a non-premultiplied pixmap in a
(12)                    very naive manner.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_invert_pixmap_raw(self)
(4)             def fz_invert_pixmap_rect(self, rect):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_invert_pixmap_rect()`.
(12)                    Invert all the pixels in a given rectangle of a (premultiplied)
(12)                    pixmap. All components of all pixels in the rectangle are
(12)                    inverted (except alpha, which is unchanged).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_invert_pixmap_rect(self, rect)
(4)             def fz_is_pixmap_monochrome(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_pixmap_monochrome()`.
(12)                    Check if the pixmap is a 1-channel image containing samples with
(12)                    only values 0 and 255
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_is_pixmap_monochrome(self)
(4)             def fz_md5_pixmap(self, digest):
(8)                 r""" Class-aware wrapper for `::fz_md5_pixmap()`."""
(8)                 return _mupdf.FzPixmap_fz_md5_pixmap(self, digest)
(4)             def fz_md5_pixmap2(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_md5_pixmap2()`.
(8)                 C++ alternative to `fz_md5_pixmap()` that returns the digest by value.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_md5_pixmap2(self)
(4)             def fz_new_bitmap_from_pixmap(self, ht):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_bitmap_from_pixmap()`.
(12)                    Make a bitmap from a pixmap and a halftone.
(12)                    pix: The pixmap to generate from. Currently must be a single
(12)                    color component with no alpha.
(12)                    ht: The halftone to use. NULL implies the default halftone.
(12)                    Returns the resultant bitmap. Throws exceptions in the case of
(12)                    failure to allocate.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_new_bitmap_from_pixmap(self, ht)
(4)             def fz_new_bitmap_from_pixmap_band(self, ht, band_start):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_bitmap_from_pixmap_band()`.
(12)                    Make a bitmap from a pixmap and a
(12)                    halftone, allowing for the position of the pixmap within an
```

```
22159: (12)                     overall banded rendering.
22160: (12)                     pix: The pixmap to generate from. Currently must be a single
22161: (12)                     color component with no alpha.
22162: (12)                     ht: The halftone to use. NULL implies the default halftone.
22163: (12)                     band_start: Vertical offset within the overall banded rendering
22164: (12)                     (in pixels)
22165: (12)                     Returns the resultant bitmap. Throws exceptions in the case of
22166: (12)                     failure to allocate.
22167: (8)                  """
22168: (8)                  return _mupdf.FzPixmap_fz_new_bitmap_from_pixmap_band(self, ht,
band_start)
22169: (4)              def fz_new_buffer_from_pixmap_as_jpeg(self, color_params, quality,
invert_cmyk):
22170: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_jpeg()`."""
22171: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_jpeg(self,
color_params, quality, invert_cmyk)
22172: (4)              def fz_new_buffer_from_pixmap_as_jpx(self, color_params, quality):
22173: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_jpx()`."""
22174: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_jpx(self,
color_params, quality)
22175: (4)              def fz_new_buffer_from_pixmap_as_pam(self, color_params):
22176: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_pam()`."""
22177: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_pam(self,
color_params)
22178: (4)              def fz_new_buffer_from_pixmap_as_png(self, color_params):
22179: (8)                  r"""
22180: (8)                  Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_png()`.
22181: (12)                     Reencode a given pixmap as a PNG into a buffer.
22182: (12)                     Ownership of the buffer is returned.
22183: (8)                  """
22184: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_png(self,
color_params)
22185: (4)              def fz_new_buffer_from_pixmap_as_pnm(self, color_params):
22186: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_pnm()`."""
22187: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_pnm(self,
color_params)
22188: (4)              def fz_new_buffer_from_pixmap_as_psd(self, color_params):
22189: (8)                  r""" Class-aware wrapper for
`::fz_new_buffer_from_pixmap_as_psd()`."""
22190: (8)                  return _mupdf.FzPixmap_fz_new_buffer_from_pixmap_as_psd(self,
color_params)
22191: (4)              def fz_new_image_from_pixmap(self, mask):
22192: (8)                  r"""
22193: (8)                  Class-aware wrapper for `::fz_new_image_from_pixmap()`.
22194: (12)                     Create an image from the given
22195: (12)                     pixmap.
22196: (12)                     pixmap: The pixmap to base the image upon. A new reference
22197: (12)                     to this is taken.
22198: (12)                     mask: NULL, or another image to use as a mask for this one.
22199: (12)                     A new reference is taken to this image. Supplying a masked
22200: (12)                     image as a mask to another image is illegal!
22201: (8)                  """
22202: (8)                  return _mupdf.FzPixmap_fz_new_image_from_pixmap(self, mask)
22203: (4)              def fz_new_pixmap_from_alpha_channel(self):
22204: (8)                  r""" Class-aware wrapper for
`::fz_new_pixmap_from_alpha_channel()`."""
22205: (8)                  return _mupdf.FzPixmap_fz_new_pixmap_from_alpha_channel(self)
22206: (4)              def fz_new_pixmap_from_color_and_mask(self, mask):
22207: (8)                  r""" Class-aware wrapper for
`::fz_new_pixmap_from_color_and_mask()`."""
22208: (8)                  return _mupdf.FzPixmap_fz_new_pixmap_from_color_and_mask(self, mask)
22209: (4)              def fz_new_pixmap_from_pixmap(self, rect):
22210: (8)                  r"""
22211: (8)                  Class-aware wrapper for `::fz_new_pixmap_from_pixmap()`.
22212: (12)                     Create a new pixmap that represents a subarea of the specified
```

```
(12)                    pixmap. A reference is taken to this pixmap that will be dropped
(12)                    on destruction.
(12)                    The supplied rectangle must be wholly contained within the
(12)                    original pixmap.
(12)                    Returns a pointer to the new pixmap. Throws exception on failure
(12)                    to allocate.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_new_pixmap_from_pixmap(self, rect)
(4)             def fz_pixmap_alpha(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_alpha()`.
(12)                    Return the number of alpha planes in a pixmap.
(12)                    Returns the number of alphas. Does not throw exceptions.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_alpha(self)
(4)             def fz_pixmap_bbox(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_bbox()`.
(12)                    Return the bounding box for a pixmap.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_bbox(self)
(4)             def fz_pixmap_colorants(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_colorants()`.
(12)                    Return the number of colorants in a pixmap.
(12)                    Returns the number of colorants (components, less any spots and
(12)                    alpha).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_colorants(self)
(4)             def fz_pixmap_colorspace(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_colorspace()`.
(12)                    Return the colorspace of a pixmap
(12)                    Returns colorspace.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_colorspace(self)
(4)             def fz_pixmap_components(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_components()`.
(12)                    Return the number of components in a pixmap.
(12)                    Returns the number of components (including spots and alpha).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_components(self)
(4)             def fz_pixmap_height(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_height()`.
(12)                    Return the height of the pixmap in pixels.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_height(self)
(4)             def fz_pixmap_samples(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_samples()`.
(12)                    Returns a pointer to the pixel data of a pixmap.
(12)                    Returns the pointer.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_samples(self)
(4)             def fz_pixmap_samples_int(self):
(8)                 r""" Class-aware wrapper for `::fz_pixmap_samples_int()`."""
(8)                 return _mupdf.FzPixmap_fz_pixmap_samples_int(self)
(4)             def fz_pixmap_size(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_size()`.
(12)                    Return sizeof fz_pixmap plus size of data, in bytes.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_pixmap_size(self)
(4)             def fz_pixmap_spots(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_pixmap_spots()`.
(12)                    Return the number of spots in a pixmap.
```

```
22282: (12)                     Returns the number of spots (components, less colorants and
22283: (12)                     alpha). Does not throw exceptions.
22284: (8)                  """
22285: (8)                  return _mupdf.FzPixmap_fz_pixmap_spots(self)
22286: (4)              def fz_pixmap_stride(self):
22287: (8)                  r"""
22288: (8)                  Class-aware wrapper for `::fz_pixmap_stride()`.
22289: (12)                     Return the number of bytes in a row in the pixmap.
22290: (8)                  """
22291: (8)                  return _mupdf.FzPixmap_fz_pixmap_stride(self)
22292: (4)              def fz_pixmap_width(self):
22293: (8)                  r"""
22294: (8)                  Class-aware wrapper for `::fz_pixmap_width()`.
22295: (12)                     Return the width of the pixmap in pixels.
22296: (8)                  """
22297: (8)                  return _mupdf.FzPixmap_fz_pixmap_width(self)
22298: (4)              def fz_pixmap_x(self):
22299: (8)                  r"""
22300: (8)                  Class-aware wrapper for `::fz_pixmap_x()`.
22301: (12)                     Return the x value of the pixmap in pixels.
22302: (8)                  """
22303: (8)                  return _mupdf.FzPixmap_fz_pixmap_x(self)
22304: (4)              def fz_pixmap_y(self):
22305: (8)                  r"""
22306: (8)                  Class-aware wrapper for `::fz_pixmap_y()`.
22307: (12)                     Return the y value of the pixmap in pixels.
22308: (8)                  """
22309: (8)                  return _mupdf.FzPixmap_fz_pixmap_y(self)
22310: (4)              def fz_samples_get(self, offset):
22311: (8)                  r"""
22312: (8)                  Class-aware wrapper for `::fz_samples_get()`.
22313: (8)                  Provides simple (but slow) access to pixmap data from Python and C#.
22314: (8)                  """
22315: (8)                  return _mupdf.FzPixmap_fz_samples_get(self, offset)
22316: (4)              def fz_samples_set(self, offset, value):
22317: (8)                  r"""
22318: (8)                  Class-aware wrapper for `::fz_samples_set()`.
22319: (8)                  Provides simple (but slow) write access to pixmap data from Python and
22320: (8)                  C#.
22321: (8)                  """
22322: (8)                  return _mupdf.FzPixmap_fz_samples_set(self, offset, value)
22323: (4)              def fz_save_pixmap_as_jpeg(self, filename, quality):
22324: (8)                  r"""
22325: (8)                  Class-aware wrapper for `::fz_save_pixmap_as_jpeg()`.
22326: (12)                     Save a pixmap as a JPEG.
22327: (8)                  """
22328: (8)                  return _mupdf.FzPixmap_fz_save_pixmap_as_jpeg(self, filename, quality)
22329: (4)              def fz_save_pixmap_as_jpx(self, filename, q):
22330: (8)                  r"""
22331: (8)                  Class-aware wrapper for `::fz_save_pixmap_as_jpx()`.
22332: (12)                     Save pixmap data as JP2K with no subsampling.
22333: (12)                     quality = 100 = lossless
22334: (12)                     otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
22335: (8)                  """
22336: (8)                  return _mupdf.FzPixmap_fz_save_pixmap_as_jpx(self, filename, q)
22337: (4)              def fz_save_pixmap_as_pam(self, filename):
22338: (8)                  r"""
22339: (8)                  Class-aware wrapper for `::fz_save_pixmap_as_pam()`.
22340: (12)                     Save a pixmap as a pnm (greyscale, rgb or cmyk, with or without
22341: (12)                     alpha).
22342: (8)                  """
22343: (8)                  return _mupdf.FzPixmap_fz_save_pixmap_as_pam(self, filename)
22344: (4)              def fz_save_pixmap_as_pbm(self, filename):
22345: (8)                  r"""
22346: (8)                  Class-aware wrapper for `::fz_save_pixmap_as_pbm()`.
22347: (12)                     Save a pixmap as a pbm. (Performing halftoning).
22348: (8)                  """
22349: (8)                  return _mupdf.FzPixmap_fz_save_pixmap_as_pbm(self, filename)
```

```
(4)             def fz_save_pixmap_as_pcl(self, filename, append, pcl):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pcl()`.
(12)                    Save an (RGB) pixmap as color PCL.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pcl(self, filename, append, pcl)
(4)             def fz_save_pixmap_as_pclm(self, filename, append, options):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pclm()`.
(12)                    Save a (Greyscale or RGB) pixmap as pclm.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pclm(self, filename, append, options)
(4)             def fz_save_pixmap_as_pdfocr(self, filename, append, options):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pdfocr()`.
(12)                    Save a (Greyscale or RGB) pixmap as pdfocr.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pdfocr(self, filename, append, options)
(4)             def fz_save_pixmap_as_pkm(self, filename):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pkm()`.
(12)                    Save a CMYK pixmap as a pkm. (Performing halftoning).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pkm(self, filename)
(4)             def fz_save_pixmap_as_png(self, filename):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_png()`.
(12)                    Save a (Greyscale or RGB) pixmap as a png.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_png(self, filename)
(4)             def fz_save_pixmap_as_pnm(self, filename):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pnm()`.
(12)                    Save a pixmap as a pnm (greyscale or rgb, no alpha).
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pnm(self, filename)
(4)             def fz_save_pixmap_as_ps(self, filename, append):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_ps()`.
(12)                    Save a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_ps(self, filename, append)
(4)             def fz_save_pixmap_as_psd(self, filename):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_psd()`.
(12)                    Save a pixmap as a PSD file.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_psd(self, filename)
(4)             def fz_save_pixmap_as_pwg(self, filename, append, pwg):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_save_pixmap_as_pwg()`.
(12)                    Save a pixmap as a PWG.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_save_pixmap_as_pwg(self, filename, append, pwg)
(4)             def fz_scale_pixmap(self, x, y, w, h, clip):
(8)                 r""" Class-aware wrapper for `::fz_scale_pixmap()`."""
(8)                 return _mupdf.FzPixmap_fz_scale_pixmap(self, x, y, w, h, clip)
(4)             def fz_set_pixmap_resolution(self, xres, yres):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_set_pixmap_resolution()`.
(12)                    Set the pixels per inch resolution of the pixmap.
(8)                 """
(8)                 return _mupdf.FzPixmap_fz_set_pixmap_resolution(self, xres, yres)
(4)             def fz_subsample_pixmap(self, factor):
(8)                 r""" Class-aware wrapper for `::fz_subsample_pixmap()`."""
```

```
22415: (8)                  return _mupdf.FzPixmap_fz_subsample_pixmap(self, factor)
22416: (4)              def fz_tint_pixmap(self, black, white):
22417: (8)                  r"""
22418: (8)                  Class-aware wrapper for `::fz_tint_pixmap()`.
22419: (12)                     Tint all the pixels in an RGB, BGR, or Gray pixmap.
22420: (12)                     black: Map black to this hexadecimal RGB color.
22421: (12)                     white: Map white to this hexadecimal RGB color.
22422: (8)                  """
22423: (8)                  return _mupdf.FzPixmap_fz_tint_pixmap(self, black, white)
22424: (4)              def __init__(self, *args):
22425: (8)                  r"""
22426: (8)                  *Overload 1:*
22427: (9)                   == Constructors.  Constructor using `fz_new_pixmap()`.
22428: (16)                         Create a new pixmap, with its origin at (0,0)
22429: (16)                         cs: The colorspace to use for the pixmap, or NULL for an alpha
22430: (16)                         plane/mask.
22431: (16)                         w: The width of the pixmap (in pixels)
22432: (16)                         h: The height of the pixmap (in pixels)
22433: (16)                         seps: Details of separations.
22434: (16)                         alpha: 0 for no alpha, 1 for alpha.
22435: (16)                         Returns a pointer to the new pixmap. Throws exception on
failure
22436: (16)                         to allocate.
22437: (8)                  |
22438: (8)                  *Overload 2:*
22439: (9)                   Constructor using `fz_new_pixmap_from_color_and_mask()`.
22440: (8)                  |
22441: (8)                  *Overload 3:*
22442: (9)                   Constructor using `fz_new_pixmap_from_display_list()`.
22443: (16)                         Render the page to a pixmap using the transform and
colorspace.
22444: (16)                         Ownership of the pixmap is returned to the caller.
22445: (8)                  |
22446: (8)                  *Overload 4:*
22447: (9)                   Constructor using
`fz_new_pixmap_from_display_list_with_separations()`.
22448: (16)                         Render the page contents with control over spot colors.
22449: (16)                         Ownership of the pixmap is returned to the caller.
22450: (8)                  |
22451: (8)                  *Overload 5:*
22452: (9)                   Constructor using `fz_new_pixmap_from_page()`.
22453: (8)                  |
22454: (8)                  *Overload 6:*
22455: (9)                   Constructor using `fz_new_pixmap_from_page_number()`.
22456: (8)                  |
22457: (8)                  *Overload 7:*
22458: (9)                   Constructor using
`fz_new_pixmap_from_page_number_with_separations()`.
22459: (8)                  |
22460: (8)                  *Overload 8:*
22461: (9)                   Constructor using `fz_new_pixmap_from_page_with_separations()`.
22462: (8)                  |
22463: (8)                  *Overload 9:*
22464: (9)                   Constructor using `fz_new_pixmap_from_pixmap()`.
22465: (16)                         Create a new pixmap that represents a subarea of the specified
22466: (16)                         pixmap. A reference is taken to this pixmap that will be
dropped
22467: (16)                         on destruction.
22468: (16)                         The supplied rectangle must be wholly contained within the
22469: (16)                         original pixmap.
22470: (16)                         Returns a pointer to the new pixmap. Throws exception on
failure
22471: (16)                         to allocate.
22472: (8)                  |
22473: (8)                  *Overload 10:*
22474: (9)                   Constructor using `fz_new_pixmap_with_bbox()`.
22475: (16)                         Create a pixmap of a given size, location and pixel format.
22476: (16)                         The bounding box specifies the size of the created pixmap and
22477: (16)                         where it will be located. The colorspace determines the number
```

```
22478: (16)                         of components per pixel. Alpha is always present. Pixmaps are
22479: (16)                         reference counted, so drop references using fz_drop_pixmap.
22480: (16)                         colorspace: Colorspace format used for the created pixmap. The
22481: (16)                         pixmap will keep a reference to the colorspace.
22482: (16)                         bbox: Bounding box specifying location/size of created pixmap.
22483: (16)                         seps: Details of separations.
22484: (16)                         alpha: 0 for no alpha, 1 for alpha.
22485: (16)                         Returns a pointer to the new pixmap. Throws exception on
failure
22486: (16)                         to allocate.
22487: (8)                  |
22488: (8)                  *Overload 11:*
22489: (9)                   Constructor using `fz_new_pixmap_with_bbox_and_data()`.
22490: (16)                         Create a pixmap of a given size, location and pixel format,
22491: (16)                         using the supplied data block.
22492: (16)                         The bounding box specifies the size of the created pixmap and
22493: (16)                         where it will be located. The colorspace determines the number
22494: (16)                         of components per pixel. Alpha is always present. Pixmaps are
22495: (16)                         reference counted, so drop references using fz_drop_pixmap.
22496: (16)                         colorspace: Colorspace format used for the created pixmap. The
22497: (16)                         pixmap will keep a reference to the colorspace.
22498: (16)                         rect: Bounding box specifying location/size of created pixmap.
22499: (16)                         seps: Details of separations.
22500: (16)                         alpha: Number of alpha planes (0 or 1).
22501: (16)                         samples: The data block to keep the samples in.
22502: (16)                         Returns a pointer to the new pixmap. Throws exception on
failure
22503: (16)                         to allocate.
22504: (8)                  |
22505: (8)                  *Overload 12:*
22506: (9)                   Constructor using `fz_new_pixmap_with_data()`.
22507: (16)                         Create a new pixmap, with its origin at
22508: (16)                         (0,0) using the supplied data block.
22509: (16)                         cs: The colorspace to use for the pixmap, or NULL for an alpha
22510: (16)                         plane/mask.
22511: (16)                         w: The width of the pixmap (in pixels)
22512: (16)                         h: The height of the pixmap (in pixels)
22513: (16)                         seps: Details of separations.
22514: (16)                         alpha: 0 for no alpha, 1 for alpha.
22515: (16)                         stride: The byte offset from the pixel data in a row to the
22516: (16)                         pixel data in the next row.
22517: (16)                         samples: The data block to keep the samples in.
22518: (16)                         Returns a pointer to the new pixmap. Throws exception on
failure to
22519: (16)                         allocate.
22520: (8)                  |
22521: (8)                  *Overload 13:*
22522: (9)                   Constructor using `pdf_new_pixmap_from_annot()`.
22523: (8)                  |
22524: (8)                  *Overload 14:*
22525: (9)                   Constructor using
`pdf_new_pixmap_from_page_contents_with_separations_and_usage()`.
22526: (8)                  |
22527: (8)                  *Overload 15:*
22528: (9)                   Constructor using `pdf_new_pixmap_from_page_contents_with_usage()`.
22529: (8)                  |
22530: (8)                  *Overload 16:*
22531: (9)                   Copy constructor using `fz_keep_pixmap()`.
22532: (8)                  |
22533: (8)                  *Overload 17:*
22534: (9)                   Default constructor, sets `m_internal` to null.
22535: (8)                  |
22536: (8)                  *Overload 18:*
22537: (9)                   Constructor using raw copy of pre-existing `::fz_pixmap`.
22538: (8)                  """
22539: (8)                  _mupdf.FzPixmap_swiginit(self, _mupdf.new_FzPixmap(*args))
22540: (4)              def storable(self):
22541: (8)                  return _mupdf.FzPixmap_storable(self)
22542: (4)              def x(self):
```

```
22543: (8)                 return _mupdf.FzPixmap_x(self)
22544: (4)             def y(self):
22545: (8)                 return _mupdf.FzPixmap_y(self)
22546: (4)             def w(self):
22547: (8)                 return _mupdf.FzPixmap_w(self)
22548: (4)             def h(self):
22549: (8)                 return _mupdf.FzPixmap_h(self)
22550: (4)             def n(self):
22551: (8)                 return _mupdf.FzPixmap_n(self)
22552: (4)             def s(self):
22553: (8)                 return _mupdf.FzPixmap_s(self)
22554: (4)             def alpha(self):
22555: (8)                 return _mupdf.FzPixmap_alpha(self)
22556: (4)             def flags(self):
22557: (8)                 return _mupdf.FzPixmap_flags(self)
22558: (4)             def stride(self):
22559: (8)                 return _mupdf.FzPixmap_stride(self)
22560: (4)             def seps(self):
22561: (8)                 return _mupdf.FzPixmap_seps(self)
22562: (4)             def xres(self):
22563: (8)                 return _mupdf.FzPixmap_xres(self)
22564: (4)             def yres(self):
22565: (8)                 return _mupdf.FzPixmap_yres(self)
22566: (4)             def colorspace(self):
22567: (8)                 return _mupdf.FzPixmap_colorspace(self)
22568: (4)             def samples(self):
22569: (8)                 return _mupdf.FzPixmap_samples(self)
22570: (4)             def underlying(self):
22571: (8)                 return _mupdf.FzPixmap_underlying(self)
22572: (4)             __swig_destroy__ = _mupdf.delete_FzPixmap
22573: (4)             def m_internal_value(self):
22574: (8)                 r""" Return numerical value of .m_internal; helps with Python
debugging."""
22575: (8)                 return _mupdf.FzPixmap_m_internal_value(self)
22576: (4)             m_internal = property(_mupdf.FzPixmap_m_internal_get,
_mupdf.FzPixmap_m_internal_set, doc=r""" Pointer to wrapped data.""")
22577: (4)             s_num_instances = property(_mupdf.FzPixmap_s_num_instances_get,
_mupdf.FzPixmap_s_num_instances_set)
22578: (0)         _mupdf.FzPixmap_swigregister(FzPixmap)
22579: (0)         class FzPixmapImage(object):
22580: (4)             r""" Wrapper class for struct `fz_pixmap_image`. Not copyable or
assignable."""
22581: (4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22582: (4)             __repr__ = _swig_repr
22583: (4)             def fz_pixmap_image_tile(self):
22584: (8)                 r"""
22585: (8)                 Class-aware wrapper for `::fz_pixmap_image_tile()`.
22586: (12)                    Retrieve the underlying fz_pixmap for an image.
22587: (12)                    Returns a pointer to the underlying fz_pixmap for an image,
22588: (12)                    or NULL if this image is not based upon an fz_pixmap.
22589: (12)                    No reference is returned. Lifespan is limited to that of
22590: (12)                    the image itself. If required, use fz_keep_pixmap to take
22591: (12)                    a reference to keep it longer.
22592: (8)                 """
22593: (8)                 return _mupdf.FzPixmapImage_fz_pixmap_image_tile(self)
22594: (4)             def fz_set_pixmap_image_tile(self, pix):
22595: (8)                 r""" Class-aware wrapper for `::fz_set_pixmap_image_tile()`."""
22596: (8)                 return _mupdf.FzPixmapImage_fz_set_pixmap_image_tile(self, pix)
22597: (4)             def __init__(self, *args):
22598: (8)                 r"""
22599: (8)                 *Overload 1:*
22600: (8)                 Default constructor, sets `m_internal` to null.
22601: (8)                 |
22602: (8)                 *Overload 2:*
22603: (8)                 Constructor using raw copy of pre-existing `::fz_pixmap_image`.
22604: (8)                 """
22605: (8)                 _mupdf.FzPixmapImage_swiginit(self, _mupdf.new_FzPixmapImage(*args))
22606: (4)             __swig_destroy__ = _mupdf.delete_FzPixmapImage
```

```
22607: (4)              def m_internal_value(self):
22608: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
22609: (8)                  return _mupdf.FzPixmapImage_m_internal_value(self)
22610: (4)              m_internal = property(_mupdf.FzPixmapImage_m_internal_get,
_mupdf.FzPixmapImage_m_internal_set, doc=r""" Pointer to wrapped data.""")
22611: (4)              s_num_instances = property(_mupdf.FzPixmapImage_s_num_instances_get,
_mupdf.FzPixmapImage_s_num_instances_set)
22612: (0)          _mupdf.FzPixmapImage_swigregister(FzPixmapImage)
22613: (0)          class FzPoint(object):
22614: (4)              r"""
22615: (4)              Wrapper class for struct `fz_point`.
22616: (4)              fz_point is a point in a two-dimensional space.
22617: (4)              """
22618: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22619: (4)              __repr__ = _swig_repr
22620: (4)              @staticmethod
22621: (4)              def fz_transform_point_xy(x, y, m):
22622: (8)                  r""" Class-aware wrapper for `::fz_transform_point_xy()`."""
22623: (8)                  return _mupdf.FzPoint_fz_transform_point_xy(x, y, m)
22624: (4)              def fz_is_point_inside_quad(self, q):
22625: (8)                  r"""
22626: (8)                  Class-aware wrapper for `::fz_is_point_inside_quad()`.
22627: (12)                     Inclusion test for quads.
22628: (8)                  """
22629: (8)                  return _mupdf.FzPoint_fz_is_point_inside_quad(self, q)
22630: (4)              def fz_is_point_inside_rect(self, r):
22631: (8)                  r"""
22632: (8)                  Class-aware wrapper for `::fz_is_point_inside_rect()`.
22633: (12)                     Inclusion test for rects. (Rect is assumed to be open, i.e.
22634: (12)                     top right corner is not included).
22635: (8)                  """
22636: (8)                  return _mupdf.FzPoint_fz_is_point_inside_rect(self, r)
22637: (4)              def fz_normalize_vector(self):
22638: (8)                  r"""
22639: (8)                  Class-aware wrapper for `::fz_normalize_vector()`.
22640: (12)                     Normalize a vector to length one.
22641: (8)                  """
22642: (8)                  return _mupdf.FzPoint_fz_normalize_vector(self)
22643: (4)              def fz_transform_point(self, *args):
22644: (8)                  r"""
22645: (8)                  *Overload 1:*
22646: (9)                   Class-aware wrapper for `::fz_transform_point()`.
22647: (16)                         Apply a transformation to a point.
22648: (16)                         transform: Transformation matrix to apply. See fz_concat,
22649: (16)                         fz_scale, fz_rotate and fz_translate for how to create a
22650: (16)                         matrix.
22651: (16)                         point: Pointer to point to update.
22652: (16)                         Returns transform (unchanged).
22653: (8)                  |
22654: (8)                  *Overload 2:*
22655: (9)                   Class-aware wrapper for `::fz_transform_point()`.
22656: (16)                         Apply a transformation to a point.
22657: (16)                         transform: Transformation matrix to apply. See fz_concat,
22658: (16)                         fz_scale, fz_rotate and fz_translate for how to create a
22659: (16)                         matrix.
22660: (16)                         point: Pointer to point to update.
22661: (16)                         Returns transform (unchanged).
22662: (8)                  """
22663: (8)                  return _mupdf.FzPoint_fz_transform_point(self, *args)
22664: (4)              def fz_transform_vector(self, *args):
22665: (8)                  r"""
22666: (8)                  *Overload 1:*
22667: (9)                   Class-aware wrapper for `::fz_transform_vector()`.
22668: (16)                         Apply a transformation to a vector.
22669: (16)                         transform: Transformation matrix to apply. See fz_concat,
22670: (16)                         fz_scale and fz_rotate for how to create a matrix. Any
22671: (16)                         translation will be ignored.
```

```
22672: (16)                         vector: Pointer to vector to update.
22673: (8)                  |
22674: (8)                  *Overload 2:*
22675: (9)                   Class-aware wrapper for `::fz_transform_vector()`.
22676: (16)                         Apply a transformation to a vector.
22677: (16)                         transform: Transformation matrix to apply. See fz_concat,
22678: (16)                         fz_scale and fz_rotate for how to create a matrix. Any
22679: (16)                         translation will be ignored.
22680: (16)                         vector: Pointer to vector to update.
22681: (8)                  """
22682: (8)                  return _mupdf.FzPoint_fz_transform_vector(self, *args)
22683: (4)              def transform(self, m):
22684: (8)                  r""" Post-multiply *this by <m> and return *this."""
22685: (8)                  return _mupdf.FzPoint_transform(self, m)
22686: (4)              def __init__(self, *args):
22687: (8)                  r"""
22688: (8)                  *Overload 1:*
22689: (8)                  Construct using specified values.
22690: (8)                  |
22691: (8)                  *Overload 2:*
22692: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
22693: (8)                  |
22694: (8)                  *Overload 3:*
22695: (8)                  Constructor using raw copy of pre-existing `::fz_point`.
22696: (8)                  |
22697: (8)                  *Overload 4:*
22698: (8)                  Constructor using raw copy of pre-existing `::fz_point`.
22699: (8)                  """
22700: (8)                  _mupdf.FzPoint_swiginit(self, _mupdf.new_FzPoint(*args))
22701: (4)              def internal(self, *args):
22702: (8)                  r"""
22703: (8)                  *Overload 1:*
22704: (8)                  Access as underlying struct.
22705: (8)                  |
22706: (8)                  *Overload 2:*
22707: (8)                  Access as underlying struct.
22708: (8)                  """
22709: (8)                  return _mupdf.FzPoint_internal(self, *args)
22710: (4)              __swig_destroy__ = _mupdf.delete_FzPoint
22711: (4)              x = property(_mupdf.FzPoint_x_get, _mupdf.FzPoint_x_set)
22712: (4)              y = property(_mupdf.FzPoint_y_get, _mupdf.FzPoint_y_set)
22713: (4)              s_num_instances = property(_mupdf.FzPoint_s_num_instances_get,
_mupdf.FzPoint_s_num_instances_set)
22714: (4)              def to_string(self):
22715: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
22716: (8)                  return _mupdf.FzPoint_to_string(self)
22717: (4)              def __eq__(self, rhs):
22718: (8)                  r""" Comparison method."""
22719: (8)                  return _mupdf.FzPoint___eq__(self, rhs)
22720: (4)              def __ne__(self, rhs):
22721: (8)                  r""" Comparison method."""
22722: (8)                  return _mupdf.FzPoint___ne__(self, rhs)
22723: (0)          _mupdf.FzPoint_swigregister(FzPoint)
22724: (0)          class FzPool(object):
22725: (4)              r"""
22726: (4)              Wrapper class for struct `fz_pool`. Not copyable or assignable.
22727: (4)              Simple pool allocators.
22728: (4)              Allocate from the pool, which can then be freed at once.
22729: (4)              """
22730: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22731: (4)              __repr__ = _swig_repr
22732: (4)              def fz_pool_alloc(self, size):
22733: (8)                  r"""
22734: (8)                  Class-aware wrapper for `::fz_pool_alloc()`.
22735: (12)                     Allocate a block of size bytes from the pool.
22736: (8)                  """
```

```
22737: (8)                  return _mupdf.FzPool_fz_pool_alloc(self, size)
22738: (4)              def fz_pool_size(self):
22739: (8)                  r"""
22740: (8)                  Class-aware wrapper for `::fz_pool_size()`.
22741: (12)                     The current size of the pool.
22742: (12)                     The number of bytes of storage currently allocated to the pool.
22743: (12)                     This is the total of the storage used for the blocks making
22744: (12)                     up the pool, rather then total of the allocated blocks so far,
22745: (12)                     so it will increase in 'lumps'.
22746: (12)                     from the pool, then the pool size may still be X
22747: (8)                  """
22748: (8)                  return _mupdf.FzPool_fz_pool_size(self)
22749: (4)              def fz_pool_strdup(self, s):
22750: (8)                  r"""
22751: (8)                  Class-aware wrapper for `::fz_pool_strdup()`.
22752: (12)                     strdup equivalent allocating from the pool.
22753: (8)                  """
22754: (8)                  return _mupdf.FzPool_fz_pool_strdup(self, s)
22755: (4)              def fz_xml_add_att(self, node, key, val):
22756: (8)                  r"""
22757: (8)                  Class-aware wrapper for `::fz_xml_add_att()`.
22758: (12)                     Add an attribute to an XML node.
22759: (8)                  """
22760: (8)                  return _mupdf.FzPool_fz_xml_add_att(self, node, key, val)
22761: (4)              def __init__(self, *args):
22762: (8)                  r"""
22763: (8)                  *Overload 1:*
22764: (9)                   == Constructors.  Constructor using `fz_new_pool()`.
22765: (16)                         Create a new pool to allocate from.
22766: (8)                  |
22767: (8)                  *Overload 2:*
22768: (9)                   Constructor using raw copy of pre-existing `::fz_pool`.
22769: (8)                  """
22770: (8)                  _mupdf.FzPool_swiginit(self, _mupdf.new_FzPool(*args))
22771: (4)              __swig_destroy__ = _mupdf.delete_FzPool
22772: (4)              def m_internal_value(self):
22773: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
22774: (8)                  return _mupdf.FzPool_m_internal_value(self)
22775: (4)              m_internal = property(_mupdf.FzPool_m_internal_get,
_mupdf.FzPool_m_internal_set, doc=r""" Pointer to wrapped data.""")
22776: (4)              s_num_instances = property(_mupdf.FzPool_s_num_instances_get,
_mupdf.FzPool_s_num_instances_set)
22777: (0)          _mupdf.FzPool_swigregister(FzPool)
22778: (0)          class FzPtrHeap(object):
22779: (4)              r""" Wrapper class for struct `fz_ptr_heap`. Not copyable or
assignable."""
22780: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22781: (4)              __repr__ = _swig_repr
22782: (4)              def fz_ptr_heap_insert(self, v, HEAP_CMP):
22783: (8)                  r""" Class-aware wrapper for `::fz_ptr_heap_insert()`."""
22784: (8)                  return _mupdf.FzPtrHeap_fz_ptr_heap_insert(self, v, HEAP_CMP)
22785: (4)              def fz_ptr_heap_sort(self, HEAP_CMP):
22786: (8)                  r""" Class-aware wrapper for `::fz_ptr_heap_sort()`."""
22787: (8)                  return _mupdf.FzPtrHeap_fz_ptr_heap_sort(self, HEAP_CMP)
22788: (4)              def fz_ptr_heap_uniq(self, HEAP_CMP):
22789: (8)                  r""" Class-aware wrapper for `::fz_ptr_heap_uniq()`."""
22790: (8)                  return _mupdf.FzPtrHeap_fz_ptr_heap_uniq(self, HEAP_CMP)
22791: (4)              def __init__(self, *args):
22792: (8)                  r"""
22793: (8)                  *Overload 1:*
22794: (8)                  Default constructor, sets `m_internal` to null.
22795: (8)                  |
22796: (8)                  *Overload 2:*
22797: (8)                  Constructor using raw copy of pre-existing `::fz_ptr_heap`.
22798: (8)                  """
22799: (8)                  _mupdf.FzPtrHeap_swiginit(self, _mupdf.new_FzPtrHeap(*args))
22800: (4)              __swig_destroy__ = _mupdf.delete_FzPtrHeap
```

```
22801: (4)              def m_internal_value(self):
22802: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
22803: (8)                  return _mupdf.FzPtrHeap_m_internal_value(self)
22804: (4)              m_internal = property(_mupdf.FzPtrHeap_m_internal_get,
_mupdf.FzPtrHeap_m_internal_set, doc=r""" Pointer to wrapped data.""")
22805: (4)              s_num_instances = property(_mupdf.FzPtrHeap_s_num_instances_get,
_mupdf.FzPtrHeap_s_num_instances_set)
22806: (0)          _mupdf.FzPtrHeap_swigregister(FzPtrHeap)
22807: (0)          class FzPwgOptions(object):
22808: (4)              r""" Wrapper class for struct `fz_pwg_options`."""
22809: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22810: (4)              __repr__ = _swig_repr
22811: (4)              def __init__(self, *args):
22812: (8)                  r"""
22813: (8)                  *Overload 1:*
22814: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
22815: (8)                  |
22816: (8)                  *Overload 2:*
22817: (8)                  Constructor using raw copy of pre-existing `::fz_pwg_options`.
22818: (8)                  """
22819: (8)                  _mupdf.FzPwgOptions_swiginit(self, _mupdf.new_FzPwgOptions(*args))
22820: (4)              def media_class(self):
22821: (8)                  return _mupdf.FzPwgOptions_media_class(self)
22822: (4)              def media_color(self):
22823: (8)                  return _mupdf.FzPwgOptions_media_color(self)
22824: (4)              def media_type(self):
22825: (8)                  return _mupdf.FzPwgOptions_media_type(self)
22826: (4)              def output_type(self):
22827: (8)                  return _mupdf.FzPwgOptions_output_type(self)
22828: (4)              def advance_distance(self):
22829: (8)                  return _mupdf.FzPwgOptions_advance_distance(self)
22830: (4)              def advance_media(self):
22831: (8)                  return _mupdf.FzPwgOptions_advance_media(self)
22832: (4)              def collate(self):
22833: (8)                  return _mupdf.FzPwgOptions_collate(self)
22834: (4)              def cut_media(self):
22835: (8)                  return _mupdf.FzPwgOptions_cut_media(self)
22836: (4)              def duplex(self):
22837: (8)                  return _mupdf.FzPwgOptions_duplex(self)
22838: (4)              def insert_sheet(self):
22839: (8)                  return _mupdf.FzPwgOptions_insert_sheet(self)
22840: (4)              def jog(self):
22841: (8)                  return _mupdf.FzPwgOptions_jog(self)
22842: (4)              def leading_edge(self):
22843: (8)                  return _mupdf.FzPwgOptions_leading_edge(self)
22844: (4)              def manual_feed(self):
22845: (8)                  return _mupdf.FzPwgOptions_manual_feed(self)
22846: (4)              def media_position(self):
22847: (8)                  return _mupdf.FzPwgOptions_media_position(self)
22848: (4)              def media_weight(self):
22849: (8)                  return _mupdf.FzPwgOptions_media_weight(self)
22850: (4)              def mirror_print(self):
22851: (8)                  return _mupdf.FzPwgOptions_mirror_print(self)
22852: (4)              def negative_print(self):
22853: (8)                  return _mupdf.FzPwgOptions_negative_print(self)
22854: (4)              def num_copies(self):
22855: (8)                  return _mupdf.FzPwgOptions_num_copies(self)
22856: (4)              def orientation(self):
22857: (8)                  return _mupdf.FzPwgOptions_orientation(self)
22858: (4)              def output_face_up(self):
22859: (8)                  return _mupdf.FzPwgOptions_output_face_up(self)
22860: (4)              def PageSize(self):
22861: (8)                  return _mupdf.FzPwgOptions_PageSize(self)
22862: (4)              def separations(self):
22863: (8)                  return _mupdf.FzPwgOptions_separations(self)
22864: (4)              def tray_switch(self):
```

```
22865: (8)                  return _mupdf.FzPwgOptions_tray_switch(self)
22866: (4)              def tumble(self):
22867: (8)                  return _mupdf.FzPwgOptions_tumble(self)
22868: (4)              def media_type_num(self):
22869: (8)                  return _mupdf.FzPwgOptions_media_type_num(self)
22870: (4)              def compression(self):
22871: (8)                  return _mupdf.FzPwgOptions_compression(self)
22872: (4)              def row_count(self):
22873: (8)                  return _mupdf.FzPwgOptions_row_count(self)
22874: (4)              def row_feed(self):
22875: (8)                  return _mupdf.FzPwgOptions_row_feed(self)
22876: (4)              def row_step(self):
22877: (8)                  return _mupdf.FzPwgOptions_row_step(self)
22878: (4)              def rendering_intent(self):
22879: (8)                  return _mupdf.FzPwgOptions_rendering_intent(self)
22880: (4)              def page_size_name(self):
22881: (8)                  return _mupdf.FzPwgOptions_page_size_name(self)
22882: (4)              __swig_destroy__ = _mupdf.delete_FzPwgOptions
22883: (4)              m_internal = property(_mupdf.FzPwgOptions_m_internal_get,
_mupdf.FzPwgOptions_m_internal_set)
22884: (4)              s_num_instances = property(_mupdf.FzPwgOptions_s_num_instances_get,
_mupdf.FzPwgOptions_s_num_instances_set, doc=r""" Wrapped data is held by value.""")
22885: (4)              def to_string(self):
22886: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
22887: (8)                  return _mupdf.FzPwgOptions_to_string(self)
22888: (4)              def __eq__(self, rhs):
22889: (8)                  r""" Comparison method."""
22890: (8)                  return _mupdf.FzPwgOptions___eq__(self, rhs)
22891: (4)              def __ne__(self, rhs):
22892: (8)                  r""" Comparison method."""
22893: (8)                  return _mupdf.FzPwgOptions___ne__(self, rhs)
22894: (0)          _mupdf.FzPwgOptions_swigregister(FzPwgOptions)
22895: (0)          class FzQuad(object):
22896: (4)              r"""
22897: (4)              Wrapper class for struct `fz_quad`.
22898: (4)              A representation for a region defined by 4 points.
22899: (4)              The significant difference between quads and rects is that
22900: (4)              the edges of quads are not axis aligned.
22901: (4)              """
22902: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22903: (4)              __repr__ = _swig_repr
22904: (4)              def fz_is_quad_inside_quad(self, haystack):
22905: (8)                  r"""
22906: (8)                  Class-aware wrapper for `::fz_is_quad_inside_quad()`.
22907: (12)                     Inclusion test for quad in quad.
22908: (12)                     This may break down if quads are not 'well formed'.
22909: (8)                  """
22910: (8)                  return _mupdf.FzQuad_fz_is_quad_inside_quad(self, haystack)
22911: (4)              def fz_is_quad_intersecting_quad(self, b):
22912: (8)                  r"""
22913: (8)                  Class-aware wrapper for `::fz_is_quad_intersecting_quad()`.
22914: (12)                     Intersection test for quads.
22915: (12)                     This may break down if quads are not 'well formed'.
22916: (8)                  """
22917: (8)                  return _mupdf.FzQuad_fz_is_quad_intersecting_quad(self, b)
22918: (4)              def fz_rect_from_quad(self):
22919: (8)                  r"""
22920: (8)                  Class-aware wrapper for `::fz_rect_from_quad()`.
22921: (12)                     Convert a quad to the smallest rect that covers it.
22922: (8)                  """
22923: (8)                  return _mupdf.FzQuad_fz_rect_from_quad(self)
22924: (4)              def fz_transform_quad(self, m):
22925: (8)                  r"""
22926: (8)                  Class-aware wrapper for `::fz_transform_quad()`.
22927: (12)                     Transform a quad by a matrix.
22928: (8)                  """
22929: (8)                  return _mupdf.FzQuad_fz_transform_quad(self, m)
```

```
22930: (4)              def __init__(self, *args):
22931: (8)                  r"""
22932: (8)                  *Overload 1:*
22933: (9)                   == Constructors.  Constructor using `fz_quad_from_rect()`.
22934: (16)                         Convert a rect to a quad (losslessly).
22935: (8)                  |
22936: (8)                  *Overload 2:*
22937: (9)                   Constructor using `fz_transform_quad()`.
22938: (16)                         Transform a quad by a matrix.
22939: (8)                  |
22940: (8)                  *Overload 3:*
22941: (9)                   We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
22942: (8)                  |
22943: (8)                  *Overload 4:*
22944: (9)                   Constructor using raw copy of pre-existing `::fz_quad`.
22945: (8)                  |
22946: (8)                  *Overload 5:*
22947: (9)                   Constructor using raw copy of pre-existing `::fz_quad`.
22948: (8)                  """
22949: (8)                  _mupdf.FzQuad_swiginit(self, _mupdf.new_FzQuad(*args))
22950: (4)              def internal(self, *args):
22951: (8)                  r"""
22952: (8)                  *Overload 1:*
22953: (8)                  Access as underlying struct.
22954: (8)                  |
22955: (8)                  *Overload 2:*
22956: (8)                  Access as underlying struct.
22957: (8)                  """
22958: (8)                  return _mupdf.FzQuad_internal(self, *args)
22959: (4)              __swig_destroy__ = _mupdf.delete_FzQuad
22960: (4)              ul = property(_mupdf.FzQuad_ul_get, _mupdf.FzQuad_ul_set)
22961: (4)              ur = property(_mupdf.FzQuad_ur_get, _mupdf.FzQuad_ur_set)
22962: (4)              ll = property(_mupdf.FzQuad_ll_get, _mupdf.FzQuad_ll_set)
22963: (4)              lr = property(_mupdf.FzQuad_lr_get, _mupdf.FzQuad_lr_set)
22964: (4)              s_num_instances = property(_mupdf.FzQuad_s_num_instances_get,
_mupdf.FzQuad_s_num_instances_set)
22965: (4)              def to_string(self):
22966: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
22967: (8)                  return _mupdf.FzQuad_to_string(self)
22968: (4)              def __eq__(self, rhs):
22969: (8)                  r""" Comparison method."""
22970: (8)                  return _mupdf.FzQuad___eq__(self, rhs)
22971: (4)              def __ne__(self, rhs):
22972: (8)                  r""" Comparison method."""
22973: (8)                  return _mupdf.FzQuad___ne__(self, rhs)
22974: (0)          _mupdf.FzQuad_swigregister(FzQuad)
22975: (0)          class FzRange(object):
22976: (4)              r""" Wrapper class for struct `fz_range`. Not copyable or assignable."""
22977: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22978: (4)              __repr__ = _swig_repr
22979: (4)              def __init__(self, *args):
22980: (8)                  r"""
22981: (8)                  *Overload 1:*
22982: (8)                  Default constructor, sets `m_internal` to null.
22983: (8)                  |
22984: (8)                  *Overload 2:*
22985: (8)                  Constructor using raw copy of pre-existing `::fz_range`.
22986: (8)                  """
22987: (8)                  _mupdf.FzRange_swiginit(self, _mupdf.new_FzRange(*args))
22988: (4)              __swig_destroy__ = _mupdf.delete_FzRange
22989: (4)              def m_internal_value(self):
22990: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
22991: (8)                  return _mupdf.FzRange_m_internal_value(self)
22992: (4)              m_internal = property(_mupdf.FzRange_m_internal_get,
_mupdf.FzRange_m_internal_set, doc=r""" Pointer to wrapped data.""")
```

```
22993: (4)              s_num_instances = property(_mupdf.FzRange_s_num_instances_get,
_mupdf.FzRange_s_num_instances_set)
22994: (0)          _mupdf.FzRange_swigregister(FzRange)
22995: (0)          class FzRect(object):
22996: (4)              r""" Wrapper class for struct `fz_rect`."""
22997: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
22998: (4)              __repr__ = _swig_repr
22999: (4)              Fixed_UNIT = _mupdf.FzRect_Fixed_UNIT
23000: (4)              Fixed_EMPTY = _mupdf.FzRect_Fixed_EMPTY
23001: (4)              Fixed_INFINITE = _mupdf.FzRect_Fixed_INFINITE
23002: (4)              def fz_adjust_rect_for_stroke(self, stroke, ctm):
23003: (8)                  r"""
23004: (8)                  Class-aware wrapper for `::fz_adjust_rect_for_stroke()`.
23005: (12)                     Given a rectangle (assumed to be the bounding box for a path),
23006: (12)                     expand it to allow for the expansion of the bbox that would be
23007: (12)                     seen by stroking the path with the given stroke state and
23008: (12)                     transform.
23009: (8)                  """
23010: (8)                  return _mupdf.FzRect_fz_adjust_rect_for_stroke(self, stroke, ctm)
23011: (4)              def fz_contains_rect(self, b):
23012: (8)                  r"""
23013: (8)                  Class-aware wrapper for `::fz_contains_rect()`.
23014: (12)                     Test rectangle inclusion.
23015: (12)                     Return true if a entirely contains b.
23016: (8)                  """
23017: (8)                  return _mupdf.FzRect_fz_contains_rect(self, b)
23018: (4)              def fz_expand_rect(self, expand):
23019: (8)                  r"""
23020: (8)                  Class-aware wrapper for `::fz_expand_rect()`.
23021: (12)                     Expand a bbox by a given amount in all directions.
23022: (8)                  """
23023: (8)                  return _mupdf.FzRect_fz_expand_rect(self, expand)
23024: (4)              def fz_include_point_in_rect(self, p):
23025: (8)                  r"""
23026: (8)                  Class-aware wrapper for `::fz_include_point_in_rect()`.
23027: (12)                     Expand a bbox to include a given point.
23028: (12)                     To create a rectangle that encompasses a sequence of points, the
23029: (12)                     rectangle must first be set to be the empty rectangle at one of
23030: (12)                     the points before including the others.
23031: (8)                  """
23032: (8)                  return _mupdf.FzRect_fz_include_point_in_rect(self, p)
23033: (4)              def fz_intersect_rect(self, *args):
23034: (8)                  r"""
23035: (8)                  *Overload 1:*
23036: (9)                   Class-aware wrapper for `::fz_intersect_rect()`.
23037: (16)                         Compute intersection of two rectangles.
23038: (16)                         Given two rectangles, update the first to be the smallest
23039: (16)                         axis-aligned rectangle that covers the area covered by both
23040: (16)                         given rectangles. If either rectangle is empty then the
23041: (16)                         intersection is also empty. If either rectangle is infinite
23042: (16)                         then the intersection is simply the non-infinite rectangle.
23043: (16)                         Should both rectangles be infinite, then the intersection is
23044: (16)                         also infinite.
23045: (8)                  |
23046: (8)                  *Overload 2:*
23047: (9)                   Class-aware wrapper for `::fz_intersect_rect()`.
23048: (16)                         Compute intersection of two rectangles.
23049: (16)                         Given two rectangles, update the first to be the smallest
23050: (16)                         axis-aligned rectangle that covers the area covered by both
23051: (16)                         given rectangles. If either rectangle is empty then the
23052: (16)                         intersection is also empty. If either rectangle is infinite
23053: (16)                         then the intersection is simply the non-infinite rectangle.
23054: (16)                         Should both rectangles be infinite, then the intersection is
23055: (16)                         also infinite.
23056: (8)                  """
23057: (8)                  return _mupdf.FzRect_fz_intersect_rect(self, *args)
23058: (4)              def fz_irect_from_rect(self):
23059: (8)                  r"""
```

```
(8)                 Class-aware wrapper for `::fz_irect_from_rect()`.
(12)                    Convert a rect into the minimal bounding box
(12)                    that covers the rectangle.
(12)                    Coordinates in a bounding box are integers, so rounding of the
(12)                    rects coordinates takes place. The top left corner is rounded
(12)                    upwards and left while the bottom right corner is rounded
(12)                    downwards and to the right.
(8)                 """
(8)                 return _mupdf.FzRect_fz_irect_from_rect(self)
(4)             def fz_is_empty_rect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_empty_rect()`.
(12)                    Check if rectangle is empty.
(12)                    An empty rectangle is defined as one whose area is zero.
(12)                    All invalid rectangles are empty.
(8)                 """
(8)                 return _mupdf.FzRect_fz_is_empty_rect(self)
(4)             def fz_is_infinite_rect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_infinite_rect()`.
(12)                    Check if rectangle is infinite.
(8)                 """
(8)                 return _mupdf.FzRect_fz_is_infinite_rect(self)
(4)             def fz_is_valid_rect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_is_valid_rect()`.
(12)                    Check if rectangle is valid.
(8)                 """
(8)                 return _mupdf.FzRect_fz_is_valid_rect(self)
(4)             def fz_new_bbox_device(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_bbox_device()`.
(12)                    Create a device to compute the bounding
(12)                    box of all marks on a page.
(12)                    The returned bounding box will be the union of all bounding
(12)                    boxes of all objects on a page.
(8)                 """
(8)                 return _mupdf.FzRect_fz_new_bbox_device(self)
(4)             def fz_new_display_list(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_new_display_list()`.
(12)                    Create an empty display list.
(12)                    A display list contains drawing commands (text, images, etc.).
(12)                    Use fz_new_list_device for populating the list.
(12)                    mediabox: Bounds of the page (in points) represented by the
(12)                    display list.
(8)                 """
(8)                 return _mupdf.FzRect_fz_new_display_list(self)
(4)             def fz_quad_from_rect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_quad_from_rect()`.
(12)                    Convert a rect to a quad (losslessly).
(8)                 """
(8)                 return _mupdf.FzRect_fz_quad_from_rect(self)
(4)             def fz_round_rect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_round_rect()`.
(12)                    Round rectangle coordinates.
(12)                    Coordinates in a bounding box are integers, so rounding of the
(12)                    rects coordinates takes place. The top left corner is rounded
(12)                    upwards and left while the bottom right corner is rounded
(12)                    downwards and to the right.
(12)                    This differs from fz_irect_from_rect, in that fz_irect_from_rect
(12)                    slavishly follows the numbers (i.e any slight over/under
(12)                    calculations can cause whole extra pixels to be added).
(12)                    fz_round_rect allows for a small amount of rounding error when
(12)                    calculating the bbox.
(8)                 """
(8)                 return _mupdf.FzRect_fz_round_rect(self)
```

```
(4)             def fz_transform_page(self, resolution, rotate):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_transform_page()`.
(12)                    Create transform matrix to draw page
(12)                    at a given resolution and rotation. Adjusts the scaling
(12)                    factors so that the page covers whole number of
(12)                    pixels and adjust the page origin to be at 0,0.
(8)                 """
(8)                 return _mupdf.FzRect_fz_transform_page(self, resolution, rotate)
(4)             def fz_transform_rect(self, m):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_transform_rect()`.
(12)                    Apply a transform to a rectangle.
(12)                    After the four corner points of the axis-aligned rectangle
(12)                    have been transformed it may not longer be axis-aligned. So a
(12)                    new axis-aligned rectangle is created covering at least the
(12)                    area of the transformed rectangle.
(12)                    transform: Transformation matrix to apply. See fz_concat,
(12)                    fz_scale and fz_rotate for how to create a matrix.
(12)                    rect: Rectangle to be transformed. The two special cases
(12)                    fz_empty_rect and fz_infinite_rect, may be used but are
(12)                    returned unchanged as expected.
(8)                 """
(8)                 return _mupdf.FzRect_fz_transform_rect(self, m)
(4)             def fz_translate_rect(self, xoff, yoff):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_translate_rect()`.
(12)                    Translate bounding box.
(12)                    Translate a bbox by a given x and y offset. Allows for overflow.
(8)                 """
(8)                 return _mupdf.FzRect_fz_translate_rect(self, xoff, yoff)
(4)             def fz_union_rect(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  Class-aware wrapper for `::fz_union_rect()`.
(16)                        Compute union of two rectangles.
(16)                        Given two rectangles, update the first to be the smallest
(16)                        axis-aligned rectangle that encompasses both given rectangles.
(16)                        If either rectangle is infinite then the union is also
infinite.
(16)                        If either rectangle is empty then the union is simply the
(16)                        non-empty rectangle. Should both rectangles be empty, then the
(16)                        union is also empty.
(8)                 |
(8)                 *Overload 2:*
(9)                  Class-aware wrapper for `::fz_union_rect()`.
(16)                        Compute union of two rectangles.
(16)                        Given two rectangles, update the first to be the smallest
(16)                        axis-aligned rectangle that encompasses both given rectangles.
(16)                        If either rectangle is infinite then the union is also
infinite.
(16)                        If either rectangle is empty then the union is simply the
(16)                        non-empty rectangle. Should both rectangles be empty, then the
(16)                        union is also empty.
(8)                 """
(8)                 return _mupdf.FzRect_fz_union_rect(self, *args)
(4)             def pdf_signature_appearance_signed(self, lang, img, left_text,
right_text, include_logo):
(8)                 r""" Class-aware wrapper for `::pdf_signature_appearance_signed()`."""
(8)                 return _mupdf.FzRect_pdf_signature_appearance_signed(self, lang, img,
left_text, right_text, include_logo)
(4)             def pdf_signature_appearance_unsigned(self, lang):
(8)                 r""" Class-aware wrapper for
`::pdf_signature_appearance_unsigned()`."""
(8)                 return _mupdf.FzRect_pdf_signature_appearance_unsigned(self, lang)
(4)             def transform(self, m):
(8)                 r""" Transforms *this using fz_transform_rect() with <m>."""
(8)                 return _mupdf.FzRect_transform(self, m)
(4)             def contains(self, *args):
```

```
(8)                 r"""
(8)                 *Overload 1:*
(8)                 Convenience method using fz_contains_rect().
(8)                 |
(8)                 *Overload 2:*
(8)                 Uses fz_contains_rect(*this, rhs).
(8)                 """
(8)                 return _mupdf.FzRect_contains(self, *args)
(4)             def is_empty(self):
(8)                 r""" Uses fz_is_empty_rect()."""
(8)                 return _mupdf.FzRect_is_empty(self)
(4)             def union_(self, rhs):
(8)                 r""" Updates *this using fz_union_rect()."""
(8)                 return _mupdf.FzRect_union_(self, rhs)
(4)             def __init__(self, *args):
(8)                 r"""
(8)                 *Overload 1:*
(9)                  == Constructors.  Constructor using `fz_bound_display_list()`.
(16)                        Return the bounding box of the page recorded in a display
list.
(8)                 |
(8)                 *Overload 2:*
(9)                  Constructor using `fz_rect_from_irect()`.
(16)                        Convert a bbox into a rect.
(16)                        For our purposes, a rect can represent all the values we meet
in
(16)                        a bbox, so nothing can go wrong.
(16)                        rect: A place to store the generated rectangle.
(16)                        bbox: The bbox to convert.
(16)                        Returns rect (updated).
(8)                 |
(8)                 *Overload 3:*
(9)                  Constructor using `fz_rect_from_quad()`.
(16)                        Convert a quad to the smallest rect that covers it.
(8)                 |
(8)                 *Overload 4:*
(9)                  Constructor using `fz_transform_rect()`.
(16)                        Apply a transform to a rectangle.
(16)                        After the four corner points of the axis-aligned rectangle
(16)                        have been transformed it may not longer be axis-aligned. So a
(16)                        new axis-aligned rectangle is created covering at least the
(16)                        area of the transformed rectangle.
(16)                        transform: Transformation matrix to apply. See fz_concat,
(16)                        fz_scale and fz_rotate for how to create a matrix.
(16)                        rect: Rectangle to be transformed. The two special cases
(16)                        fz_empty_rect and fz_infinite_rect, may be used but are
(16)                        returned unchanged as expected.
(8)                 |
(8)                 *Overload 5:*
(9)                  Construct from specified values.
(8)                 |
(8)                 *Overload 6:*
(9)                  Copy constructor using plain copy.
(8)                 |
(8)                 *Overload 7:*
(9)                  Construct from fz_unit_rect, fz_empty_rect or fz_infinite_rect.
(8)                 |
(8)                 *Overload 8:*
(9)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
(8)                 |
(8)                 *Overload 9:*
(9)                  Constructor using raw copy of pre-existing `::fz_rect`.
(8)                 |
(8)                 *Overload 10:*
(9)                  Constructor using raw copy of pre-existing `::fz_rect`.
(8)                 """
(8)                 _mupdf.FzRect_swiginit(self, _mupdf.new_FzRect(*args))
(4)             def internal(self, *args):
```

```
23259: (8)                  r"""
23260: (8)                  *Overload 1:*
23261: (8)                  Access as underlying struct.
23262: (8)                  |
23263: (8)                  *Overload 2:*
23264: (8)                  Access as underlying struct.
23265: (8)                  """
23266: (8)                  return _mupdf.FzRect_internal(self, *args)
23267: (4)              __swig_destroy__ = _mupdf.delete_FzRect
23268: (4)              x0 = property(_mupdf.FzRect_x0_get, _mupdf.FzRect_x0_set)
23269: (4)              y0 = property(_mupdf.FzRect_y0_get, _mupdf.FzRect_y0_set)
23270: (4)              x1 = property(_mupdf.FzRect_x1_get, _mupdf.FzRect_x1_set)
23271: (4)              y1 = property(_mupdf.FzRect_y1_get, _mupdf.FzRect_y1_set)
23272: (4)              s_num_instances = property(_mupdf.FzRect_s_num_instances_get,
_mupdf.FzRect_s_num_instances_set)
23273: (4)              def to_string(self):
23274: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
23275: (8)                  return _mupdf.FzRect_to_string(self)
23276: (4)              def __eq__(self, rhs):
23277: (8)                  r""" Comparison method."""
23278: (8)                  return _mupdf.FzRect___eq__(self, rhs)
23279: (4)              def __ne__(self, rhs):
23280: (8)                  r""" Comparison method."""
23281: (8)                  return _mupdf.FzRect___ne__(self, rhs)
23282: (0)          _mupdf.FzRect_swigregister(FzRect)
23283: (0)          class FzSeparations(object):
23284: (4)              """ Wrapper class for struct `fz_separations`."""
23285: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23286: (4)              __repr__ = _swig_repr
23287: (4)              def fz_add_separation(self, name, cs, cs_channel):
23288: (8)                  r"""
23289: (8)                  Class-aware wrapper for `::fz_add_separation()`.
23290: (12)                     Add a separation (null terminated name, colorspace)
23291: (8)                  """
23292: (8)                  return _mupdf.FzSeparations_fz_add_separation(self, name, cs,
cs_channel)
23293: (4)              def fz_add_separation_equivalents(self, rgba, cmyk, name):
23294: (8)                  r"""
23295: (8)                  Class-aware wrapper for `::fz_add_separation_equivalents()`.
23296: (12)                     Add a separation with equivalents (null terminated name,
23297: (12)                     colorspace)
23298: (12)                     (old, deprecated)
23299: (8)                  """
23300: (8)                  return _mupdf.FzSeparations_fz_add_separation_equivalents(self, rgba,
cmyk, name)
23301: (4)              def fz_clone_separations_for_overprint(self):
23302: (8)                  r"""
23303: (8)                  Class-aware wrapper for `::fz_clone_separations_for_overprint()`.
23304: (12)                     Return a separations object with all the spots in the input
23305: (12)                     separations object that are set to composite, reset to be
23306: (12)                     enabled. If there ARE no spots in the object, this returns
23307: (12)                     NULL. If the object already has all its spots enabled, then
23308: (12)                     just returns another handle on the same object.
23309: (8)                  """
23310: (8)                  return _mupdf.FzSeparations_fz_clone_separations_for_overprint(self)
23311: (4)              def fz_compare_separations(self, sep2):
23312: (8)                  r"""
23313: (8)                  Class-aware wrapper for `::fz_compare_separations()`.
23314: (12)                     Compare 2 separations structures (or NULLs).
23315: (12)                     Return 0 if identical, non-zero if not identical.
23316: (8)                  """
23317: (8)                  return _mupdf.FzSeparations_fz_compare_separations(self, sep2)
23318: (4)              def fz_count_active_separations(self):
23319: (8)                  r"""
23320: (8)                  Class-aware wrapper for `::fz_count_active_separations()`.
23321: (12)                     Return the number of active separations.
23322: (8)                  """
```

```
23323: (8)                  return _mupdf.FzSeparations_fz_count_active_separations(self)
23324: (4)              def fz_count_separations(self):
23325: (8)                  r""" Class-aware wrapper for `::fz_count_separations()`."""
23326: (8)                  return _mupdf.FzSeparations_fz_count_separations(self)
23327: (4)              def fz_separation_equivalent(self, idx, dst_cs, dst_color, prf,
color_params):
23328: (8)                  r"""
23329: (8)                  Class-aware wrapper for `::fz_separation_equivalent()`.
23330: (8)                  This method has out-params. Python/C# wrappers look like:
23331: (12)                     `fz_separation_equivalent(int idx, ::fz_colorspace *dst_cs,
::fz_colorspace *prf, ::fz_color_params color_params)` => float dst_color
23332: (12)                     Get the equivalent separation color in a given colorspace.
23333: (8)                  """
23334: (8)                  return _mupdf.FzSeparations_fz_separation_equivalent(self, idx,
dst_cs, dst_color, prf, color_params)
23335: (4)              def fz_separation_name(self, separation):
23336: (8)                  r""" Class-aware wrapper for `::fz_separation_name()`."""
23337: (8)                  return _mupdf.FzSeparations_fz_separation_name(self, separation)
23338: (4)              def fz_set_separation_behavior(self, separation, behavior):
23339: (8)                  r"""
23340: (8)                  Class-aware wrapper for `::fz_set_separation_behavior()`.
23341: (12)                     Control the rendering of a given separation.
23342: (8)                  """
23343: (8)                  return _mupdf.FzSeparations_fz_set_separation_behavior(self,
separation, behavior)
23344: (4)              def __init__(self, *args):
23345: (8)                  r"""
23346: (8)                  *Overload 1:*
23347: (9)                   == Constructors.  Constructor using `fz_new_separations()`.
23348: (16)                         Create a new separations structure (initially empty)
23349: (8)                  |
23350: (8)                  *Overload 2:*
23351: (9)                   Copy constructor using `fz_keep_separations()`.
23352: (8)                  |
23353: (8)                  *Overload 3:*
23354: (9)                   Constructor using raw copy of pre-existing `::fz_separations`.
23355: (8)                  |
23356: (8)                  *Overload 4:*
23357: (9)                   Constructor using raw copy of pre-existing `::fz_separations`.
23358: (8)                  """
23359: (8)                  _mupdf.FzSeparations_swiginit(self, _mupdf.new_FzSeparations(*args))
23360: (4)              __swig_destroy__ = _mupdf.delete_FzSeparations
23361: (4)              def m_internal_value(self):
23362: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23363: (8)                  return _mupdf.FzSeparations_m_internal_value(self)
23364: (4)              m_internal = property(_mupdf.FzSeparations_m_internal_get,
_mupdf.FzSeparations_m_internal_set, doc=r""" Pointer to wrapped data.""")
23365: (4)              s_num_instances = property(_mupdf.FzSeparations_s_num_instances_get,
_mupdf.FzSeparations_s_num_instances_set)
23366: (0)          _mupdf.FzSeparations_swigregister(FzSeparations)
23367: (0)          class FzSha256(object):
23368: (4)              r"""
23369: (4)              Wrapper class for struct `fz_sha256`. Not copyable or assignable.
23370: (4)              Structure definition is public to enable stack
23371: (4)              based allocation. Do not access the members directly.
23372: (4)              """
23373: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23374: (4)              __repr__ = _swig_repr
23375: (4)              def fz_sha256_final(self, digest):
23376: (8)                  r"""
23377: (8)                  Class-aware wrapper for `::fz_sha256_final()`.
23378: (12)                     MD5 finalization. Ends an MD5 message-digest operation, writing
23379: (12)                     the message digest and zeroizing the context.
23380: (12)                     Never throws an exception.
23381: (8)                  """
23382: (8)                  return _mupdf.FzSha256_fz_sha256_final(self, digest)
23383: (4)              def fz_sha256_init(self):
```

```
23384: (8)                  r"""
23385: (8)                  Class-aware wrapper for `::fz_sha256_init()`.
23386: (12)                     SHA256 initialization. Begins an SHA256 operation, initialising
23387: (12)                     the supplied context.
23388: (12)                     Never throws an exception.
23389: (8)                  """
23390: (8)                  return _mupdf.FzSha256_fz_sha256_init(self)
23391: (4)              def fz_sha256_update(self, input, inlen):
23392: (8)                  r"""
23393: (8)                  Class-aware wrapper for `::fz_sha256_update()`.
23394: (12)                     SHA256 block update operation. Continues an SHA256 message-
23395: (12)                     digest operation, processing another message block, and updating
23396: (12)                     the context.
23397: (12)                     Never throws an exception.
23398: (8)                  """
23399: (8)                  return _mupdf.FzSha256_fz_sha256_update(self, input, inlen)
23400: (4)              def __init__(self, *args):
23401: (8)                  r"""
23402: (8)                  *Overload 1:*
23403: (8)                  Default constructor, sets `m_internal` to null.
23404: (8)                  |
23405: (8)                  *Overload 2:*
23406: (8)                  Constructor using raw copy of pre-existing `::fz_sha256`.
23407: (8)                  """
23408: (8)                  _mupdf.FzSha256_swiginit(self, _mupdf.new_FzSha256(*args))
23409: (4)              __swig_destroy__ = _mupdf.delete_FzSha256
23410: (4)              def m_internal_value(self):
23411: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23412: (8)                  return _mupdf.FzSha256_m_internal_value(self)
23413: (4)              m_internal = property(_mupdf.FzSha256_m_internal_get,
_mupdf.FzSha256_m_internal_set, doc=r""" Pointer to wrapped data.""")
23414: (4)              s_num_instances = property(_mupdf.FzSha256_s_num_instances_get,
_mupdf.FzSha256_s_num_instances_set)
23415: (0)          _mupdf.FzSha256_swigregister(FzSha256)
23416: (0)          class FzSha384(object):
23417: (4)              r""" Wrapper class for struct `fz_sha384`. Not copyable or assignable."""
23418: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23419: (4)              __repr__ = _swig_repr
23420: (4)              def __init__(self, *args):
23421: (8)                  r"""
23422: (8)                  *Overload 1:*
23423: (8)                  Default constructor, sets `m_internal` to null.
23424: (8)                  |
23425: (8)                  *Overload 2:*
23426: (8)                  Constructor using raw copy of pre-existing `::fz_sha384`.
23427: (8)                  """
23428: (8)                  _mupdf.FzSha384_swiginit(self, _mupdf.new_FzSha384(*args))
23429: (4)              __swig_destroy__ = _mupdf.delete_FzSha384
23430: (4)              def m_internal_value(self):
23431: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23432: (8)                  return _mupdf.FzSha384_m_internal_value(self)
23433: (4)              m_internal = property(_mupdf.FzSha384_m_internal_get,
_mupdf.FzSha384_m_internal_set, doc=r""" Pointer to wrapped data.""")
23434: (4)              s_num_instances = property(_mupdf.FzSha384_s_num_instances_get,
_mupdf.FzSha384_s_num_instances_set)
23435: (0)          _mupdf.FzSha384_swigregister(FzSha384)
23436: (0)          class FzSha512(object):
23437: (4)              r"""
23438: (4)              Wrapper class for struct `fz_sha512`. Not copyable or assignable.
23439: (4)              Structure definition is public to enable stack
23440: (4)              based allocation. Do not access the members directly.
23441: (4)              """
23442: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23443: (4)              __repr__ = _swig_repr
23444: (4)              def fz_sha384_final(self, digest):
```

```
23445: (8)                  r"""
23446: (8)                  Class-aware wrapper for `::fz_sha384_final()`.
23447: (12)                     SHA384 finalization. Ends an SHA384 message-digest operation,
23448: (12)                     writing the message digest and zeroizing the context.
23449: (12)                     Never throws an exception.
23450: (8)                  """
23451: (8)                  return _mupdf.FzSha512_fz_sha384_final(self, digest)
23452: (4)              def fz_sha384_init(self):
23453: (8)                  r"""
23454: (8)                  Class-aware wrapper for `::fz_sha384_init()`.
23455: (12)                     SHA384 initialization. Begins an SHA384 operation, initialising
23456: (12)                     the supplied context.
23457: (12)                     Never throws an exception.
23458: (8)                  """
23459: (8)                  return _mupdf.FzSha512_fz_sha384_init(self)
23460: (4)              def fz_sha384_update(self, input, inlen):
23461: (8)                  r"""
23462: (8)                  Class-aware wrapper for `::fz_sha384_update()`.
23463: (12)                     SHA384 block update operation. Continues an SHA384 message-
23464: (12)                     digest operation, processing another message block, and updating
23465: (12)                     the context.
23466: (12)                     Never throws an exception.
23467: (8)                  """
23468: (8)                  return _mupdf.FzSha512_fz_sha384_update(self, input, inlen)
23469: (4)              def fz_sha512_final(self, digest):
23470: (8)                  r"""
23471: (8)                  Class-aware wrapper for `::fz_sha512_final()`.
23472: (12)                     SHA512 finalization. Ends an SHA512 message-digest operation,
23473: (12)                     writing the message digest and zeroizing the context.
23474: (12)                     Never throws an exception.
23475: (8)                  """
23476: (8)                  return _mupdf.FzSha512_fz_sha512_final(self, digest)
23477: (4)              def fz_sha512_init(self):
23478: (8)                  r"""
23479: (8)                  Class-aware wrapper for `::fz_sha512_init()`.
23480: (12)                     SHA512 initialization. Begins an SHA512 operation, initialising
23481: (12)                     the supplied context.
23482: (12)                     Never throws an exception.
23483: (8)                  """
23484: (8)                  return _mupdf.FzSha512_fz_sha512_init(self)
23485: (4)              def fz_sha512_update(self, input, inlen):
23486: (8)                  r"""
23487: (8)                  Class-aware wrapper for `::fz_sha512_update()`.
23488: (12)                     SHA512 block update operation. Continues an SHA512 message-
23489: (12)                     digest operation, processing another message block, and updating
23490: (12)                     the context.
23491: (12)                     Never throws an exception.
23492: (8)                  """
23493: (8)                  return _mupdf.FzSha512_fz_sha512_update(self, input, inlen)
23494: (4)              def __init__(self, *args):
23495: (8)                  r"""
23496: (8)                  *Overload 1:*
23497: (8)                  Default constructor, sets `m_internal` to null.
23498: (8)                  |
23499: (8)                  *Overload 2:*
23500: (8)                  Constructor using raw copy of pre-existing `::fz_sha512`.
23501: (8)                  """
23502: (8)                  _mupdf.FzSha512_swiginit(self, _mupdf.new_FzSha512(*args))
23503: (4)              __swig_destroy__ = _mupdf.delete_FzSha512
23504: (4)              def m_internal_value(self):
23505: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23506: (8)                  return _mupdf.FzSha512_m_internal_value(self)
23507: (4)              m_internal = property(_mupdf.FzSha512_m_internal_get,
_mupdf.FzSha512_m_internal_set, doc=r""" Pointer to wrapped data.""")
23508: (4)              s_num_instances = property(_mupdf.FzSha512_s_num_instances_get,
_mupdf.FzSha512_s_num_instances_set)
23509: (0)          _mupdf.FzSha512_swigregister(FzSha512)
23510: (0)          class FzShade(object):
```

```
23511: (4)              r"""
23512: (4)              Wrapper class for struct `fz_shade`.
23513: (4)              Structure is public to allow derived classes. Do not
23514: (4)              access the members directly.
23515: (4)              """
23516: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23517: (4)              __repr__ = _swig_repr
23518: (4)              def fz_bound_shade(self, ctm):
23519: (8)                  r"""
23520: (8)                  Class-aware wrapper for `::fz_bound_shade()`.
23521: (12)                     Bound a given shading.
23522: (12)                     shade: The shade to bound.
23523: (12)                     ctm: The transform to apply to the shade before bounding.
23524: (12)                     r: Pointer to storage to put the bounds in.
23525: (12)                     Returns r, updated to contain the bounds for the shading.
23526: (8)                  """
23527: (8)                  return _mupdf.FzShade_fz_bound_shade(self, ctm)
23528: (4)              def fz_paint_shade(self, override_cs, ctm, dest, color_params, bbox, eop,
cache):
23529: (8)                  r"""
23530: (8)                  Class-aware wrapper for `::fz_paint_shade()`.
23531: (8)                  This method has out-params. Python/C# wrappers look like:
23532: (12)                     `fz_paint_shade(::fz_colorspace *override_cs, ::fz_matrix ctm,
::fz_pixmap *dest, ::fz_color_params color_params, ::fz_irect bbox, const ::fz_overprint *eop,
::fz_shade_color_cache **cache)` =>
23533: (12)                     Render a shade to a given pixmap.
23534: (12)                     shade: The shade to paint.
23535: (12)                     override_cs: NULL, or colorspace to override the shades
23536: (12)                     inbuilt colorspace.
23537: (12)                     ctm: The transform to apply.
23538: (12)                     dest: The pixmap to render into.
23539: (12)                     color_params: The color rendering settings
23540: (12)                     bbox: Pointer to a bounding box to limit the rendering
23541: (12)                     of the shade.
23542: (12)                     eop: NULL, or pointer to overprint bitmap.
23543: (12)                     cache: *cache is used to cache color information. If *cache is
NULL it
23544: (12)                     is set to point to a new fz_shade_color_cache. If cache is NULL it
is
23545: (12)                     ignored.
23546: (8)                  """
23547: (8)                  return _mupdf.FzShade_fz_paint_shade(self, override_cs, ctm, dest,
color_params, bbox, eop, cache)
23548: (4)              def fz_process_shade(self, ctm, scissor, prepare, process, process_arg):
23549: (8)                  r"""
23550: (8)                  Class-aware wrapper for `::fz_process_shade()`.
23551: (12)                     Process a shade, using supplied callback functions. This
23552: (12)                     decomposes the shading to a mesh (even ones that are not
23553: (12)                     natively meshes, such as linear or radial shadings), and
23554: (12)                     processes triangles from those meshes.
23555: (12)                     shade: The shade to process.
23556: (12)                     ctm: The transform to use
23557: (12)                     prepare: Callback function to 'prepare' each vertex.
23558: (12)                     This function is passed an array of floats, and populates
23559: (12)                     a fz_vertex structure.
23560: (12)                     process: This function is passed 3 pointers to vertex
23561: (12)                     structures, and actually performs the processing (typically
23562: (12)                     filling the area between the vertexes).
23563: (12)                     process_arg: An opaque argument passed through from caller
23564: (12)                     to callback functions.
23565: (8)                  """
23566: (8)                  return _mupdf.FzShade_fz_process_shade(self, ctm, scissor, prepare,
process, process_arg)
23567: (4)              def fz_paint_shade_no_cache(self, override_cs, ctm, dest, color_params,
bbox, eop):
23568: (8)                  r""" Extra wrapper for fz_paint_shade(), passing cache=NULL."""
23569: (8)                  return _mupdf.FzShade_fz_paint_shade_no_cache(self, override_cs, ctm,
dest, color_params, bbox, eop)
```

```
23570: (4)              def __init__(self, *args):
23571: (8)                  r"""
23572: (8)                  *Overload 1:*
23573: (8)                  Copy constructor using `fz_keep_shade()`.
23574: (8)                  |
23575: (8)                  *Overload 2:*
23576: (8)                  Default constructor, sets `m_internal` to null.
23577: (8)                  |
23578: (8)                  *Overload 3:*
23579: (8)                  Constructor using raw copy of pre-existing `::fz_shade`.
23580: (8)                  """
23581: (8)                  _mupdf.FzShade_swiginit(self, _mupdf.new_FzShade(*args))
23582: (4)              __swig_destroy__ = _mupdf.delete_FzShade
23583: (4)              def m_internal_value(self):
23584: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23585: (8)                  return _mupdf.FzShade_m_internal_value(self)
23586: (4)              m_internal = property(_mupdf.FzShade_m_internal_get,
_mupdf.FzShade_m_internal_set, doc=r""" Pointer to wrapped data.""")
23587: (4)              s_num_instances = property(_mupdf.FzShade_s_num_instances_get,
_mupdf.FzShade_s_num_instances_set)
23588: (0)          _mupdf.FzShade_swigregister(FzShade)
23589: (0)          class FzShadeColorCache(object):
23590: (4)              r""" Wrapper class for struct `fz_shade_color_cache`. Not copyable or
assignable."""
23591: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23592: (4)              __repr__ = _swig_repr
23593: (4)              def __init__(self, *args):
23594: (8)                  r"""
23595: (8)                  *Overload 1:*
23596: (8)                  Default constructor, sets `m_internal` to null.
23597: (8)                  |
23598: (8)                  *Overload 2:*
23599: (8)                  Constructor using raw copy of pre-existing `::fz_shade_color_cache`.
23600: (8)                  """
23601: (8)                  _mupdf.FzShadeColorCache_swiginit(self,
_mupdf.new_FzShadeColorCache(*args))
23602: (4)              __swig_destroy__ = _mupdf.delete_FzShadeColorCache
23603: (4)              def m_internal_value(self):
23604: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23605: (8)                  return _mupdf.FzShadeColorCache_m_internal_value(self)
23606: (4)              m_internal = property(_mupdf.FzShadeColorCache_m_internal_get,
_mupdf.FzShadeColorCache_m_internal_set, doc=r""" Pointer to wrapped data.""")
23607: (4)              s_num_instances = property(_mupdf.FzShadeColorCache_s_num_instances_get,
_mupdf.FzShadeColorCache_s_num_instances_set)
23608: (0)          _mupdf.FzShadeColorCache_swigregister(FzShadeColorCache)
23609: (0)          class FzShaperDataT(object):
23610: (4)              r"""
23611: (4)              Wrapper class for struct `fz_shaper_data_t`. Not copyable or assignable.
23612: (4)              In order to shape a given font, we need to
23613: (4)              declare it to a shaper library (harfbuzz, by default, but others
23614: (4)              are possible). To avoid redeclaring it every time we need to
23615: (4)              shape, we hold a shaper handle and the destructor for it within
23616: (4)              the font itself. The handle is initialised by the caller when
23617: (4)              first required and the destructor is called when the fz_font is
23618: (4)              destroyed.
23619: (4)              """
23620: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23621: (4)              __repr__ = _swig_repr
23622: (4)              def __init__(self, *args):
23623: (8)                  r"""
23624: (8)                  *Overload 1:*
23625: (8)                  Default constructor, sets `m_internal` to null.
23626: (8)                  |
23627: (8)                  *Overload 2:*
23628: (8)                  Constructor using raw copy of pre-existing `::fz_shaper_data_t`.
```

```
23629: (8)                  """
23630: (8)                  _mupdf.FzShaperDataT_swiginit(self, _mupdf.new_FzShaperDataT(*args))
23631: (4)              __swig_destroy__ = _mupdf.delete_FzShaperDataT
23632: (4)              def m_internal_value(self):
23633: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23634: (8)                  return _mupdf.FzShaperDataT_m_internal_value(self)
23635: (4)              m_internal = property(_mupdf.FzShaperDataT_m_internal_get,
_mupdf.FzShaperDataT_m_internal_set, doc=r""" Pointer to wrapped data.""")
23636: (4)              s_num_instances = property(_mupdf.FzShaperDataT_s_num_instances_get,
_mupdf.FzShaperDataT_s_num_instances_set)
23637: (0)          _mupdf.FzShaperDataT_swigregister(FzShaperDataT)
23638: (0)          class FzStextBlock(object):
23639: (4)              """ Wrapper class for struct `fz_stext_block`."""
23640: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23641: (4)              __repr__ = _swig_repr
23642: (4)              def i_transform(self):
23643: (8)                  r""" Returns m_internal.u.i.transform if m_internal->type is
FZ_STEXT_BLOCK_IMAGE, else throws."""
23644: (8)                  return _mupdf.FzStextBlock_i_transform(self)
23645: (4)              def i_image(self):
23646: (8)                  r""" Returns m_internal.u.i.image if m_internal->type is
FZ_STEXT_BLOCK_IMAGE, else throws."""
23647: (8)                  return _mupdf.FzStextBlock_i_image(self)
23648: (4)              def begin(self):
23649: (8)                  r""" Used for iteration over linked list of FzStextLine items starting
at fz_stext_line::u.t.first_line."""
23650: (8)                  return _mupdf.FzStextBlock_begin(self)
23651: (4)              def end(self):
23652: (8)                  r""" Used for iteration over linked list of FzStextLine items starting
at fz_stext_line::u.t.first_line."""
23653: (8)                  return _mupdf.FzStextBlock_end(self)
23654: (4)              def __init__(self, *args):
23655: (8)                  r"""
23656: (8)                  *Overload 1:*
23657: (8)                  We use default copy constructor and operator=.  Default constructor,
sets `m_internal` to null.
23658: (8)                  |
23659: (8)                  *Overload 2:*
23660: (8)                  Constructor using raw copy of pre-existing `::fz_stext_block`.
23661: (8)                  """
23662: (8)                  _mupdf.FzStextBlock_swiginit(self, _mupdf.new_FzStextBlock(*args))
23663: (4)              __swig_destroy__ = _mupdf.delete_FzStextBlock
23664: (4)              def m_internal_value(self):
23665: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23666: (8)                  return _mupdf.FzStextBlock_m_internal_value(self)
23667: (4)              m_internal = property(_mupdf.FzStextBlock_m_internal_get,
_mupdf.FzStextBlock_m_internal_set, doc=r""" Pointer to wrapped data.""")
23668: (4)              s_num_instances = property(_mupdf.FzStextBlock_s_num_instances_get,
_mupdf.FzStextBlock_s_num_instances_set)
23669: (0)          _mupdf.FzStextBlock_swigregister(FzStextBlock)
23670: (0)          class FzStextChar(object):
23671: (4)              """
23672: (4)              Wrapper class for struct `fz_stext_char`.
23673: (4)              Text extraction device: Used for searching, format conversion etc.
23674: (4)              (In development - Subject to change in future versions)
23675: (4)              """
23676: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23677: (4)              __repr__ = _swig_repr
23678: (4)              def __init__(self, *args):
23679: (8)                  r"""
23680: (8)                  *Overload 1:*
23681: (8)                  We use default copy constructor and operator=.  Default constructor,
sets `m_internal` to null.
23682: (8)                  |
23683: (8)                  *Overload 2:*
```

```
23684: (8)                  Constructor using raw copy of pre-existing `::fz_stext_char`.
23685: (8)                  """
23686: (8)                  _mupdf.FzStextChar_swiginit(self, _mupdf.new_FzStextChar(*args))
23687: (4)              __swig_destroy__ = _mupdf.delete_FzStextChar
23688: (4)              def m_internal_value(self):
23689: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23690: (8)                  return _mupdf.FzStextChar_m_internal_value(self)
23691: (4)              m_internal = property(_mupdf.FzStextChar_m_internal_get,
_mupdf.FzStextChar_m_internal_set, doc=r""" Pointer to wrapped data.""")
23692: (4)              s_num_instances = property(_mupdf.FzStextChar_s_num_instances_get,
_mupdf.FzStextChar_s_num_instances_set)
23693: (0)          _mupdf.FzStextChar_swigregister(FzStextChar)
23694: (0)          class FzStextLine(object):
23695: (4)              r""" Wrapper class for struct `fz_stext_line`."""
23696: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23697: (4)              __repr__ = _swig_repr
23698: (4)              def begin(self):
23699: (8)                  r""" Used for iteration over linked list of FzStextChar items starting
at fz_stext_char::first_char."""
23700: (8)                  return _mupdf.FzStextLine_begin(self)
23701: (4)              def end(self):
23702: (8)                  r""" Used for iteration over linked list of FzStextChar items starting
at fz_stext_char::first_char."""
23703: (8)                  return _mupdf.FzStextLine_end(self)
23704: (4)              def __init__(self, *args):
23705: (8)                  r"""
23706: (8)                  *Overload 1:*
23707: (8)                  We use default copy constructor and operator=.  Default constructor,
sets `m_internal` to null.
23708: (8)                  |
23709: (8)                  *Overload 2:*
23710: (8)                  Constructor using raw copy of pre-existing `::fz_stext_line`.
23711: (8)                  """
23712: (8)                  _mupdf.FzStextLine_swiginit(self, _mupdf.new_FzStextLine(*args))
23713: (4)              __swig_destroy__ = _mupdf.delete_FzStextLine
23714: (4)              def m_internal_value(self):
23715: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23716: (8)                  return _mupdf.FzStextLine_m_internal_value(self)
23717: (4)              m_internal = property(_mupdf.FzStextLine_m_internal_get,
_mupdf.FzStextLine_m_internal_set, doc=r""" Pointer to wrapped data.""")
23718: (4)              s_num_instances = property(_mupdf.FzStextLine_s_num_instances_get,
_mupdf.FzStextLine_s_num_instances_set)
23719: (0)          _mupdf.FzStextLine_swigregister(FzStextLine)
23720: (0)          class FzStextOptions(object):
23721: (4)              r"""
23722: (4)              Wrapper class for struct `fz_stext_options`.
23723: (4)              Options for creating structured text.
23724: (4)              """
23725: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23726: (4)              __repr__ = _swig_repr
23727: (4)              def fz_parse_stext_options(self, string):
23728: (8)                  r"""
23729: (8)                  Class-aware wrapper for `::fz_parse_stext_options()`.
23730: (12)                     Parse stext device options from a comma separated key-value
23731: (12)                     string.
23732: (8)                  """
23733: (8)                  return _mupdf.FzStextOptions_fz_parse_stext_options(self, string)
23734: (4)              def __init__(self, *args):
23735: (8)                  r"""
23736: (8)                  *Overload 1:*
23737: (8)                  Construct with .flags set to <flags>.
23738: (8)                  |
23739: (8)                  *Overload 2:*
23740: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
```

```
23741: (8)                  |
23742: (8)                  *Overload 3:*
23743: (8)                  Constructor using raw copy of pre-existing `::fz_stext_options`.
23744: (8)                  |
23745: (8)                  *Overload 4:*
23746: (8)                  Constructor using raw copy of pre-existing `::fz_stext_options`.
23747: (8)                  """
23748: (8)                  _mupdf.FzStextOptions_swiginit(self, _mupdf.new_FzStextOptions(*args))
23749: (4)              def internal(self, *args):
23750: (8)                  r"""
23751: (8)                  *Overload 1:*
23752: (8)                  Access as underlying struct.
23753: (8)                  |
23754: (8)                  *Overload 2:*
23755: (8)                  Access as underlying struct.
23756: (8)                  """
23757: (8)                  return _mupdf.FzStextOptions_internal(self, *args)
23758: (4)              __swig_destroy__ = _mupdf.delete_FzStextOptions
23759: (4)              flags = property(_mupdf.FzStextOptions_flags_get,
_mupdf.FzStextOptions_flags_set)
23760: (4)              scale = property(_mupdf.FzStextOptions_scale_get,
_mupdf.FzStextOptions_scale_set)
23761: (4)              s_num_instances = property(_mupdf.FzStextOptions_s_num_instances_get,
_mupdf.FzStextOptions_s_num_instances_set)
23762: (4)              def to_string(self):
23763: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
23764: (8)                  return _mupdf.FzStextOptions_to_string(self)
23765: (4)              def __eq__(self, rhs):
23766: (8)                  r""" Comparison method."""
23767: (8)                  return _mupdf.FzStextOptions___eq__(self, rhs)
23768: (4)              def __ne__(self, rhs):
23769: (8)                  r""" Comparison method."""
23770: (8)                  return _mupdf.FzStextOptions___ne__(self, rhs)
23771: (0)          _mupdf.FzStextOptions_swigregister(FzStextOptions)
23772: (0)          class FzStextPage(object):
23773: (4)              r"""
23774: (4)              Wrapper class for struct `fz_stext_page`. Not copyable or assignable.
23775: (4)              A text page is a list of blocks, together with an overall
23776: (4)              bounding box.
23777: (4)              """
23778: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23779: (4)              __repr__ = _swig_repr
23780: (4)              def fz_highlight_selection(self, a, b, quads, max_quads):
23781: (8)                  r"""
23782: (8)                  Class-aware wrapper for `::fz_highlight_selection()`.
23783: (12)                     Return a list of quads to highlight lines inside the selection
23784: (12)                     points.
23785: (8)                  """
23786: (8)                  return _mupdf.FzStextPage_fz_highlight_selection(self, a, b, quads,
max_quads)
23787: (4)              def fz_highlight_selection2(self, a, b, max_quads):
23788: (8)                  r"""
23789: (8)                  Class-aware wrapper for `::fz_highlight_selection2()`.
23790: (8)                  C++ alternative to fz_highlight_selection() that returns quads in a
23791: (8)                  std::vector.
23792: (8)                  """
23793: (8)                  return _mupdf.FzStextPage_fz_highlight_selection2(self, a, b,
max_quads)
23794: (4)              def fz_new_buffer_from_stext_page(self):
23795: (8)                  r"""
23796: (8)                  Class-aware wrapper for `::fz_new_buffer_from_stext_page()`.
23797: (12)                     Convert structured text into plain text.
23798: (8)                  """
23799: (8)                  return _mupdf.FzStextPage_fz_new_buffer_from_stext_page(self)
23800: (4)              def fz_new_stext_device(self, options):
23801: (8)                  r"""
23802: (8)                  Class-aware wrapper for `::fz_new_stext_device()`.
```

```
23803: (12)                     Create a device to extract the text on a page.
23804: (12)                     Gather the text on a page into blocks and lines.
23805: (12)                     The reading order is taken from the order the text is drawn in
23806: (12)                     the source file, so may not be accurate.
23807: (12)                     page: The text page to which content should be added. This will
23808: (12)                     usually be a newly created (empty) text page, but it can be one
23809: (12)                     containing data already (for example when merging multiple
23810: (12)                     pages, or watermarking).
23811: (12)                     options: Options to configure the stext device.
23812: (8)                  """
23813: (8)                  return _mupdf.FzStextPage_fz_new_stext_device(self, options)
23814: (4)              def fz_search_stext_page(self, needle, hit_mark, hit_bbox, hit_max):
23815: (8)                  r"""
23816: (8)                  Class-aware wrapper for `::fz_search_stext_page()`.
23817: (8)                  This method has out-params. Python/C# wrappers look like:
23818: (12)                     `fz_search_stext_page(const char *needle, ::fz_quad *hit_bbox, int
hit_max)` => `(int, int hit_mark)`
23819: (12)                     Search for occurrence of 'needle' in text page.
23820: (12)                     Return the number of hits and store hit quads in the passed in
23821: (12)                     array.
23822: (12)                     NOTE: This is an experimental interface and subject to change
23823: (12)                     without notice.
23824: (8)                  """
23825: (8)                  return _mupdf.FzStextPage_fz_search_stext_page(self, needle, hit_mark,
hit_bbox, hit_max)
23826: (4)              def fz_snap_selection(self, ap, bp, mode):
23827: (8)                  r""" Class-aware wrapper for `::fz_snap_selection()`."""
23828: (8)                  return _mupdf.FzStextPage_fz_snap_selection(self, ap, bp, mode)
23829: (4)              def fz_copy_selection(self, a, b, crlf):
23830: (8)                  r""" Wrapper for fz_copy_selection() that returns std::string."""
23831: (8)                  return _mupdf.FzStextPage_fz_copy_selection(self, a, b, crlf)
23832: (4)              def fz_copy_rectangle(self, area, crlf):
23833: (8)                  r""" Wrapper for fz_copy_rectangle() that returns a std::string."""
23834: (8)                  return _mupdf.FzStextPage_fz_copy_rectangle(self, area, crlf)
23835: (4)              def search_stext_page(self, needle, hit_mark, max_quads):
23836: (8)                  r""" Wrapper for fz_search_stext_page() that returns std::vector of
Quads."""
23837: (8)                  return _mupdf.FzStextPage_search_stext_page(self, needle, hit_mark,
max_quads)
23838: (4)              def begin(self):
23839: (8)                  r""" Used for iteration over linked list of FzStextBlock items
starting at fz_stext_block::first_block."""
23840: (8)                  return _mupdf.FzStextPage_begin(self)
23841: (4)              def end(self):
23842: (8)                  r""" Used for iteration over linked list of FzStextBlock items
starting at fz_stext_block::first_block."""
23843: (8)                  return _mupdf.FzStextPage_end(self)
23844: (4)              def __init__(self, *args):
23845: (8)                  r"""
23846: (8)                  *Overload 1:*
23847: (9)                   == Constructors.  Constructor using `fz_new_stext_page()`.
23848: (16)                         Create an empty text page.
23849: (16)                         The text page is filled out by the text device to contain the
23850: (16)                         blocks and lines of text on the page.
23851: (16)                         mediabox: optional mediabox information.
23852: (8)                  |
23853: (8)                  *Overload 2:*
23854: (9)                   Constructor using `fz_new_stext_page_from_chapter_page_number()`.
23855: (8)                  |
23856: (8)                  *Overload 3:*
23857: (9)                   Constructor using `fz_new_stext_page_from_display_list()`.
23858: (8)                  |
23859: (8)                  *Overload 4:*
23860: (9)                   Constructor using `fz_new_stext_page_from_page()`.
23861: (16)                         Extract text from page.
23862: (16)                         Ownership of the fz_stext_page is returned to the caller.
23863: (8)                  |
23864: (8)                  *Overload 5:*
23865: (9)                   Constructor using `fz_new_stext_page_from_page_number()`.
```

```
23866: (8)                  |
23867: (8)                  *Overload 6:*
23868: (9)                   Constructor using `pdf_new_stext_page_from_annot()`.
23869: (8)                  |
23870: (8)                  *Overload 7:*
23871: (9)                   Default constructor, sets `m_internal` to null.
23872: (8)                  |
23873: (8)                  *Overload 8:*
23874: (9)                   Constructor using raw copy of pre-existing `::fz_stext_page`.
23875: (8)                  """
23876: (8)                  _mupdf.FzStextPage_swiginit(self, _mupdf.new_FzStextPage(*args))
23877: (4)              __swig_destroy__ = _mupdf.delete_FzStextPage
23878: (4)              def m_internal_value(self):
23879: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23880: (8)                  return _mupdf.FzStextPage_m_internal_value(self)
23881: (4)              m_internal = property(_mupdf.FzStextPage_m_internal_get,
_mupdf.FzStextPage_m_internal_set, doc=r""" Pointer to wrapped data.""")
23882: (4)              s_num_instances = property(_mupdf.FzStextPage_s_num_instances_get,
_mupdf.FzStextPage_s_num_instances_set)
23883: (0)          _mupdf.FzStextPage_swigregister(FzStextPage)
23884: (0)          class FzStorable(object):
23885: (4)              r"""
23886: (4)              Wrapper class for struct `fz_storable`.
23887: (4)              Resource store
23888: (4)              MuPDF stores decoded "objects" into a store for potential reuse.
23889: (4)              If the size of the store gets too big, objects stored within it
23890: (4)              can be evicted and freed to recover space. When MuPDF comes to
23891: (4)              decode such an object, it will check to see if a version of this
23892: (4)              object is already in the store - if it is, it will simply reuse
23893: (4)              it. If not, it will decode it and place it into the store.
23894: (4)              All objects that can be placed into the store are derived from
23895: (4)              the fz_storable type (i.e. this should be the first component of
23896: (4)              the objects structure). This allows for consistent (thread safe)
23897: (4)              reference counting, and includes a function that will be called
23898: (4)              to free the object as soon as the reference count reaches zero.
23899: (4)              Most objects offer fz_keep_XXXX/fz_drop_XXXX functions derived
23900: (4)              from fz_keep_storable/fz_drop_storable. Creation of such objects
23901: (4)              includes a call to FZ_INIT_STORABLE to set up the fz_storable
23902: (4)              header.
23903: (4)              """
23904: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23905: (4)              __repr__ = _swig_repr
23906: (4)              def __init__(self, *args):
23907: (8)                  r"""
23908: (8)                  *Overload 1:*
23909: (8)                  Copy constructor using `fz_keep_storable()`.
23910: (8)                  |
23911: (8)                  *Overload 2:*
23912: (8)                  Default constructor, sets `m_internal` to null.
23913: (8)                  |
23914: (8)                  *Overload 3:*
23915: (8)                  Constructor using raw copy of pre-existing `::fz_storable`.
23916: (8)                  """
23917: (8)                  _mupdf.FzStorable_swiginit(self, _mupdf.new_FzStorable(*args))
23918: (4)              __swig_destroy__ = _mupdf.delete_FzStorable
23919: (4)              def m_internal_value(self):
23920: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23921: (8)                  return _mupdf.FzStorable_m_internal_value(self)
23922: (4)              m_internal = property(_mupdf.FzStorable_m_internal_get,
_mupdf.FzStorable_m_internal_set, doc=r""" Pointer to wrapped data.""")
23923: (4)              s_num_instances = property(_mupdf.FzStorable_s_num_instances_get,
_mupdf.FzStorable_s_num_instances_set)
23924: (0)          _mupdf.FzStorable_swigregister(FzStorable)
23925: (0)          class FzStore(object):
23926: (4)              r""" Wrapper class for struct `fz_store`. Not copyable or assignable."""
23927: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
23928: (4)              __repr__ = _swig_repr
23929: (4)              def __init__(self, *args):
23930: (8)                  r"""
23931: (8)                  *Overload 1:*
23932: (8)                  Default constructor, sets `m_internal` to null.
23933: (8)                  |
23934: (8)                  *Overload 2:*
23935: (8)                  Constructor using raw copy of pre-existing `::fz_store`.
23936: (8)                  """
23937: (8)                  _mupdf.FzStore_swiginit(self, _mupdf.new_FzStore(*args))
23938: (4)              __swig_destroy__ = _mupdf.delete_FzStore
23939: (4)              def m_internal_value(self):
23940: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
23941: (8)                  return _mupdf.FzStore_m_internal_value(self)
23942: (4)              m_internal = property(_mupdf.FzStore_m_internal_get,
_mupdf.FzStore_m_internal_set, doc=r""" Pointer to wrapped data.""")
23943: (4)              s_num_instances = property(_mupdf.FzStore_s_num_instances_get,
_mupdf.FzStore_s_num_instances_set)
23944: (0)          _mupdf.FzStore_swigregister(FzStore)
23945: (0)          class FzStoreHash(object):
23946: (4)              r"""
23947: (4)              Wrapper class for struct `fz_store_hash`. Not copyable or assignable.
23948: (4)              The store can be seen as a dictionary that maps keys to
23949: (4)              fz_storable values. In order to allow keys of different types to
23950: (4)              be stored, we have a structure full of functions for each key
23951: (4)              'type'; this fz_store_type pointer is stored with each key, and
23952: (4)              tells the store how to perform certain operations (like taking/
23953: (4)              dropping a reference, comparing two keys, outputting details for
23954: (4)              debugging etc).
23955: (4)              The store uses a hash table internally for speed where possible.
23956: (4)              In order for this to work, we need a mechanism for turning a
23957: (4)              generic 'key' into 'a hashable string'. For this purpose the
23958: (4)              type structure contains a make_hash_key function pointer that
23959: (4)              maps from a void * to a fz_store_hash structure. If
23960: (4)              make_hash_key function returns 0, then the key is determined not
23961: (4)              to be hashable, and the value is not stored in the hash table.
23962: (4)              Some objects can be used both as values within the store, and as
23963: (4)              a component of keys within the store. We refer to these objects
23964: (4)              as "key storable" objects. In this case, we need to take
23965: (4)              additional care to ensure that we do not end up keeping an item
23966: (4)              within the store, purely because its value is referred to by
23967: (4)              another key in the store.
23968: (4)              An example of this are fz_images in PDF files. Each fz_image is
23969: (4)              placed into the    store to enable it to be easily reused. When the
23970: (4)              image is rendered, a pixmap is generated from the image, and the
23971: (4)              pixmap is placed into the store so it can be reused on
23972: (4)              subsequent renders. The image forms part of the key for the
23973: (4)              pixmap.
23974: (4)              When we close the pdf document (and any associated pages/display
23975: (4)              lists etc), we drop the images from the store. This may leave us
23976: (4)              in the position of the images having non-zero reference counts
23977: (4)              purely because they are used as part of the keys for the
23978: (4)              pixmaps.
23979: (4)              We therefore use special reference counting functions to keep
23980: (4)              track of these "key storable" items, and hence store the number
23981: (4)              of references to these items that are used in keys.
23982: (4)              When the number of references to an object == the number of
23983: (4)              references to an object from keys in the store, we know that we
23984: (4)              can remove all the items which have that object as part of the
23985: (4)              key. This is done by running a pass over the store, 'reaping'
23986: (4)              those items.
23987: (4)              Reap passes are slower than we would like as they touch every
23988: (4)              item in the store. We therefore provide a way to 'batch' such
23989: (4)              reap passes together, using fz_defer_reap_start/
23990: (4)              fz_defer_reap_end to bracket a region in which many may be
23991: (4)              triggered.
23992: (4)              """
```

```
23993: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
23994: (4)              __repr__ = _swig_repr
23995: (4)              def __init__(self, *args):
23996: (8)                  r"""
23997: (8)                  *Overload 1:*
23998: (8)                  Default constructor, sets `m_internal` to null.
23999: (8)                  |
24000: (8)                  *Overload 2:*
24001: (8)                  Constructor using raw copy of pre-existing `::fz_store_hash`.
24002: (8)                  """
24003: (8)                  _mupdf.FzStoreHash_swiginit(self, _mupdf.new_FzStoreHash(*args))
24004: (4)              __swig_destroy__ = _mupdf.delete_FzStoreHash
24005: (4)              def m_internal_value(self):
24006: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24007: (8)                  return _mupdf.FzStoreHash_m_internal_value(self)
24008: (4)              m_internal = property(_mupdf.FzStoreHash_m_internal_get,
_mupdf.FzStoreHash_m_internal_set, doc=r""" Pointer to wrapped data.""")
24009: (4)              s_num_instances = property(_mupdf.FzStoreHash_s_num_instances_get,
_mupdf.FzStoreHash_s_num_instances_set)
24010: (0)          _mupdf.FzStoreHash_swigregister(FzStoreHash)
24011: (0)          class FzStoreType(object):
24012: (4)              r"""
24013: (4)              Wrapper class for struct `fz_store_type`. Not copyable or assignable.
24014: (4)              Every type of object to be placed into the store defines an
24015: (4)              fz_store_type. This contains the pointers to functions to
24016: (4)              make hashes, manipulate keys, and check for needing reaping.
24017: (4)              """
24018: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24019: (4)              __repr__ = _swig_repr
24020: (4)              def __init__(self, *args):
24021: (8)                  r"""
24022: (8)                  *Overload 1:*
24023: (8)                  Default constructor, sets `m_internal` to null.
24024: (8)                  |
24025: (8)                  *Overload 2:*
24026: (8)                  Constructor using raw copy of pre-existing `::fz_store_type`.
24027: (8)                  """
24028: (8)                  _mupdf.FzStoreType_swiginit(self, _mupdf.new_FzStoreType(*args))
24029: (4)              __swig_destroy__ = _mupdf.delete_FzStoreType
24030: (4)              def m_internal_value(self):
24031: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24032: (8)                  return _mupdf.FzStoreType_m_internal_value(self)
24033: (4)              m_internal = property(_mupdf.FzStoreType_m_internal_get,
_mupdf.FzStoreType_m_internal_set, doc=r""" Pointer to wrapped data.""")
24034: (4)              s_num_instances = property(_mupdf.FzStoreType_s_num_instances_get,
_mupdf.FzStoreType_s_num_instances_set)
24035: (0)          _mupdf.FzStoreType_swigregister(FzStoreType)
24036: (0)          class FzStory(object):
24037: (4)              r""" Wrapper class for struct `fz_story`. Not copyable or assignable."""
24038: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24039: (4)              __repr__ = _swig_repr
24040: (4)              def fz_draw_story(self, dev, ctm):
24041: (8)                  r""" Class-aware wrapper for `::fz_draw_story()`."""
24042: (8)                  return _mupdf.FzStory_fz_draw_story(self, dev, ctm)
24043: (4)              def fz_place_story(self, where, filled):
24044: (8)                  r""" Class-aware wrapper for `::fz_place_story()`."""
24045: (8)                  return _mupdf.FzStory_fz_place_story(self, where, filled)
24046: (4)              def fz_place_story_flags(self, where, filled, flags):
24047: (8)                  r""" Class-aware wrapper for `::fz_place_story_flags()`."""
24048: (8)                  return _mupdf.FzStory_fz_place_story_flags(self, where, filled, flags)
24049: (4)              def fz_reset_story(self):
24050: (8)                  r""" Class-aware wrapper for `::fz_reset_story()`."""
24051: (8)                  return _mupdf.FzStory_fz_reset_story(self)
24052: (4)              def fz_story_document(self):
```

```
24053: (8)                  r""" Class-aware wrapper for `::fz_story_document()`."""
24054: (8)                  return _mupdf.FzStory_fz_story_document(self)
24055: (4)              def fz_story_positions(self, cb, arg):
24056: (8)                  r""" Class-aware wrapper for `::fz_story_positions()`."""
24057: (8)                  return _mupdf.FzStory_fz_story_positions(self, cb, arg)
24058: (4)              def fz_story_warnings(self):
24059: (8)                  r""" Class-aware wrapper for `::fz_story_warnings()`."""
24060: (8)                  return _mupdf.FzStory_fz_story_warnings(self)
24061: (4)              def __init__(self, *args):
24062: (8)                  r"""
24063: (8)                  *Overload 1:*
24064: (8)                  == Constructors.  Constructor using `fz_new_story()`.
24065: (8)                  |
24066: (8)                  *Overload 2:*
24067: (8)                  Default constructor, sets `m_internal` to null.
24068: (8)                  |
24069: (8)                  *Overload 3:*
24070: (8)                  Constructor using raw copy of pre-existing `::fz_story`.
24071: (8)                  """
24072: (8)                  _mupdf.FzStory_swiginit(self, _mupdf.new_FzStory(*args))
24073: (4)              __swig_destroy__ = _mupdf.delete_FzStory
24074: (4)              def m_internal_value(self):
24075: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24076: (8)                  return _mupdf.FzStory_m_internal_value(self)
24077: (4)              m_internal = property(_mupdf.FzStory_m_internal_get,
_mupdf.FzStory_m_internal_set, doc=r""" Pointer to wrapped data.""")
24078: (4)              s_num_instances = property(_mupdf.FzStory_s_num_instances_get,
_mupdf.FzStory_s_num_instances_set)
24079: (0)          _mupdf.FzStory_swigregister(FzStory)
24080: (0)          class FzStoryElementPosition(object):
24081: (4)              r""" Wrapper class for struct `fz_story_element_position`."""
24082: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24083: (4)              __repr__ = _swig_repr
24084: (4)              def __init__(self, *args):
24085: (8)                  r"""
24086: (8)                  *Overload 1:*
24087: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
24088: (8)                  |
24089: (8)                  *Overload 2:*
24090: (8)                  Constructor using raw copy of pre-existing
`::fz_story_element_position`.
24091: (8)                  |
24092: (8)                  *Overload 3:*
24093: (8)                  Constructor using raw copy of pre-existing
`::fz_story_element_position`.
24094: (8)                  """
24095: (8)                  _mupdf.FzStoryElementPosition_swiginit(self,
_mupdf.new_FzStoryElementPosition(*args))
24096: (4)              def internal(self, *args):
24097: (8)                  r"""
24098: (8)                  *Overload 1:*
24099: (8)                  Access as underlying struct.
24100: (8)                  |
24101: (8)                  *Overload 2:*
24102: (8)                  Access as underlying struct.
24103: (8)                  """
24104: (8)                  return _mupdf.FzStoryElementPosition_internal(self, *args)
24105: (4)              __swig_destroy__ = _mupdf.delete_FzStoryElementPosition
24106: (4)              depth = property(_mupdf.FzStoryElementPosition_depth_get,
_mupdf.FzStoryElementPosition_depth_set)
24107: (4)              heading = property(_mupdf.FzStoryElementPosition_heading_get,
_mupdf.FzStoryElementPosition_heading_set)
24108: (4)              id = property(_mupdf.FzStoryElementPosition_id_get,
_mupdf.FzStoryElementPosition_id_set)
24109: (4)              href = property(_mupdf.FzStoryElementPosition_href_get,
_mupdf.FzStoryElementPosition_href_set)
```

```
24110: (4)              rect = property(_mupdf.FzStoryElementPosition_rect_get,
_mupdf.FzStoryElementPosition_rect_set)
24111: (4)              text = property(_mupdf.FzStoryElementPosition_text_get,
_mupdf.FzStoryElementPosition_text_set)
24112: (4)              open_close = property(_mupdf.FzStoryElementPosition_open_close_get,
_mupdf.FzStoryElementPosition_open_close_set)
24113: (4)              rectangle_num = property(_mupdf.FzStoryElementPosition_rectangle_num_get,
_mupdf.FzStoryElementPosition_rectangle_num_set)
24114: (4)              s_num_instances =
property(_mupdf.FzStoryElementPosition_s_num_instances_get,
_mupdf.FzStoryElementPosition_s_num_instances_set)
24115: (4)              def to_string(self):
24116: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
24117: (8)                  return _mupdf.FzStoryElementPosition_to_string(self)
24118: (4)              def __eq__(self, rhs):
24119: (8)                  r""" Comparison method."""
24120: (8)                  return _mupdf.FzStoryElementPosition___eq__(self, rhs)
24121: (4)              def __ne__(self, rhs):
24122: (8)                  r""" Comparison method."""
24123: (8)                  return _mupdf.FzStoryElementPosition___ne__(self, rhs)
24124: (0)          _mupdf.FzStoryElementPosition_swigregister(FzStoryElementPosition)
24125: (0)          class FzStream(object):
24126: (4)              r"""
24127: (4)              Wrapper class for struct `fz_stream`.
24128: (4)              fz_stream is a buffered reader capable of seeking in both
24129: (4)              directions.
24130: (4)              Streams are reference counted, so references must be dropped
24131: (4)              by a call to fz_drop_stream.
24132: (4)              Only the data between rp and wp is valid.
24133: (4)              """
24134: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24135: (4)              __repr__ = _swig_repr
24136: (4)              @staticmethod
24137: (4)              def fz_open_file_autodelete(filename):
24138: (8)                  r"""
24139: (8)                  Class-aware wrapper for `::fz_open_file_autodelete()`.
24140: (12)                     Do the same as fz_open_file, but delete the file upon close.
24141: (8)                  """
24142: (8)                  return _mupdf.FzStream_fz_open_file_autodelete(filename)
24143: (4)              def fz_available(self, max):
24144: (8)                  r"""
24145: (8)                  Class-aware wrapper for `::fz_available()`.
24146: (12)                     Ask how many bytes are available immediately from
24147: (12)                     a given stream.
24148: (12)                     stm: The stream to read from.
24149: (12)                     max: A hint for the underlying stream; the maximum number of
24150: (12)                     bytes that we are sure we will want to read. If you do not know
24151: (12)                     this number, give 1.
24152: (12)                     Returns the number of bytes immediately available between the
24153: (12)                     read and write pointers. This number is guaranteed only to be 0
24154: (12)                     if we have hit EOF. The number of bytes returned here need have
24155: (12)                     no relation to max (could be larger, could be smaller).
24156: (8)                  """
24157: (8)                  return _mupdf.FzStream_fz_available(self, max)
24158: (4)              def fz_decomp_image_from_stream(self, image, subarea, indexed, l2factor,
l2extra):
24159: (8)                  r"""
24160: (8)                  Class-aware wrapper for `::fz_decomp_image_from_stream()`.
24161: (8)                  This method has out-params. Python/C# wrappers look like:
24162: (12)                     `fz_decomp_image_from_stream(::fz_compressed_image *image,
::fz_irect *subarea, int indexed, int l2factor)` => `(fz_pixmap *, int l2extra)`
24163: (12)                     Decode a subarea of a compressed image. l2factor is the amount
24164: (12)                     of subsampling inbuilt to the stream (i.e. performed by the
24165: (12)                     decoder). If non NULL, l2extra is the extra amount of
24166: (12)                     subsampling that should be performed by this routine. This will
24167: (12)                     be updated on exit to the amount of subsampling that is still
24168: (12)                     required to be done.
```

```
24169: (12)                     Returns a kept reference.
24170: (8)                  """
24171: (8)                  return _mupdf.FzStream_fz_decomp_image_from_stream(self, image,
subarea, indexed, l2factor, l2extra)
24172: (4)              def fz_is_cfb_archive(self):
24173: (8)                  r"""
24174: (8)                  Class-aware wrapper for `::fz_is_cfb_archive()`.
24175: (12)                     Detect if stream object is a cfb archive.
24176: (12)                     Assumes that the stream object is seekable.
24177: (8)                  """
24178: (8)                  return _mupdf.FzStream_fz_is_cfb_archive(self)
24179: (4)              def fz_is_eof(self):
24180: (8)                  r"""
24181: (8)                  Class-aware wrapper for `::fz_is_eof()`.
24182: (12)                     Query if the stream has reached EOF (during normal bytewise
24183: (12)                     reading).
24184: (12)                     See fz_is_eof_bits for the equivalent function for bitwise
24185: (12)                     reading.
24186: (8)                  """
24187: (8)                  return _mupdf.FzStream_fz_is_eof(self)
24188: (4)              def fz_is_eof_bits(self):
24189: (8)                  r"""
24190: (8)                  Class-aware wrapper for `::fz_is_eof_bits()`.
24191: (12)                     Query if the stream has reached EOF (during bitwise
24192: (12)                     reading).
24193: (12)                     See fz_is_eof for the equivalent function for bytewise
24194: (12)                     reading.
24195: (8)                  """
24196: (8)                  return _mupdf.FzStream_fz_is_eof_bits(self)
24197: (4)              def fz_is_libarchive_archive(self):
24198: (8)                  r"""
24199: (8)                  Class-aware wrapper for `::fz_is_libarchive_archive()`.
24200: (12)                     Detect if stream object is an archive supported by libarchive.
24201: (12)                     Assumes that the stream object is seekable.
24202: (8)                  """
24203: (8)                  return _mupdf.FzStream_fz_is_libarchive_archive(self)
24204: (4)              def fz_is_tar_archive(self):
24205: (8)                  r"""
24206: (8)                  Class-aware wrapper for `::fz_is_tar_archive()`.
24207: (12)                     Detect if stream object is a tar archive.
24208: (12)                     Assumes that the stream object is seekable.
24209: (8)                  """
24210: (8)                  return _mupdf.FzStream_fz_is_tar_archive(self)
24211: (4)              def fz_is_zip_archive(self):
24212: (8)                  r"""
24213: (8)                  Class-aware wrapper for `::fz_is_zip_archive()`.
24214: (12)                     Detect if stream object is a zip archive.
24215: (12)                     Assumes that the stream object is seekable.
24216: (8)                  """
24217: (8)                  return _mupdf.FzStream_fz_is_zip_archive(self)
24218: (4)              def fz_new_archive_of_size(self, size):
24219: (8)                  r""" Class-aware wrapper for `::fz_new_archive_of_size()`."""
24220: (8)                  return _mupdf.FzStream_fz_new_archive_of_size(self, size)
24221: (4)              def fz_open_a85d(self):
24222: (8)                  r"""
24223: (8)                  Class-aware wrapper for `::fz_open_a85d()`.
24224: (12)                     a85d filter performs ASCII 85 Decoding of data read
24225: (12)                     from the chained filter.
24226: (8)                  """
24227: (8)                  return _mupdf.FzStream_fz_open_a85d(self)
24228: (4)              def fz_open_aesd(self, key, keylen):
24229: (8)                  r"""
24230: (8)                  Class-aware wrapper for `::fz_open_aesd()`.
24231: (12)                     aesd filter performs AES decoding of data read from the chained
24232: (12)                     filter using the supplied key.
24233: (8)                  """
24234: (8)                  return _mupdf.FzStream_fz_open_aesd(self, key, keylen)
24235: (4)              def fz_open_ahxd(self):
24236: (8)                  r"""
```

```
(8)                 Class-aware wrapper for `::fz_open_ahxd()`.
(12)                    ahxd filter performs ASCII Hex decoding of data read
(12)                    from the chained filter.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_ahxd(self)
(4)             def fz_open_arc4(self, key, keylen):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_arc4()`.
(12)                    arc4 filter performs RC4 decoding of data read from the chained
(12)                    filter using the supplied key.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_arc4(self, key, keylen)
(4)             def fz_open_archive_with_stream(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_archive_with_stream()`.
(12)                    Open zip or tar archive stream.
(12)                    Open an archive using a seekable stream object rather than
(12)                    opening a file or directory on disk.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_archive_with_stream(self)
(4)             def fz_open_cfb_archive_with_stream(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_cfb_archive_with_stream()`.
(12)                    Open a cfb file as an archive.
(12)                    Open an archive using a seekable stream object rather than
(12)                    opening a file or directory on disk.
(12)                    An exception is thrown if the file is not recognised as a chm.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_cfb_archive_with_stream(self)
(4)             def fz_open_dctd(self, color_transform, invert_cmyk, l2factor,
jpegtables):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_dctd()`.
(12)                    dctd filter performs DCT (JPEG) decoding of data read
(12)                    from the chained filter.
(12)                    color_transform implements the PDF color_transform option
(16)                        use -1 for default behavior
(16)                        use 0 to disable YUV-RGB / YCCK-CMYK transforms
(16)                        use 1 to enable YUV-RGB / YCCK-CMYK transforms
(12)                    invert_cmyk implements the necessary inversion for Photoshop CMYK
images
(16)                        use 0 if embedded in PDF
(16)                        use 1 if not embedded in PDF
(12)                    For subsampling on decode, set l2factor to the log2 of the
(12)                    reduction required (therefore 0 = full size decode).
(12)                    jpegtables is an optional stream from which the JPEG tables
(12)                    can be read. Use NULL if not required.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_dctd(self, color_transform,
invert_cmyk, l2factor, jpegtables)
(4)             def fz_open_endstream_filter(self, len, offset):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_endstream_filter()`.
(12)                    The endstream filter reads a PDF substream, and starts to look
(12)                    for an 'endstream' token after the specified length.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_endstream_filter(self, len, offset)
(4)             def fz_open_faxd(self, k, end_of_line, encoded_byte_align, columns, rows,
end_of_block, black_is_1):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_faxd()`.
(12)                    faxd filter performs FAX decoding of data read from
(12)                    the chained filter.
(12)                    k: see fax specification (fax default is 0).
(12)                    end_of_line: whether we expect end of line markers (fax default
(12)                    is 0).
(12)                    encoded_byte_align: whether we align to bytes after each line
(12)                    (fax default is 0).
(12)                    columns: how many columns in the image (fax default is 1728).
```

```
24302: (12)                     rows: 0 for unspecified or the number of rows of data to expect.
24303: (12)                     end_of_block: whether we expect end of block markers (fax
24304: (12)                     default is 1).
24305: (12)                     black_is_1: determines the polarity of the image (fax default is
24306: (12)                     0).
24307: (8)                  """
24308: (8)                  return _mupdf.FzStream_fz_open_faxd(self, k, end_of_line,
encoded_byte_align, columns, rows, end_of_block, black_is_1)
24309: (4)              def fz_open_flated(self, window_bits):
24310: (8)                  r"""
24311: (8)                  Class-aware wrapper for `::fz_open_flated()`.
24312: (12)                     flated filter performs LZ77 decoding (inflating) of data read
24313: (12)                     from the chained filter.
24314: (12)                     window_bits: How large a decompression window to use. Typically
24315: (12)                     15. A negative number, -n, means to use n bits, but to expect
24316: (12)                     raw data with no header.
24317: (8)                  """
24318: (8)                  return _mupdf.FzStream_fz_open_flated(self, window_bits)
24319: (4)              def fz_open_image_decomp_stream(self, arg_1, l2factor):
24320: (8)                  r"""
24321: (8)                  Class-aware wrapper for `::fz_open_image_decomp_stream()`.
24322: (8)                  This method has out-params. Python/C# wrappers look like:
24323: (12)                     `fz_open_image_decomp_stream(::fz_compression_params *arg_1)` =>
`(fz_stream *, int l2factor)`
24324: (12)                     Open a stream to read the decompressed version of another stream
24325: (12)                     with optional log2 subsampling.
24326: (8)                  """
24327: (8)                  return _mupdf.FzStream_fz_open_image_decomp_stream(self, arg_1,
l2factor)
24328: (4)              def fz_open_jbig2d(self, globals, embedded):
24329: (8)                  r"""
24330: (8)                  Class-aware wrapper for `::fz_open_jbig2d()`.
24331: (12)                     Open a filter that performs jbig2 decompression on the chained
24332: (12)                     stream, using the optional globals record.
24333: (8)                  """
24334: (8)                  return _mupdf.FzStream_fz_open_jbig2d(self, globals, embedded)
24335: (4)              def fz_open_leecher(self, buf):
24336: (8)                  r"""
24337: (8)                  Class-aware wrapper for `::fz_open_leecher()`.
24338: (12)                     Attach a filter to a stream that will store any
24339: (12)                     characters read from the stream into the supplied buffer.
24340: (12)                     chain: The underlying stream to leech from.
24341: (12)                     buf: The buffer into which the read data should be appended.
24342: (12)                     The buffer will be resized as required.
24343: (12)                     Returns pointer to newly created stream. May throw exceptions on
24344: (12)                     failure to allocate.
24345: (8)                  """
24346: (8)                  return _mupdf.FzStream_fz_open_leecher(self, buf)
24347: (4)              def fz_open_libarchive_archive_with_stream(self):
24348: (8)                  r"""
24349: (8)                  Class-aware wrapper for `::fz_open_libarchive_archive_with_stream()`.
24350: (12)                     Open an archive using libarchive.
24351: (12)                     Open an archive using a seekable stream object rather than
24352: (12)                     opening a file or directory on disk.
24353: (12)                     An exception is thrown if the stream is not supported by
libarchive.
24354: (8)                  """
24355: (8)                  return _mupdf.FzStream_fz_open_libarchive_archive_with_stream(self)
24356: (4)              def fz_open_libarchived(self):
24357: (8)                  r"""
24358: (8)                  Class-aware wrapper for `::fz_open_libarchived()`.
24359: (12)                     libarchived filter performs generic compressed decoding of data
24360: (12)                     in any format understood by libarchive from the chained filter.
24361: (12)                     This will throw an exception if libarchive is not built in, or
24362: (12)                     if the compression format is not recognised.
24363: (8)                  """
24364: (8)                  return _mupdf.FzStream_fz_open_libarchived(self)
24365: (4)              def fz_open_lzwd(self, early_change, min_bits, reverse_bits, old_tiff):
24366: (8)                  r"""
```

```
(8)                 Class-aware wrapper for `::fz_open_lzwd()`.
(12)                    lzwd filter performs LZW decoding of data read from the chained
(12)                    filter.
(12)                    early_change: (Default 1) specifies whether to change codes 1
(12)                    bit early.
(12)                    min_bits: (Default 9) specifies the minimum number of bits to
(12)                    use.
(12)                    reverse_bits: (Default 0) allows for compatibility with gif and
(12)                    old style tiffs (1).
(12)                    old_tiff: (Default 0) allows for different handling of the clear
(12)                    code, as found in old style tiffs.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_lzwd(self, early_change, min_bits,
reverse_bits, old_tiff)
(4)             def fz_open_null_filter(self, len, offset):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_null_filter()`.
(12)                    The null filter reads a specified amount of data from the
(12)                    substream.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_null_filter(self, len, offset)
(4)             def fz_open_predict(self, predictor, columns, colors, bpc):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_predict()`.
(12)                    predict filter performs pixel prediction on data read from
(12)                    the chained filter.
(12)                    predictor: 1 = copy, 2 = tiff, other = inline PNG predictor
(12)                    columns: width of image in pixels
(12)                    colors: number of components.
(12)                    bpc: bits per component (typically 8)
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_predict(self, predictor, columns,
colors, bpc)
(4)             def fz_open_range_filter(self, ranges, nranges):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_range_filter()`.
(12)                    The range filter copies data from specified ranges of the
(12)                    chained stream.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_range_filter(self, ranges, nranges)
(4)             def fz_open_rld(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_rld()`.
(12)                    rld filter performs Run Length Decoding of data read
(12)                    from the chained filter.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_rld(self)
(4)             def fz_open_sgilog16(self, w):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_sgilog16()`.
(12)                    SGI Log 16bit (greyscale) decode from the chained filter.
(12)                    Decodes lines of w pixels to 8bpp greyscale.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_sgilog16(self, w)
(4)             def fz_open_sgilog24(self, w):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_sgilog24()`.
(12)                    SGI Log 24bit (LUV) decode from the chained filter.
(12)                    Decodes lines of w pixels to 8bpc rgb.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_sgilog24(self, w)
(4)             def fz_open_sgilog32(self, w):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_sgilog32()`.
(12)                    SGI Log 32bit (LUV) decode from the chained filter.
(12)                    Decodes lines of w pixels to 8bpc rgb.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_sgilog32(self, w)
(4)             def fz_open_tar_archive_with_stream(self):
```

```
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_tar_archive_with_stream()`.
(12)                    Open a tar archive stream.
(12)                    Open an archive using a seekable stream object rather than
(12)                    opening a file or directory on disk.
(12)                    An exception is thrown if the stream is not a tar archive as
(12)                    indicated by the presence of a tar signature.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_tar_archive_with_stream(self)
(4)             def fz_open_thunder(self, w):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_thunder()`.
(12)                    4bit greyscale Thunderscan decoding from the chained filter.
(12)                    Decodes lines of w pixels to 8bpp greyscale.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_thunder(self, w)
(4)             def fz_open_zip_archive_with_stream(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_open_zip_archive_with_stream()`.
(12)                    Open a zip archive stream.
(12)                    Open an archive using a seekable stream object rather than
(12)                    opening a file or directory on disk.
(12)                    An exception is thrown if the stream is not a zip archive as
(12)                    indicated by the presence of a zip signature.
(8)                 """
(8)                 return _mupdf.FzStream_fz_open_zip_archive_with_stream(self)
(4)             def fz_parse_xml_stream(self, preserve_white):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_parse_xml_stream()`.
(12)                    Parse the contents of buffer into a tree of xml nodes.
(12)                    preserve_white: whether to keep or delete all-whitespace nodes.
(8)                 """
(8)                 return _mupdf.FzStream_fz_parse_xml_stream(self, preserve_white)
(4)             def fz_peek_byte(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_peek_byte()`.
(12)                    Peek at the next byte in a stream.
(12)                    stm: The stream to peek at.
(12)                    Returns -1 for EOF, or the next byte that will be read.
(8)                 """
(8)                 return _mupdf.FzStream_fz_peek_byte(self)
(4)             def fz_read(self, data, len):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_read()`.
(12)                    Read from a stream into a given data block.
(12)                    stm: The stream to read from.
(12)                    data: The data block to read into.
(12)                    len: The length of the data block (in bytes).
(12)                    Returns the number of bytes read. May throw exceptions.
(8)                 """
(8)                 return _mupdf.FzStream_fz_read(self, data, len)
(4)             def fz_read_all(self, initial):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_read_all()`.
(12)                    Read all of a stream into a buffer.
(12)                    stm: The stream to read from
(12)                    initial: Suggested initial size for the buffer.
(12)                    Returns a buffer created from reading from the stream. May throw
(12)                    exceptions on failure to allocate.
(8)                 """
(8)                 return _mupdf.FzStream_fz_read_all(self, initial)
(4)             def fz_read_best(self, initial, truncated, worst_case):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_read_best()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_read_best(size_t initial, size_t worst_case)` => `(fz_buffer
*, int truncated)`
(12)                    Attempt to read a stream into a buffer. If truncated
(12)                    is NULL behaves as fz_read_all, sets a truncated flag in case of
```

```
24502: (12)                     error.
24503: (12)                     stm: The stream to read from.
24504: (12)                     initial: Suggested initial size for the buffer.
24505: (12)                     truncated: Flag to store success/failure indication in.
24506: (12)                     worst_case: 0 for unknown, otherwise an upper bound for the
24507: (12)                     size of the stream.
24508: (12)                     Returns a buffer created from reading from the stream.
24509: (8)                  """
24510: (8)                  return _mupdf.FzStream_fz_read_best(self, initial, truncated,
worst_case)
24511: (4)              def fz_read_bits(self, n):
24512: (8)                  r"""
24513: (8)                  Class-aware wrapper for `::fz_read_bits()`.
24514: (12)                     Read the next n bits from a stream (assumed to
24515: (12)                     be packed most significant bit first).
24516: (12)                     stm: The stream to read from.
24517: (12)                     n: The number of bits to read, between 1 and 8*sizeof(int)
24518: (12)                     inclusive.
24519: (12)                     Returns -1 for EOF, or the required number of bits.
24520: (8)                  """
24521: (8)                  return _mupdf.FzStream_fz_read_bits(self, n)
24522: (4)              def fz_read_byte(self):
24523: (8)                  r"""
24524: (8)                  Class-aware wrapper for `::fz_read_byte()`.
24525: (12)                     Read the next byte from a stream.
24526: (12)                     stm: The stream t read from.
24527: (12)                     Returns -1 for end of stream, or the next byte. May
24528: (12)                     throw exceptions.
24529: (8)                  """
24530: (8)                  return _mupdf.FzStream_fz_read_byte(self)
24531: (4)              def fz_read_float(self):
24532: (8)                  r""" Class-aware wrapper for `::fz_read_float()`."""
24533: (8)                  return _mupdf.FzStream_fz_read_float(self)
24534: (4)              def fz_read_float_le(self):
24535: (8)                  r""" Class-aware wrapper for `::fz_read_float_le()`."""
24536: (8)                  return _mupdf.FzStream_fz_read_float_le(self)
24537: (4)              def fz_read_int16(self):
24538: (8)                  r""" Class-aware wrapper for `::fz_read_int16()`."""
24539: (8)                  return _mupdf.FzStream_fz_read_int16(self)
24540: (4)              def fz_read_int16_le(self):
24541: (8)                  r""" Class-aware wrapper for `::fz_read_int16_le()`."""
24542: (8)                  return _mupdf.FzStream_fz_read_int16_le(self)
24543: (4)              def fz_read_int32(self):
24544: (8)                  r""" Class-aware wrapper for `::fz_read_int32()`."""
24545: (8)                  return _mupdf.FzStream_fz_read_int32(self)
24546: (4)              def fz_read_int32_le(self):
24547: (8)                  r""" Class-aware wrapper for `::fz_read_int32_le()`."""
24548: (8)                  return _mupdf.FzStream_fz_read_int32_le(self)
24549: (4)              def fz_read_int64(self):
24550: (8)                  r""" Class-aware wrapper for `::fz_read_int64()`."""
24551: (8)                  return _mupdf.FzStream_fz_read_int64(self)
24552: (4)              def fz_read_int64_le(self):
24553: (8)                  r""" Class-aware wrapper for `::fz_read_int64_le()`."""
24554: (8)                  return _mupdf.FzStream_fz_read_int64_le(self)
24555: (4)              def fz_read_line(self, buf, max):
24556: (8)                  r"""
24557: (8)                  Class-aware wrapper for `::fz_read_line()`.
24558: (12)                     Read a line from stream into the buffer until either a
24559: (12)                     terminating newline or EOF, which it replaces with a null byte
24560: (12)                     ('').
24561: (12)                     Returns buf on success, and NULL when end of file occurs while
24562: (12)                     no characters have been read.
24563: (8)                  """
24564: (8)                  return _mupdf.FzStream_fz_read_line(self, buf, max)
24565: (4)              def fz_read_rbits(self, n):
24566: (8)                  r"""
24567: (8)                  Class-aware wrapper for `::fz_read_rbits()`.
24568: (12)                     Read the next n bits from a stream (assumed to
24569: (12)                     be packed least significant bit first).
```

```
24570: (12)                     stm: The stream to read from.
24571: (12)                     n: The number of bits to read, between 1 and 8*sizeof(int)
24572: (12)                     inclusive.
24573: (12)                     Returns (unsigned int)-1 for EOF, or the required number of bits.
24574: (8)                  """
24575: (8)                  return _mupdf.FzStream_fz_read_rbits(self, n)
24576: (4)              def fz_read_rune(self):
24577: (8)                  r"""
24578: (8)                  Class-aware wrapper for `::fz_read_rune()`.
24579: (12)                     Read a utf-8 rune from a stream.
24580: (12)                     In the event of encountering badly formatted utf-8 codes
24581: (12)                     (such as a leading code with an unexpected number of following
24582: (12)                     codes) no error/exception is given, but undefined values may be
24583: (12)                     returned.
24584: (8)                  """
24585: (8)                  return _mupdf.FzStream_fz_read_rune(self)
24586: (4)              def fz_read_string(self, buffer, len):
24587: (8)                  r"""
24588: (8)                  Class-aware wrapper for `::fz_read_string()`.
24589: (12)                     Read a null terminated string from the stream into
24590: (12)                     a buffer of a given length. The buffer will be null terminated.
24591: (12)                     Throws on failure (including the failure to fit the entire
24592: (12)                     string including the terminator into the buffer).
24593: (8)                  """
24594: (8)                  return _mupdf.FzStream_fz_read_string(self, buffer, len)
24595: (4)              def fz_read_uint16(self):
24596: (8)                  r"""
24597: (8)                  Class-aware wrapper for `::fz_read_uint16()`.
24598: (12)                     fz_read_[u]int(16|24|32|64)(_le)?
24599: (12)                     Read a 16/32/64 bit signed/unsigned integer from stream,
24600: (12)                     in big or little-endian byte orders.
24601: (12)                     Throws an exception if EOF is encountered.
24602: (8)                  """
24603: (8)                  return _mupdf.FzStream_fz_read_uint16(self)
24604: (4)              def fz_read_uint16_le(self):
24605: (8)                  r""" Class-aware wrapper for `::fz_read_uint16_le()`."""
24606: (8)                  return _mupdf.FzStream_fz_read_uint16_le(self)
24607: (4)              def fz_read_uint24(self):
24608: (8)                  r""" Class-aware wrapper for `::fz_read_uint24()`."""
24609: (8)                  return _mupdf.FzStream_fz_read_uint24(self)
24610: (4)              def fz_read_uint24_le(self):
24611: (8)                  r""" Class-aware wrapper for `::fz_read_uint24_le()`."""
24612: (8)                  return _mupdf.FzStream_fz_read_uint24_le(self)
24613: (4)              def fz_read_uint32(self):
24614: (8)                  r""" Class-aware wrapper for `::fz_read_uint32()`."""
24615: (8)                  return _mupdf.FzStream_fz_read_uint32(self)
24616: (4)              def fz_read_uint32_le(self):
24617: (8)                  r""" Class-aware wrapper for `::fz_read_uint32_le()`."""
24618: (8)                  return _mupdf.FzStream_fz_read_uint32_le(self)
24619: (4)              def fz_read_uint64(self):
24620: (8)                  r""" Class-aware wrapper for `::fz_read_uint64()`."""
24621: (8)                  return _mupdf.FzStream_fz_read_uint64(self)
24622: (4)              def fz_read_uint64_le(self):
24623: (8)                  r""" Class-aware wrapper for `::fz_read_uint64_le()`."""
24624: (8)                  return _mupdf.FzStream_fz_read_uint64_le(self)
24625: (4)              def fz_read_utf16_be(self):
24626: (8)                  r""" Class-aware wrapper for `::fz_read_utf16_be()`."""
24627: (8)                  return _mupdf.FzStream_fz_read_utf16_be(self)
24628: (4)              def fz_read_utf16_le(self):
24629: (8)                  r"""
24630: (8)                  Class-aware wrapper for `::fz_read_utf16_le()`.
24631: (12)                     Read a utf-16 rune from a stream. (little endian and
24632: (12)                     big endian respectively).
24633: (12)                     In the event of encountering badly formatted utf-16 codes
24634: (12)                     (mismatched surrogates) no error/exception is given, but
24635: (12)                     undefined values may be returned.
24636: (8)                  """
24637: (8)                  return _mupdf.FzStream_fz_read_utf16_le(self)
24638: (4)              def fz_seek(self, offset, whence):
```

```
24639: (8)                  r"""
24640: (8)                  Class-aware wrapper for `::fz_seek()`.
24641: (12)                     Seek within a stream.
24642: (12)                     stm: The stream to seek within.
24643: (12)                     offset: The offset to seek to.
24644: (12)                     whence: From where the offset is measured (see fseek).
24645: (12)                     SEEK_SET - start of stream.
24646: (12)                     SEEK_CUR - current position.
24647: (12)                     SEEK_END - end of stream.
24648: (8)                  """
24649: (8)                  return _mupdf.FzStream_fz_seek(self, offset, whence)
24650: (4)              def fz_skip(self, len):
24651: (8)                  r"""
24652: (8)                  Class-aware wrapper for `::fz_skip()`.
24653: (12)                     Read from a stream discarding data.
24654: (12)                     stm: The stream to read from.
24655: (12)                     len: The number of bytes to read.
24656: (12)                     Returns the number of bytes read. May throw exceptions.
24657: (8)                  """
24658: (8)                  return _mupdf.FzStream_fz_skip(self, len)
24659: (4)              def fz_skip_space(self):
24660: (8)                  r"""
24661: (8)                  Class-aware wrapper for `::fz_skip_space()`.
24662: (12)                     Skip over whitespace (bytes <= 32) in a stream.
24663: (8)                  """
24664: (8)                  return _mupdf.FzStream_fz_skip_space(self)
24665: (4)              def fz_skip_string(self, str):
24666: (8)                  r"""
24667: (8)                  Class-aware wrapper for `::fz_skip_string()`.
24668: (12)                     Skip over a given string in a stream. Return 0 if successfully
24669: (12)                     skipped, non-zero otherwise. As many characters will be skipped
24670: (12)                     over as matched in the string.
24671: (8)                  """
24672: (8)                  return _mupdf.FzStream_fz_skip_string(self, str)
24673: (4)              def fz_stream_filename(self):
24674: (8)                  r"""
24675: (8)                  Class-aware wrapper for `::fz_stream_filename()`.
24676: (12)                     Return the filename (UTF-8 encoded) from which a stream was
opened.
24677: (12)                     Returns NULL if the filename is not available (or the stream was
24678: (12)                     opened from a source other than a file).
24679: (8)                  """
24680: (8)                  return _mupdf.FzStream_fz_stream_filename(self)
24681: (4)              def fz_sync_bits(self):
24682: (8)                  r"""
24683: (8)                  Class-aware wrapper for `::fz_sync_bits()`.
24684: (12)                     Called after reading bits to tell the stream
24685: (12)                     that we are about to return to reading bytewise. Resyncs
24686: (12)                     the stream to whole byte boundaries.
24687: (8)                  """
24688: (8)                  return _mupdf.FzStream_fz_sync_bits(self)
24689: (4)              def fz_tell(self):
24690: (8)                  r"""
24691: (8)                  Class-aware wrapper for `::fz_tell()`.
24692: (12)                     return the current reading position within a stream
24693: (8)                  """
24694: (8)                  return _mupdf.FzStream_fz_tell(self)
24695: (4)              def fz_try_open_archive_with_stream(self):
24696: (8)                  r"""
24697: (8)                  Class-aware wrapper for `::fz_try_open_archive_with_stream()`.
24698: (12)                     Open zip or tar archive stream.
24699: (12)                     Does the same as fz_open_archive_with_stream, but will not throw
24700: (12)                     an error in the event of failing to recognise the format. Will
24701: (12)                     still throw errors in other cases though!
24702: (8)                  """
24703: (8)                  return _mupdf.FzStream_fz_try_open_archive_with_stream(self)
24704: (4)              def fz_unpack_stream(self, depth, w, h, n, indexed, pad, skip):
24705: (8)                  r""" Class-aware wrapper for `::fz_unpack_stream()`."""
24706: (8)                  return _mupdf.FzStream_fz_unpack_stream(self, depth, w, h, n, indexed,
```

```
pad, skip)
24707: (4)              def fz_unread_byte(self):
24708: (8)                  r"""
24709: (8)                  Class-aware wrapper for `::fz_unread_byte()`.
24710: (12)                     Unread the single last byte successfully
24711: (12)                     read from a stream. Do not call this without having
24712: (12)                     successfully read a byte.
24713: (12)                     stm: The stream to operate upon.
24714: (8)                  """
24715: (8)                  return _mupdf.FzStream_fz_unread_byte(self)
24716: (4)              def pdf_load_cmap(self):
24717: (8)                  r""" Class-aware wrapper for `::pdf_load_cmap()`."""
24718: (8)                  return _mupdf.FzStream_pdf_load_cmap(self)
24719: (4)              def pdf_open_crypt(self, crypt, num, gen):
24720: (8)                  r""" Class-aware wrapper for `::pdf_open_crypt()`."""
24721: (8)                  return _mupdf.FzStream_pdf_open_crypt(self, crypt, num, gen)
24722: (4)              def pdf_open_crypt_with_filter(self, crypt, name, num, gen):
24723: (8)                  r""" Class-aware wrapper for `::pdf_open_crypt_with_filter()`."""
24724: (8)                  return _mupdf.FzStream_pdf_open_crypt_with_filter(self, crypt, name,
num, gen)
24725: (4)              def pdf_open_document_with_stream(self):
24726: (8)                  r""" Class-aware wrapper for `::pdf_open_document_with_stream()`."""
24727: (8)                  return _mupdf.FzStream_pdf_open_document_with_stream(self)
24728: (4)              def __init__(self, *args):
24729: (8)                  r"""
24730: (8)                  *Overload 1:*
24731: (9)                   == Constructors.  Constructor using `fz_new_stream()`.
24732: (16)                         Create a new stream object with the given
24733: (16)                         internal state and function pointers.
24734: (16)                         state: Internal state (opaque to everything but
implementation).
24735: (16)                         next: Should provide the next set of bytes (up to max) of
stream
24736: (16)                         data. Return the number of bytes read, or EOF when there is no
24737: (16)                         more data.
24738: (16)                         drop: Should clean up and free the internal state. May not
24739: (16)                         throw exceptions.
24740: (8)                  |
24741: (8)                  *Overload 2:*
24742: (9)                   Constructor using `fz_open_file()`.
24743: (16)                         Open the named file and wrap it in a stream.
24744: (16)                         filename: Path to a file. On non-Windows machines the filename
24745: (16)                         should be exactly as it would be passed to fopen(2). On
Windows
24746: (16)                         machines, the path should be UTF-8 encoded so that non-ASCII
24747: (16)                         characters can be represented. Other platforms do the encoding
24748: (16)                         as standard anyway (and in most cases, particularly for MacOS
24749: (16)                         and Linux, the encoding they use is UTF-8 anyway).
24750: (8)                  |
24751: (8)                  *Overload 3:*
24752: (9)                   Constructor using `fz_open_file_ptr_no_close()`.
24753: (16)                         Create a stream from a FILE * that will not be closed
24754: (16)                         when the stream is dropped.
24755: (8)                  |
24756: (8)                  *Overload 4:*
24757: (9)                   Constructor using `fz_open_memory()`.
24758: (16)                         Open a block of memory as a stream.
24759: (16)                         data: Pointer to start of data block. Ownership of the data
24760: (16)                         block is NOT passed in.
24761: (16)                         len: Number of bytes in data block.
24762: (16)                         Returns pointer to newly created stream. May throw exceptions
on
24763: (16)                         failure to allocate.
24764: (8)                  |
24765: (8)                  *Overload 5:*
24766: (9)                   Construct using fz_open_file().
24767: (8)                  |
24768: (8)                  *Overload 6:*
24769: (9)                   Copy constructor using `fz_keep_stream()`.
```

```
24770: (8)                  |
24771: (8)                  *Overload 7:*
24772: (9)                   Default constructor, sets `m_internal` to null.
24773: (8)                  |
24774: (8)                  *Overload 8:*
24775: (9)                   Constructor using raw copy of pre-existing `::fz_stream`.
24776: (8)                  """
24777: (8)                  _mupdf.FzStream_swiginit(self, _mupdf.new_FzStream(*args))
24778: (4)              __swig_destroy__ = _mupdf.delete_FzStream
24779: (4)              def m_internal_value(self):
24780: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24781: (8)                  return _mupdf.FzStream_m_internal_value(self)
24782: (4)              m_internal = property(_mupdf.FzStream_m_internal_get,
_mupdf.FzStream_m_internal_set, doc=r""" Pointer to wrapped data.""")
24783: (4)              s_num_instances = property(_mupdf.FzStream_s_num_instances_get,
_mupdf.FzStream_s_num_instances_set)
24784: (0)          _mupdf.FzStream_swigregister(FzStream)
24785: (0)          class FzString(object):
24786: (4)              r""" Wrapper class for struct `fz_string`."""
24787: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24788: (4)              __repr__ = _swig_repr
24789: (4)              def __init__(self, *args):
24790: (8)                  r"""
24791: (8)                  *Overload 1:*
24792: (8)                  == Constructors.  Constructor using `fz_new_string()`.
24793: (8)                  |
24794: (8)                  *Overload 2:*
24795: (8)                  Copy constructor using `fz_keep_string()`.
24796: (8)                  |
24797: (8)                  *Overload 3:*
24798: (8)                  Default constructor, sets `m_internal` to null.
24799: (8)                  |
24800: (8)                  *Overload 4:*
24801: (8)                  Constructor using raw copy of pre-existing `::fz_string`.
24802: (8)                  """
24803: (8)                  _mupdf.FzString_swiginit(self, _mupdf.new_FzString(*args))
24804: (4)              __swig_destroy__ = _mupdf.delete_FzString
24805: (4)              def m_internal_value(self):
24806: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24807: (8)                  return _mupdf.FzString_m_internal_value(self)
24808: (4)              m_internal = property(_mupdf.FzString_m_internal_get,
_mupdf.FzString_m_internal_set, doc=r""" Pointer to wrapped data.""")
24809: (4)              s_num_instances = property(_mupdf.FzString_s_num_instances_get,
_mupdf.FzString_s_num_instances_set)
24810: (0)          _mupdf.FzString_swigregister(FzString)
24811: (0)          class FzStrokeState(object):
24812: (4)              r""" Wrapper class for struct `fz_stroke_state`."""
24813: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24814: (4)              __repr__ = _swig_repr
24815: (4)              def fz_clone_stroke_state(self):
24816: (8)                  r"""
24817: (8)                  Class-aware wrapper for `::fz_clone_stroke_state()`.
24818: (12)                     Create an identical stroke_state structure and return a
24819: (12)                     reference to it.
24820: (12)                     stroke: The stroke state reference to clone.
24821: (12)                     Exceptions may be thrown in the event of a failure to
24822: (12)                     allocate.
24823: (8)                  """
24824: (8)                  return _mupdf.FzStrokeState_fz_clone_stroke_state(self)
24825: (4)              def fz_unshare_stroke_state(self):
24826: (8)                  r"""
24827: (8)                  Class-aware wrapper for `::fz_unshare_stroke_state()`.
24828: (12)                     Given a reference to a (possibly) shared stroke_state structure,
24829: (12)                     return a reference to an equivalent stroke_state structure
24830: (12)                     that is guaranteed to be unshared (i.e. one that can
```

```
24831: (12)                     safely be modified).
24832: (12)                     shared: The reference to a (possibly) shared structure
24833: (12)                     to unshare. Ownership of this reference is passed in
24834: (12)                     to this function, even in the case of exceptions being
24835: (12)                     thrown.
24836: (12)                     Exceptions may be thrown in the event of failure to
24837: (12)                     allocate if required.
24838: (8)                  """
24839: (8)                  return _mupdf.FzStrokeState_fz_unshare_stroke_state(self)
24840: (4)              def fz_unshare_stroke_state_with_dash_len(self, len):
24841: (8)                  r"""
24842: (8)                  Class-aware wrapper for `::fz_unshare_stroke_state_with_dash_len()`.
24843: (12)                     Given a reference to a (possibly) shared stroke_state structure,
24844: (12)                     return a reference to a stroke_state structure (with room for a
24845: (12)                     given amount of dash data) that is guaranteed to be unshared
24846: (12)                     (i.e. one that can safely be modified).
24847: (12)                     shared: The reference to a (possibly) shared structure
24848: (12)                     to unshare. Ownership of this reference is passed in
24849: (12)                     to this function, even in the case of exceptions being
24850: (12)                     thrown.
24851: (12)                     Exceptions may be thrown in the event of failure to
24852: (12)                     allocate if required.
24853: (8)                  """
24854: (8)                  return
_mupdf.FzStrokeState_fz_unshare_stroke_state_with_dash_len(self, len)
24855: (4)              def __init__(self, *args):
24856: (8)                  r"""
24857: (8)                  *Overload 1:*
24858: (9)                   == Constructors.  Constructor using `fz_new_stroke_state()`.
24859: (16)                         Create a new (empty) stroke state structure (with no dash
24860: (16)                         data) and return a reference to it.
24861: (16)                         Throws exception on failure to allocate.
24862: (8)                  |
24863: (8)                  *Overload 2:*
24864: (9)                   Constructor using `fz_new_stroke_state_with_dash_len()`.
24865: (16)                         Create a new (empty) stroke state structure, with room for
24866: (16)                         dash data of the given length, and return a reference to it.
24867: (16)                         len: The number of dash elements to allow room for.
24868: (16)                         Throws exception on failure to allocate.
24869: (8)                  |
24870: (8)                  *Overload 3:*
24871: (9)                   Copy constructor using `fz_keep_stroke_state()`.
24872: (8)                  |
24873: (8)                  *Overload 4:*
24874: (9)                   Constructor using raw copy of pre-existing `::fz_stroke_state`.
24875: (8)                  """
24876: (8)                  _mupdf.FzStrokeState_swiginit(self, _mupdf.new_FzStrokeState(*args))
24877: (4)              __swig_destroy__ = _mupdf.delete_FzStrokeState
24878: (4)              def m_internal_value(self):
24879: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24880: (8)                  return _mupdf.FzStrokeState_m_internal_value(self)
24881: (4)              m_internal = property(_mupdf.FzStrokeState_m_internal_get,
_mupdf.FzStrokeState_m_internal_set, doc=r""" Pointer to wrapped data.""")
24882: (4)              s_num_instances = property(_mupdf.FzStrokeState_s_num_instances_get,
_mupdf.FzStrokeState_s_num_instances_set)
24883: (0)          _mupdf.FzStrokeState_swigregister(FzStrokeState)
24884: (0)          class FzStyleContext(object):
24885: (4)              r""" Wrapper class for struct `fz_style_context`. Not copyable or
assignable."""
24886: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24887: (4)              __repr__ = _swig_repr
24888: (4)              def __init__(self, *args):
24889: (8)                  r"""
24890: (8)                  *Overload 1:*
24891: (8)                  Default constructor, sets `m_internal` to null.
24892: (8)                  |
24893: (8)                  *Overload 2:*
```

```
24894: (8)                  Constructor using raw copy of pre-existing `::fz_style_context`.
24895: (8)                  """
24896: (8)                  _mupdf.FzStyleContext_swiginit(self, _mupdf.new_FzStyleContext(*args))
24897: (4)              __swig_destroy__ = _mupdf.delete_FzStyleContext
24898: (4)              def m_internal_value(self):
24899: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24900: (8)                  return _mupdf.FzStyleContext_m_internal_value(self)
24901: (4)              m_internal = property(_mupdf.FzStyleContext_m_internal_get,
_mupdf.FzStyleContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
24902: (4)              s_num_instances = property(_mupdf.FzStyleContext_s_num_instances_get,
_mupdf.FzStyleContext_s_num_instances_set)
24903: (0)          _mupdf.FzStyleContext_swigregister(FzStyleContext)
24904: (0)          class FzText(object):
24905: (4)              r""" Wrapper class for struct `fz_text`."""
24906: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24907: (4)              __repr__ = _swig_repr
24908: (4)              def fz_bound_text(self, stroke, ctm):
24909: (8)                  r"""
24910: (8)                  Class-aware wrapper for `::fz_bound_text()`.
24911: (12)                     Find the bounds of a given text object.
24912: (12)                     text: The text object to find the bounds of.
24913: (12)                     stroke: Pointer to the stroke attributes (for stroked
24914: (12)                     text), or NULL (for filled text).
24915: (12)                     ctm: The matrix in use.
24916: (12)                     r: pointer to storage for the bounds.
24917: (12)                     Returns a pointer to r, which is updated to contain the
24918: (12)                     bounding box for the text object.
24919: (8)                  """
24920: (8)                  return _mupdf.FzText_fz_bound_text(self, stroke, ctm)
24921: (4)              def fz_show_glyph(self, font, trm, glyph, unicode, wmode, bidi_level,
markup_dir, language):
24922: (8)                  r"""
24923: (8)                  Class-aware wrapper for `::fz_show_glyph()`.
24924: (12)                     Add a glyph/unicode value to a text object.
24925: (12)                     text: Text object to add to.
24926: (12)                     font: The font the glyph should be added in.
24927: (12)                     trm: The transform to use for the glyph.
24928: (12)                     glyph: The glyph id to add.
24929: (12)                     unicode: The unicode character for the glyph.
24930: (12)                     cid: The CJK CID value or raw character code.
24931: (12)                     wmode: 1 for vertical mode, 0 for horizontal.
24932: (12)                     bidi_level: The bidirectional level for this glyph.
24933: (12)                     markup_dir: The direction of the text as specified in the
24934: (12)                     markup.
24935: (12)                     language: The language in use (if known, 0 otherwise)
24936: (12)                     (e.g. FZ_LANG_zh_Hans).
24937: (12)                     Throws exception on failure to allocate.
24938: (8)                  """
24939: (8)                  return _mupdf.FzText_fz_show_glyph(self, font, trm, glyph, unicode,
wmode, bidi_level, markup_dir, language)
24940: (4)              def fz_show_glyph_aux(self, font, trm, glyph, unicode, cid, wmode,
bidi_level, markup_dir, lang):
24941: (8)                  r""" Class-aware wrapper for `::fz_show_glyph_aux()`."""
24942: (8)                  return _mupdf.FzText_fz_show_glyph_aux(self, font, trm, glyph,
unicode, cid, wmode, bidi_level, markup_dir, lang)
24943: (4)              def fz_show_string(self, font, trm, s, wmode, bidi_level, markup_dir,
language):
24944: (8)                  r"""
24945: (8)                  Class-aware wrapper for `::fz_show_string()`.
24946: (12)                     Add a UTF8 string to a text object.
24947: (12)                     text: Text object to add to.
24948: (12)                     font: The font the string should be added in.
24949: (12)                     trm: The transform to use.
24950: (12)                     s: The utf-8 string to add.
24951: (12)                     wmode: 1 for vertical mode, 0 for horizontal.
24952: (12)                     bidi_level: The bidirectional level for this glyph.
24953: (12)                     markup_dir: The direction of the text as specified in the markup.
```

```
24954: (12)                     language: The language in use (if known, 0 otherwise)
24955: (16)                         (e.g. FZ_LANG_zh_Hans).
24956: (12)                     Returns the transform updated with the advance width of the
24957: (12)                     string.
24958: (8)                  """
24959: (8)                  return _mupdf.FzText_fz_show_string(self, font, trm, s, wmode,
bidi_level, markup_dir, language)
24960: (4)              def __init__(self, *args):
24961: (8)                  r"""
24962: (8)                  *Overload 1:*
24963: (9)                   == Constructors.  Constructor using `fz_new_text()`.
24964: (16)                         Create a new empty fz_text object.
24965: (16)                         Throws exception on failure to allocate.
24966: (8)                  |
24967: (8)                  *Overload 2:*
24968: (9)                   Copy constructor using `fz_keep_text()`.
24969: (8)                  |
24970: (8)                  *Overload 3:*
24971: (9)                   Constructor using raw copy of pre-existing `::fz_text`.
24972: (8)                  """
24973: (8)                  _mupdf.FzText_swiginit(self, _mupdf.new_FzText(*args))
24974: (4)              __swig_destroy__ = _mupdf.delete_FzText
24975: (4)              def m_internal_value(self):
24976: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
24977: (8)                  return _mupdf.FzText_m_internal_value(self)
24978: (4)              m_internal = property(_mupdf.FzText_m_internal_get,
_mupdf.FzText_m_internal_set, doc=r""" Pointer to wrapped data.""")
24979: (4)              s_num_instances = property(_mupdf.FzText_s_num_instances_get,
_mupdf.FzText_s_num_instances_set)
24980: (0)          _mupdf.FzText_swigregister(FzText)
24981: (0)          class FzTextDecoder(object):
24982: (4)              r"""
24983: (5)               Wrapper class for struct `fz_text_decoder`. Not copyable or assignable.
24984: (4)              A text decoder (to read arbitrary encodings and convert to unicode).
24985: (4)              """
24986: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
24987: (4)              __repr__ = _swig_repr
24988: (4)              def fz_init_text_decoder(self, encoding):
24989: (8)                  r""" Class-aware wrapper for `::fz_init_text_decoder()`."""
24990: (8)                  return _mupdf.FzTextDecoder_fz_init_text_decoder(self, encoding)
24991: (4)              def __init__(self, *args):
24992: (8)                  r"""
24993: (8)                  *Overload 1:*
24994: (8)                  Default constructor, sets `m_internal` to null.
24995: (8)                  |
24996: (8)                  *Overload 2:*
24997: (8)                  Constructor using raw copy of pre-existing `::fz_text_decoder`.
24998: (8)                  """
24999: (8)                  _mupdf.FzTextDecoder_swiginit(self, _mupdf.new_FzTextDecoder(*args))
25000: (4)              __swig_destroy__ = _mupdf.delete_FzTextDecoder
25001: (4)              def m_internal_value(self):
25002: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25003: (8)                  return _mupdf.FzTextDecoder_m_internal_value(self)
25004: (4)              m_internal = property(_mupdf.FzTextDecoder_m_internal_get,
_mupdf.FzTextDecoder_m_internal_set, doc=r""" Pointer to wrapped data.""")
25005: (4)              s_num_instances = property(_mupdf.FzTextDecoder_s_num_instances_get,
_mupdf.FzTextDecoder_s_num_instances_set)
25006: (0)          _mupdf.FzTextDecoder_swigregister(FzTextDecoder)
25007: (0)          class FzTextItem(object):
25008: (4)              r"""
25009: (4)              Wrapper class for struct `fz_text_item`. Not copyable or assignable.
25010: (4)              Text buffer.
25011: (4)              The trm field contains the a, b, c and d coefficients.
25012: (4)              The e and f coefficients come from the individual elements,
25013: (4)              together they form the transform matrix for the glyph.
25014: (4)              Glyphs are referenced by glyph ID.
```

```
25015: (4)              The Unicode text equivalent is kept in a separate array
25016: (4)              with indexes into the glyph array.
25017: (4)              """
25018: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25019: (4)              __repr__ = _swig_repr
25020: (4)              def __init__(self, *args):
25021: (8)                  r"""
25022: (8)                  *Overload 1:*
25023: (8)                  Default constructor, sets `m_internal` to null.
25024: (8)                  |
25025: (8)                  *Overload 2:*
25026: (8)                  Constructor using raw copy of pre-existing `::fz_text_item`.
25027: (8)                  """
25028: (8)                  _mupdf.FzTextItem_swiginit(self, _mupdf.new_FzTextItem(*args))
25029: (4)              __swig_destroy__ = _mupdf.delete_FzTextItem
25030: (4)              def m_internal_value(self):
25031: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25032: (8)                  return _mupdf.FzTextItem_m_internal_value(self)
25033: (4)              m_internal = property(_mupdf.FzTextItem_m_internal_get,
_mupdf.FzTextItem_m_internal_set, doc=r""" Pointer to wrapped data.""")
25034: (4)              s_num_instances = property(_mupdf.FzTextItem_s_num_instances_get,
_mupdf.FzTextItem_s_num_instances_set)
25035: (0)          _mupdf.FzTextItem_swigregister(FzTextItem)
25036: (0)          class FzTextSpan(object):
25037: (4)              r""" Wrapper class for struct `fz_text_span`. Not copyable or
assignable."""
25038: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25039: (4)              __repr__ = _swig_repr
25040: (4)              def font(self):
25041: (8)                  r""" Gives class-aware access to m_internal->font."""
25042: (8)                  return _mupdf.FzTextSpan_font(self)
25043: (4)              def trm(self):
25044: (8)                  r""" Gives class-aware access to m_internal->trm."""
25045: (8)                  return _mupdf.FzTextSpan_trm(self)
25046: (4)              def items(self, i):
25047: (8)                  r"""
25048: (8)                  Gives access to m_internal->items[i].
25049: (32)                                         Returned reference is only valid as long as
`this`.
25050: (32)                                         Provided mainly for use by SWIG bindings.
25051: (8)                  """
25052: (8)                  return _mupdf.FzTextSpan_items(self, i)
25053: (4)              def __init__(self, *args):
25054: (8)                  r"""
25055: (8)                  *Overload 1:*
25056: (8)                  Default constructor, sets `m_internal` to null.
25057: (8)                  |
25058: (8)                  *Overload 2:*
25059: (8)                  Constructor using raw copy of pre-existing `::fz_text_span`.
25060: (8)                  """
25061: (8)                  _mupdf.FzTextSpan_swiginit(self, _mupdf.new_FzTextSpan(*args))
25062: (4)              __swig_destroy__ = _mupdf.delete_FzTextSpan
25063: (4)              def m_internal_value(self):
25064: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25065: (8)                  return _mupdf.FzTextSpan_m_internal_value(self)
25066: (4)              m_internal = property(_mupdf.FzTextSpan_m_internal_get,
_mupdf.FzTextSpan_m_internal_set, doc=r""" Pointer to wrapped data.""")
25067: (4)              s_num_instances = property(_mupdf.FzTextSpan_s_num_instances_get,
_mupdf.FzTextSpan_s_num_instances_set)
25068: (0)          _mupdf.FzTextSpan_swigregister(FzTextSpan)
25069: (0)          class FzTransition(object):
25070: (4)              r""" Wrapper class for struct `fz_transition`."""
25071: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25072: (4)              __repr__ = _swig_repr
```

```
25073: (4)              def __init__(self, *args):
25074: (8)                  r"""
25075: (8)                  *Overload 1:*
25076: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
25077: (8)                  |
25078: (8)                  *Overload 2:*
25079: (8)                  Constructor using raw copy of pre-existing `::fz_transition`.
25080: (8)                  |
25081: (8)                  *Overload 3:*
25082: (8)                  Constructor using raw copy of pre-existing `::fz_transition`.
25083: (8)                  """
25084: (8)                  _mupdf.FzTransition_swiginit(self, _mupdf.new_FzTransition(*args))
25085: (4)              def internal(self, *args):
25086: (8)                  r"""
25087: (8)                  *Overload 1:*
25088: (8)                  Access as underlying struct.
25089: (8)                  |
25090: (8)                  *Overload 2:*
25091: (8)                  Access as underlying struct.
25092: (8)                  """
25093: (8)                  return _mupdf.FzTransition_internal(self, *args)
25094: (4)              __swig_destroy__ = _mupdf.delete_FzTransition
25095: (4)              type = property(_mupdf.FzTransition_type_get,
_mupdf.FzTransition_type_set)
25096: (4)              duration = property(_mupdf.FzTransition_duration_get,
_mupdf.FzTransition_duration_set)
25097: (4)              vertical = property(_mupdf.FzTransition_vertical_get,
_mupdf.FzTransition_vertical_set)
25098: (4)              outwards = property(_mupdf.FzTransition_outwards_get,
_mupdf.FzTransition_outwards_set)
25099: (4)              direction = property(_mupdf.FzTransition_direction_get,
_mupdf.FzTransition_direction_set)
25100: (4)              state0 = property(_mupdf.FzTransition_state0_get,
_mupdf.FzTransition_state0_set)
25101: (4)              state1 = property(_mupdf.FzTransition_state1_get,
_mupdf.FzTransition_state1_set)
25102: (4)              s_num_instances = property(_mupdf.FzTransition_s_num_instances_get,
_mupdf.FzTransition_s_num_instances_set)
25103: (4)              def to_string(self):
25104: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
25105: (8)                  return _mupdf.FzTransition_to_string(self)
25106: (4)              def __eq__(self, rhs):
25107: (8)                  r""" Comparison method."""
25108: (8)                  return _mupdf.FzTransition___eq__(self, rhs)
25109: (4)              def __ne__(self, rhs):
25110: (8)                  r""" Comparison method."""
25111: (8)                  return _mupdf.FzTransition___ne__(self, rhs)
25112: (0)          _mupdf.FzTransition_swigregister(FzTransition)
25113: (0)          class FzTree(object):
25114: (4)              r"""
25115: (4)              Wrapper class for struct `fz_tree`. Not copyable or assignable.
25116: (4)              AA-tree to look up things by strings.
25117: (4)              """
25118: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25119: (4)              __repr__ = _swig_repr
25120: (4)              def fz_new_tree_archive(self):
25121: (8)                  r"""
25122: (8)                  Class-aware wrapper for `::fz_new_tree_archive()`.
25123: (12)                     Create an archive that holds named buffers.
25124: (12)                     tree can either be a preformed tree with fz_buffers as values,
25125: (12)                     or it can be NULL for an empty tree.
25126: (8)                  """
25127: (8)                  return _mupdf.FzTree_fz_new_tree_archive(self)
25128: (4)              def fz_tree_lookup(self, key):
25129: (8)                  r"""
25130: (8)                  Class-aware wrapper for `::fz_tree_lookup()`.
```

```
25131: (12)                     Look for the value of a node in the tree with the given key.
25132: (12)                     Simple pointer equivalence is used for key.
25133: (12)                     Returns NULL for no match.
25134: (8)                  """
25135: (8)                  return _mupdf.FzTree_fz_tree_lookup(self, key)
25136: (4)              def __init__(self, *args):
25137: (8)                  r"""
25138: (8)                  *Overload 1:*
25139: (8)                  Default constructor, sets `m_internal` to null.
25140: (8)                  |
25141: (8)                  *Overload 2:*
25142: (8)                  Constructor using raw copy of pre-existing `::fz_tree`.
25143: (8)                  """
25144: (8)                  _mupdf.FzTree_swiginit(self, _mupdf.new_FzTree(*args))
25145: (4)              __swig_destroy__ = _mupdf.delete_FzTree
25146: (4)              def m_internal_value(self):
25147: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25148: (8)                  return _mupdf.FzTree_m_internal_value(self)
25149: (4)              m_internal = property(_mupdf.FzTree_m_internal_get,
_mupdf.FzTree_m_internal_set, doc=r""" Pointer to wrapped data.""")
25150: (4)              s_num_instances = property(_mupdf.FzTree_s_num_instances_get,
_mupdf.FzTree_s_num_instances_set)
25151: (0)          _mupdf.FzTree_swigregister(FzTree)
25152: (0)          class FzTuningContext(object):
25153: (4)              r""" Wrapper class for struct `fz_tuning_context`. Not copyable or
assignable."""
25154: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25155: (4)              __repr__ = _swig_repr
25156: (4)              def __init__(self, *args):
25157: (8)                  r"""
25158: (8)                  *Overload 1:*
25159: (8)                  Default constructor, sets `m_internal` to null.
25160: (8)                  |
25161: (8)                  *Overload 2:*
25162: (8)                  Constructor using raw copy of pre-existing `::fz_tuning_context`.
25163: (8)                  """
25164: (8)                  _mupdf.FzTuningContext_swiginit(self,
_mupdf.new_FzTuningContext(*args))
25165: (4)              __swig_destroy__ = _mupdf.delete_FzTuningContext
25166: (4)              def m_internal_value(self):
25167: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25168: (8)                  return _mupdf.FzTuningContext_m_internal_value(self)
25169: (4)              m_internal = property(_mupdf.FzTuningContext_m_internal_get,
_mupdf.FzTuningContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
25170: (4)              s_num_instances = property(_mupdf.FzTuningContext_s_num_instances_get,
_mupdf.FzTuningContext_s_num_instances_set)
25171: (0)          _mupdf.FzTuningContext_swigregister(FzTuningContext)
25172: (0)          class FzVertex(object):
25173: (4)              r"""
25174: (4)              Wrapper class for struct `fz_vertex`. Not copyable or assignable.
25175: (4)              Handy routine for processing mesh based shades
25176: (4)              """
25177: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25178: (4)              __repr__ = _swig_repr
25179: (4)              def __init__(self, *args):
25180: (8)                  r"""
25181: (8)                  *Overload 1:*
25182: (8)                  Default constructor, sets `m_internal` to null.
25183: (8)                  |
25184: (8)                  *Overload 2:*
25185: (8)                  Constructor using raw copy of pre-existing `::fz_vertex`.
25186: (8)                  """
25187: (8)                  _mupdf.FzVertex_swiginit(self, _mupdf.new_FzVertex(*args))
25188: (4)              __swig_destroy__ = _mupdf.delete_FzVertex
25189: (4)              def m_internal_value(self):
```

```
25190: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25191: (8)                  return _mupdf.FzVertex_m_internal_value(self)
25192: (4)              m_internal = property(_mupdf.FzVertex_m_internal_get,
_mupdf.FzVertex_m_internal_set, doc=r""" Pointer to wrapped data.""")
25193: (4)              s_num_instances = property(_mupdf.FzVertex_s_num_instances_get,
_mupdf.FzVertex_s_num_instances_set)
25194: (0)          _mupdf.FzVertex_swigregister(FzVertex)
25195: (0)          class FzWarnContext(object):
25196: (4)              r""" Wrapper class for struct `fz_warn_context`. Not copyable or
assignable."""
25197: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25198: (4)              __repr__ = _swig_repr
25199: (4)              def __init__(self, *args):
25200: (8)                  r"""
25201: (8)                  *Overload 1:*
25202: (8)                  Default constructor, sets `m_internal` to null.
25203: (8)                  |
25204: (8)                  *Overload 2:*
25205: (8)                  Constructor using raw copy of pre-existing `::fz_warn_context`.
25206: (8)                  """
25207: (8)                  _mupdf.FzWarnContext_swiginit(self, _mupdf.new_FzWarnContext(*args))
25208: (4)              __swig_destroy__ = _mupdf.delete_FzWarnContext
25209: (4)              def m_internal_value(self):
25210: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25211: (8)                  return _mupdf.FzWarnContext_m_internal_value(self)
25212: (4)              m_internal = property(_mupdf.FzWarnContext_m_internal_get,
_mupdf.FzWarnContext_m_internal_set, doc=r""" Pointer to wrapped data.""")
25213: (4)              s_num_instances = property(_mupdf.FzWarnContext_s_num_instances_get,
_mupdf.FzWarnContext_s_num_instances_set)
25214: (0)          _mupdf.FzWarnContext_swigregister(FzWarnContext)
25215: (0)          class FzWriteStoryPosition(object):
25216: (4)              r""" Wrapper class for struct `fz_write_story_position`. Not copyable or
assignable."""
25217: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25218: (4)              __repr__ = _swig_repr
25219: (4)              def __init__(self, *args):
25220: (8)                  r"""
25221: (8)                  *Overload 1:*
25222: (8)                  Default constructor, sets `m_internal` to null.
25223: (8)                  |
25224: (8)                  *Overload 2:*
25225: (8)                  Constructor using raw copy of pre-existing
`::fz_write_story_position`.
25226: (8)                  """
25227: (8)                  _mupdf.FzWriteStoryPosition_swiginit(self,
_mupdf.new_FzWriteStoryPosition(*args))
25228: (4)              __swig_destroy__ = _mupdf.delete_FzWriteStoryPosition
25229: (4)              def m_internal_value(self):
25230: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25231: (8)                  return _mupdf.FzWriteStoryPosition_m_internal_value(self)
25232: (4)              m_internal = property(_mupdf.FzWriteStoryPosition_m_internal_get,
_mupdf.FzWriteStoryPosition_m_internal_set, doc=r""" Pointer to wrapped data.""")
25233: (4)              s_num_instances =
property(_mupdf.FzWriteStoryPosition_s_num_instances_get,
_mupdf.FzWriteStoryPosition_s_num_instances_set)
25234: (0)          _mupdf.FzWriteStoryPosition_swigregister(FzWriteStoryPosition)
25235: (0)          class FzWriteStoryPositions(object):
25236: (4)              r""" Wrapper class for struct `fz_write_story_positions`. Not copyable or
assignable."""
25237: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25238: (4)              __repr__ = _swig_repr
25239: (4)              def __init__(self, *args):
25240: (8)                  r"""
```

```
25241: (8)                  *Overload 1:*
25242: (8)                  Default constructor, sets `m_internal` to null.
25243: (8)                  |
25244: (8)                  *Overload 2:*
25245: (8)                  Constructor using raw copy of pre-existing
`::fz_write_story_positions`.
25246: (8)                  """
25247: (8)                  _mupdf.FzWriteStoryPositions_swiginit(self,
_mupdf.new_FzWriteStoryPositions(*args))
25248: (4)              __swig_destroy__ = _mupdf.delete_FzWriteStoryPositions
25249: (4)              def m_internal_value(self):
25250: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25251: (8)                  return _mupdf.FzWriteStoryPositions_m_internal_value(self)
25252: (4)              m_internal = property(_mupdf.FzWriteStoryPositions_m_internal_get,
_mupdf.FzWriteStoryPositions_m_internal_set, doc=r""" Pointer to wrapped data.""")
25253: (4)              s_num_instances =
property(_mupdf.FzWriteStoryPositions_s_num_instances_get,
_mupdf.FzWriteStoryPositions_s_num_instances_set)
25254: (0)          _mupdf.FzWriteStoryPositions_swigregister(FzWriteStoryPositions)
25255: (0)          class FzXml(object):
25256: (4)              r"""
25257: (4)              Wrapper class for struct `fz_xml`.
25258: (4)              XML document model
25259: (4)              """
25260: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25261: (4)              __repr__ = _swig_repr
25262: (4)              def fz_debug_xml(self, level):
25263: (8)                  r"""
25264: (8)                  Class-aware wrapper for `::fz_debug_xml()`.
25265: (12)                     Pretty-print an XML tree to stdout. (Deprecated, use
25266: (12)                     fz_output_xml in preference).
25267: (8)                  """
25268: (8)                  return _mupdf.FzXml_fz_debug_xml(self, level)
25269: (4)              def fz_detach_xml(self):
25270: (8)                  r"""
25271: (8)                  Class-aware wrapper for `::fz_detach_xml()`.
25272: (12)                     Detach a node from the tree, unlinking it from its parent,
25273: (12)                     and setting the document root to the node.
25274: (8)                  """
25275: (8)                  return _mupdf.FzXml_fz_detach_xml(self)
25276: (4)              def fz_dom_add_attribute(self, att, value):
25277: (8)                  r"""
25278: (8)                  Class-aware wrapper for `::fz_dom_add_attribute()`.
25279: (12)                     Add an attribute to an element.
25280: (12)                     Ownership of att and value remain with the caller.
25281: (8)                  """
25282: (8)                  return _mupdf.FzXml_fz_dom_add_attribute(self, att, value)
25283: (4)              def fz_dom_append_child(self, child):
25284: (8)                  r"""
25285: (8)                  Class-aware wrapper for `::fz_dom_append_child()`.
25286: (12)                     Insert an element as the last child of a parent, unlinking the
25287: (12)                     child from its current position if required.
25288: (8)                  """
25289: (8)                  return _mupdf.FzXml_fz_dom_append_child(self, child)
25290: (4)              def fz_dom_attribute(self, att):
25291: (8)                  r"""
25292: (8)                  Class-aware wrapper for `::fz_dom_attribute()`.
25293: (12)                     Retrieve the value of a given attribute from a given element.
25294: (12)                     Returns a borrowed pointer to the value or NULL if not found.
25295: (8)                  """
25296: (8)                  return _mupdf.FzXml_fz_dom_attribute(self, att)
25297: (4)              def fz_dom_body(self):
25298: (8)                  r"""
25299: (8)                  Class-aware wrapper for `::fz_dom_body()`.
25300: (12)                     Return a borrowed reference for the 'body' element of
25301: (12)                     the given DOM.
25302: (8)                  """
```

```
(8)                 return _mupdf.FzXml_fz_dom_body(self)
(4)             def fz_dom_clone(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_clone()`.
(12)                    Clone an element (and its children).
(12)                    A borrowed reference to the clone is returned. The clone is not
(12)                    yet linked into the DOM.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_clone(self)
(4)             def fz_dom_create_element(self, tag):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_create_element()`.
(12)                    Create an element of a given tag type for the given DOM.
(12)                    The element is not linked into the DOM yet.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_create_element(self, tag)
(4)             def fz_dom_create_text_node(self, text):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_create_text_node()`.
(12)                    Create a text node for the given DOM.
(12)                    The element is not linked into the DOM yet.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_create_text_node(self, text)
(4)             def fz_dom_document_element(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_document_element()`.
(12)                    Return a borrowed reference for the document (the top
(12)                    level element) of the DOM.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_document_element(self)
(4)             def fz_dom_find(self, tag, att, match):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_find()`.
(12)                    Find the first element matching the requirements in a depth first
traversal from elt.
(12)                    The tagname must match tag, unless tag is NULL, when all tag names
are considered to match.
(12)                    If att is NULL, then all tags match.
(12)                    Otherwise:
(16)                        If match is NULL, then only nodes that have an att attribute
match.
(16)                        If match is non-NULL, then only nodes that have an att
attribute that matches match match.
(12)                    Returns NULL (if no match found), or a borrowed reference to the
first matching element.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_find(self, tag, att, match)
(4)             def fz_dom_find_next(self, tag, att, match):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_find_next()`.
(12)                    Find the next element matching the requirements.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_find_next(self, tag, att, match)
(4)             def fz_dom_first_child(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_first_child()`.
(12)                    Return a borrowed reference to the first child of a node,
(12)                    or NULL if there isn't one.
(8)                 """
(8)                 return _mupdf.FzXml_fz_dom_first_child(self)
(4)             def fz_dom_get_attribute(self, i, att):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_dom_get_attribute()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `fz_dom_get_attribute(int i)` => `(const char *, const char *att)`
(12)                    Enumerate through the attributes of an element.
(12)                    Call with i=0,1,2,3... to enumerate attributes.
(12)                    On return *att and the return value will be NULL if there are not
(12)                    that many attributes to read. Otherwise, *att will be filled in
```

```
25367: (12)                     with a borrowed pointer to the attribute name, and the return
25368: (12)                     value will be a borrowed pointer to the value.
25369: (8)                  """
25370: (8)                  return _mupdf.FzXml_fz_dom_get_attribute(self, i, att)
25371: (4)              def fz_dom_insert_after(self, new_elt):
25372: (8)                  r"""
25373: (8)                  Class-aware wrapper for `::fz_dom_insert_after()`.
25374: (12)                     Insert an element (new_elt), after another element (node),
25375: (12)                     unlinking the new_elt from its current position if required.
25376: (8)                  """
25377: (8)                  return _mupdf.FzXml_fz_dom_insert_after(self, new_elt)
25378: (4)              def fz_dom_insert_before(self, new_elt):
25379: (8)                  r"""
25380: (8)                  Class-aware wrapper for `::fz_dom_insert_before()`.
25381: (12)                     Insert an element (new_elt), before another element (node),
25382: (12)                     unlinking the new_elt from its current position if required.
25383: (8)                  """
25384: (8)                  return _mupdf.FzXml_fz_dom_insert_before(self, new_elt)
25385: (4)              def fz_dom_next(self):
25386: (8)                  r"""
25387: (8)                  Class-aware wrapper for `::fz_dom_next()`.
25388: (12)                     Return a borrowed reference to the next sibling of a node,
25389: (12)                     or NULL if there isn't one.
25390: (8)                  """
25391: (8)                  return _mupdf.FzXml_fz_dom_next(self)
25392: (4)              def fz_dom_parent(self):
25393: (8)                  r"""
25394: (8)                  Class-aware wrapper for `::fz_dom_parent()`.
25395: (12)                     Return a borrowed reference to the parent of a node,
25396: (12)                     or NULL if there isn't one.
25397: (8)                  """
25398: (8)                  return _mupdf.FzXml_fz_dom_parent(self)
25399: (4)              def fz_dom_previous(self):
25400: (8)                  r"""
25401: (8)                  Class-aware wrapper for `::fz_dom_previous()`.
25402: (12)                     Return a borrowed reference to the previous sibling of a node,
25403: (12)                     or NULL if there isn't one.
25404: (8)                  """
25405: (8)                  return _mupdf.FzXml_fz_dom_previous(self)
25406: (4)              def fz_dom_remove(self):
25407: (8)                  r"""
25408: (8)                  Class-aware wrapper for `::fz_dom_remove()`.
25409: (12)                     Remove an element from the DOM. The element can be added back
elsewhere
25410: (12)                     if required.
25411: (12)                     No reference counting changes for the element.
25412: (8)                  """
25413: (8)                  return _mupdf.FzXml_fz_dom_remove(self)
25414: (4)              def fz_dom_remove_attribute(self, att):
25415: (8)                  r"""
25416: (8)                  Class-aware wrapper for `::fz_dom_remove_attribute()`.
25417: (12)                     Remove an attribute from an element.
25418: (8)                  """
25419: (8)                  return _mupdf.FzXml_fz_dom_remove_attribute(self, att)
25420: (4)              def fz_new_display_list_from_svg_xml(self, xml, base_uri, dir, w, h):
25421: (8)                  r"""
25422: (8)                  Class-aware wrapper for `::fz_new_display_list_from_svg_xml()`.
25423: (8)                  This method has out-params. Python/C# wrappers look like:
25424: (12)                     `fz_new_display_list_from_svg_xml(::fz_xml *xml, const char
*base_uri, ::fz_archive *dir)` => `(fz_display_list *, float w, float h)`
25425: (12)                     Parse an SVG document into a display-list.
25426: (8)                  """
25427: (8)                  return _mupdf.FzXml_fz_new_display_list_from_svg_xml(self, xml,
base_uri, dir, w, h)
25428: (4)              def fz_new_image_from_svg_xml(self, xml, base_uri, dir):
25429: (8)                  r"""
25430: (8)                  Class-aware wrapper for `::fz_new_image_from_svg_xml()`.
25431: (12)                     Create a scalable image from an SVG document.
25432: (8)                  """
```

```
(8)                 return _mupdf.FzXml_fz_new_image_from_svg_xml(self, xml, base_uri,
dir)
(4)             def fz_xml_att(self, att):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_att()`.
(12)                    Return the value of an attribute of an XML node.
(12)                    NULL if the attribute doesn't exist.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_att(self, att)
(4)             def fz_xml_att_alt(self, one, two):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_att_alt()`.
(12)                    Return the value of an attribute of an XML node.
(12)                    If the first attribute doesn't exist, try the second.
(12)                    NULL if neither attribute exists.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_att_alt(self, one, two)
(4)             def fz_xml_att_eq(self, name, match):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_att_eq()`.
(12)                    Check for a matching attribute on an XML node.
(12)                    If the node has the requested attribute (name), and the value
(12)                    matches (match) then return 1. Otherwise, 0.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_att_eq(self, name, match)
(4)             def fz_xml_down(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_down()`.
(12)                    Return first child of XML node.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_down(self)
(4)             def fz_xml_find(self, tag):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_find()`.
(12)                    Search the siblings of XML nodes starting with item looking for
(12)                    the first with the given tag.
(12)                    Return NULL if none found.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_find(self, tag)
(4)             def fz_xml_find_dfs(self, tag, att, match):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_find_dfs()`.
(12)                    Perform a depth first search from item, returning the first
(12)                    child that matches the given tag (or any tag if tag is NULL),
(12)                    with the given attribute (if att is non NULL), that matches
(12)                    match (if match is non NULL).
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_find_dfs(self, tag, att, match)
(4)             def fz_xml_find_dfs_top(self, tag, att, match, top):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_find_dfs_top()`.
(12)                    Perform a depth first search from item, returning the first
(12)                    child that matches the given tag (or any tag if tag is NULL),
(12)                    with the given attribute (if att is non NULL), that matches
(12)                    match (if match is non NULL). The search stops if it ever
(12)                    reaches the top of the tree, or the declared 'top' item.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_find_dfs_top(self, tag, att, match, top)
(4)             def fz_xml_find_down(self, tag):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_find_down()`.
(12)                    Search the siblings of XML nodes starting with the first child
(12)                    of item looking for the first with the given tag.
(12)                    Return NULL if none found.
(8)                 """
(8)                 return _mupdf.FzXml_fz_xml_find_down(self, tag)
(4)             def fz_xml_find_down_match(self, tag, att, match):
(8)                 r"""
(8)                 Class-aware wrapper for `::fz_xml_find_down_match()`.
```

```
25501: (12)                     Search the siblings of XML nodes starting with the first child
25502: (12)                     of item looking for the first with the given tag (or any tag if
25503: (12)                     tag is NULL), and with a matching attribute.
25504: (12)                     Return NULL if none found.
25505: (8)                  """
25506: (8)                  return _mupdf.FzXml_fz_xml_find_down_match(self, tag, att, match)
25507: (4)              def fz_xml_find_match(self, tag, att, match):
25508: (8)                  r"""
25509: (8)                  Class-aware wrapper for `::fz_xml_find_match()`.
25510: (12)                     Search the siblings of XML nodes starting with item looking for
25511: (12)                     the first with the given tag (or any tag if tag is NULL), and
25512: (12)                     with a matching attribute.
25513: (12)                     Return NULL if none found.
25514: (8)                  """
25515: (8)                  return _mupdf.FzXml_fz_xml_find_match(self, tag, att, match)
25516: (4)              def fz_xml_find_next(self, tag):
25517: (8)                  r"""
25518: (8)                  Class-aware wrapper for `::fz_xml_find_next()`.
25519: (12)                     Search the siblings of XML nodes starting with the first sibling
25520: (12)                     of item looking for the first with the given tag.
25521: (12)                     Return NULL if none found.
25522: (8)                  """
25523: (8)                  return _mupdf.FzXml_fz_xml_find_next(self, tag)
25524: (4)              def fz_xml_find_next_dfs(self, tag, att, match):
25525: (8)                  r"""
25526: (8)                  Class-aware wrapper for `::fz_xml_find_next_dfs()`.
25527: (12)                     Perform a depth first search onwards from item, returning the
first
25528: (12)                     child that matches the given tag (or any tag if tag is NULL),
25529: (12)                     with the given attribute (if att is non NULL), that matches
25530: (12)                     match (if match is non NULL).
25531: (8)                  """
25532: (8)                  return _mupdf.FzXml_fz_xml_find_next_dfs(self, tag, att, match)
25533: (4)              def fz_xml_find_next_dfs_top(self, tag, att, match, top):
25534: (8)                  r"""
25535: (8)                  Class-aware wrapper for `::fz_xml_find_next_dfs_top()`.
25536: (12)                     Perform a depth first search onwards from item, returning the
first
25537: (12)                     child that matches the given tag (or any tag if tag is NULL),
25538: (12)                     with the given attribute (if att is non NULL), that matches
25539: (12)                     match (if match is non NULL). The search stops if it ever reaches
25540: (12)                     the top of the tree, or the declared 'top' item.
25541: (8)                  """
25542: (8)                  return _mupdf.FzXml_fz_xml_find_next_dfs_top(self, tag, att, match,
top)
25543: (4)              def fz_xml_find_next_match(self, tag, att, match):
25544: (8)                  r"""
25545: (8)                  Class-aware wrapper for `::fz_xml_find_next_match()`.
25546: (12)                     Search the siblings of XML nodes starting with the first sibling
25547: (12)                     of item looking for the first with the given tag (or any tag if
tag
25548: (12)                     is NULL), and with a matching attribute.
25549: (12)                     Return NULL if none found.
25550: (8)                  """
25551: (8)                  return _mupdf.FzXml_fz_xml_find_next_match(self, tag, att, match)
25552: (4)              def fz_xml_is_tag(self, name):
25553: (8)                  r"""
25554: (8)                  Class-aware wrapper for `::fz_xml_is_tag()`.
25555: (12)                     Return true if the tag name matches.
25556: (8)                  """
25557: (8)                  return _mupdf.FzXml_fz_xml_is_tag(self, name)
25558: (4)              def fz_xml_next(self):
25559: (8)                  r"""
25560: (8)                  Class-aware wrapper for `::fz_xml_next()`.
25561: (12)                     Return next sibling of XML node.
25562: (8)                  """
25563: (8)                  return _mupdf.FzXml_fz_xml_next(self)
25564: (4)              def fz_xml_prev(self):
25565: (8)                  r"""
```

```
25566: (8)                  Class-aware wrapper for `::fz_xml_prev()`.
25567: (12)                     Return previous sibling of XML node.
25568: (8)                  """
25569: (8)                  return _mupdf.FzXml_fz_xml_prev(self)
25570: (4)              def fz_xml_root(self):
25571: (8)                  r"""
25572: (8)                  Class-aware wrapper for `::fz_xml_root()`.
25573: (12)                     Return the topmost XML node of a document.
25574: (8)                  """
25575: (8)                  return _mupdf.FzXml_fz_xml_root(self)
25576: (4)              def fz_xml_tag(self):
25577: (8)                  r"""
25578: (8)                  Class-aware wrapper for `::fz_xml_tag()`.
25579: (12)                     Return tag of XML node. Return NULL for text nodes.
25580: (8)                  """
25581: (8)                  return _mupdf.FzXml_fz_xml_tag(self)
25582: (4)              def fz_xml_text(self):
25583: (8)                  r"""
25584: (8)                  Class-aware wrapper for `::fz_xml_text()`.
25585: (12)                     Return the text content of an XML node.
25586: (12)                     Return NULL if the node is a tag.
25587: (8)                  """
25588: (8)                  return _mupdf.FzXml_fz_xml_text(self)
25589: (4)              def fz_xml_up(self):
25590: (8)                  r"""
25591: (8)                  Class-aware wrapper for `::fz_xml_up()`.
25592: (12)                     Return parent of XML node.
25593: (8)                  """
25594: (8)                  return _mupdf.FzXml_fz_xml_up(self)
25595: (4)              def __init__(self, *args):
25596: (8)                  r"""
25597: (8)                  *Overload 1:*
25598: (8)                  Copy constructor using `fz_keep_xml()`.
25599: (8)                  |
25600: (8)                  *Overload 2:*
25601: (8)                  Default constructor, sets `m_internal` to null.
25602: (8)                  |
25603: (8)                  *Overload 3:*
25604: (8)                  Constructor using raw copy of pre-existing `::fz_xml`.
25605: (8)                  """
25606: (8)                  _mupdf.FzXml_swiginit(self, _mupdf.new_FzXml(*args))
25607: (4)              __swig_destroy__ = _mupdf.delete_FzXml
25608: (4)              def m_internal_value(self):
25609: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25610: (8)                  return _mupdf.FzXml_m_internal_value(self)
25611: (4)              m_internal = property(_mupdf.FzXml_m_internal_get,
_mupdf.FzXml_m_internal_set, doc=r""" Pointer to wrapped data.""")
25612: (4)              s_num_instances = property(_mupdf.FzXml_s_num_instances_get,
_mupdf.FzXml_s_num_instances_set)
25613: (0)          _mupdf.FzXml_swigregister(FzXml)
25614: (0)          class FzXmlDoc(object):
25615: (4)              r""" Wrapper class for struct `fz_xml_doc`. Not copyable or assignable."""
25616: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25617: (4)              __repr__ = _swig_repr
25618: (4)              def __init__(self, *args):
25619: (8)                  r"""
25620: (8)                  *Overload 1:*
25621: (8)                  Default constructor, sets `m_internal` to null.
25622: (8)                  |
25623: (8)                  *Overload 2:*
25624: (8)                  Constructor using raw copy of pre-existing `::fz_xml_doc`.
25625: (8)                  """
25626: (8)                  _mupdf.FzXmlDoc_swiginit(self, _mupdf.new_FzXmlDoc(*args))
25627: (4)              __swig_destroy__ = _mupdf.delete_FzXmlDoc
25628: (4)              def m_internal_value(self):
25629: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
```

```
25630: (8)                  return _mupdf.FzXmlDoc_m_internal_value(self)
25631: (4)              m_internal = property(_mupdf.FzXmlDoc_m_internal_get,
_mupdf.FzXmlDoc_m_internal_set, doc=r""" Pointer to wrapped data.""")
25632: (4)              s_num_instances = property(_mupdf.FzXmlDoc_s_num_instances_get,
_mupdf.FzXmlDoc_s_num_instances_set)
25633: (0)          _mupdf.FzXmlDoc_swigregister(FzXmlDoc)
25634: (0)          class FzZipWriter(object):
25635: (4)              r"""
25636: (4)              Wrapper class for struct `fz_zip_writer`. Not copyable or assignable.
25637: (4)              fz_zip_writer offers methods for creating and writing zip files.
25638: (4)              It can be seen as the reverse of the fz_archive zip
25639: (4)              implementation.
25640: (4)              """
25641: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25642: (4)              __repr__ = _swig_repr
25643: (4)              def fz_close_zip_writer(self):
25644: (8)                  r"""
25645: (8)                  Class-aware wrapper for `::fz_close_zip_writer()`.
25646: (12)                     Close the zip file for writing.
25647: (12)                     This flushes any pending data to the file. This can throw
25648: (12)                     exceptions.
25649: (8)                  """
25650: (8)                  return _mupdf.FzZipWriter_fz_close_zip_writer(self)
25651: (4)              def fz_write_zip_entry(self, name, buf, compress):
25652: (8)                  r"""
25653: (8)                  Class-aware wrapper for `::fz_write_zip_entry()`.
25654: (12)                     Given a buffer of data, (optionally) compress it, and add it to
25655: (12)                     the zip file with the given name.
25656: (8)                  """
25657: (8)                  return _mupdf.FzZipWriter_fz_write_zip_entry(self, name, buf,
compress)
25658: (4)              def __init__(self, *args):
25659: (8)                  r"""
25660: (8)                  *Overload 1:*
25661: (9)                   == Constructors.  Constructor using `fz_new_zip_writer()`.
25662: (16)                         Create a new zip writer that writes to a given file.
25663: (16)                         Open an archive using a seekable stream object rather than
25664: (16)                         opening a file or directory on disk.
25665: (8)                  |
25666: (8)                  *Overload 2:*
25667: (9)                   Constructor using `fz_new_zip_writer_with_output()`.
25668: (16)                         Create a new zip writer that writes to a given output stream.
25669: (16)                         Ownership of out passes in immediately upon calling this
function.
25670: (16)                         The caller should never drop the fz_output, even if this
function throws
25671: (16)                         an exception.
25672: (8)                  |
25673: (8)                  *Overload 3:*
25674: (9)                   Default constructor, sets `m_internal` to null.
25675: (8)                  |
25676: (8)                  *Overload 4:*
25677: (9)                   Constructor using raw copy of pre-existing `::fz_zip_writer`.
25678: (8)                  """
25679: (8)                  _mupdf.FzZipWriter_swiginit(self, _mupdf.new_FzZipWriter(*args))
25680: (4)              __swig_destroy__ = _mupdf.delete_FzZipWriter
25681: (4)              def m_internal_value(self):
25682: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25683: (8)                  return _mupdf.FzZipWriter_m_internal_value(self)
25684: (4)              m_internal = property(_mupdf.FzZipWriter_m_internal_get,
_mupdf.FzZipWriter_m_internal_set, doc=r""" Pointer to wrapped data.""")
25685: (4)              s_num_instances = property(_mupdf.FzZipWriter_s_num_instances_get,
_mupdf.FzZipWriter_s_num_instances_set)
25686: (0)          _mupdf.FzZipWriter_swigregister(FzZipWriter)
25687: (0)          class PdfAlertEvent(object):
25688: (4)              r""" Wrapper class for struct `pdf_alert_event`. Not copyable or
assignable."""
```

```
25689: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25690: (4)              __repr__ = _swig_repr
25691: (4)              def __init__(self, *args):
25692: (8)                  r"""
25693: (8)                  *Overload 1:*
25694: (8)                  Default constructor, sets `m_internal` to null.
25695: (8)                  |
25696: (8)                  *Overload 2:*
25697: (8)                  Constructor using raw copy of pre-existing `::pdf_alert_event`.
25698: (8)                  """
25699: (8)                  _mupdf.PdfAlertEvent_swiginit(self, _mupdf.new_PdfAlertEvent(*args))
25700: (4)              __swig_destroy__ = _mupdf.delete_PdfAlertEvent
25701: (4)              def m_internal_value(self):
25702: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
25703: (8)                  return _mupdf.PdfAlertEvent_m_internal_value(self)
25704: (4)              m_internal = property(_mupdf.PdfAlertEvent_m_internal_get,
_mupdf.PdfAlertEvent_m_internal_set, doc=r""" Pointer to wrapped data.""")
25705: (4)              s_num_instances = property(_mupdf.PdfAlertEvent_s_num_instances_get,
_mupdf.PdfAlertEvent_s_num_instances_set)
25706: (0)          _mupdf.PdfAlertEvent_swigregister(PdfAlertEvent)
25707: (0)          class PdfAnnot(object):
25708: (4)              r""" Wrapper class for struct `pdf_annot`."""
25709: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
25710: (4)              __repr__ = _swig_repr
25711: (4)              def pdf_add_annot_border_dash_item(self, length):
25712: (8)                  r""" Class-aware wrapper for `::pdf_add_annot_border_dash_item()`."""
25713: (8)                  return _mupdf.PdfAnnot_pdf_add_annot_border_dash_item(self, length)
25714: (4)              def pdf_add_annot_ink_list_stroke(self):
25715: (8)                  r""" Class-aware wrapper for `::pdf_add_annot_ink_list_stroke()`."""
25716: (8)                  return _mupdf.PdfAnnot_pdf_add_annot_ink_list_stroke(self)
25717: (4)              def pdf_add_annot_ink_list_stroke_vertex(self, p):
25718: (8)                  r""" Class-aware wrapper for
`::pdf_add_annot_ink_list_stroke_vertex()`."""
25719: (8)                  return _mupdf.PdfAnnot_pdf_add_annot_ink_list_stroke_vertex(self, p)
25720: (4)              def pdf_add_annot_quad_point(self, quad):
25721: (8)                  r""" Class-aware wrapper for `::pdf_add_annot_quad_point()`."""
25722: (8)                  return _mupdf.PdfAnnot_pdf_add_annot_quad_point(self, quad)
25723: (4)              def pdf_add_annot_vertex(self, p):
25724: (8)                  r""" Class-aware wrapper for `::pdf_add_annot_vertex()`."""
25725: (8)                  return _mupdf.PdfAnnot_pdf_add_annot_vertex(self, p)
25726: (4)              def pdf_annot_MK_BC(self, n, color):
25727: (8)                  r"""
25728: (8)                  Class-aware wrapper for `::pdf_annot_MK_BC()`.
25729: (8)                  This method has out-params. Python/C# wrappers look like:
25730: (12)                     `pdf_annot_MK_BC(float color[4])` => int n
25731: (8)                  """
25732: (8)                  return _mupdf.PdfAnnot_pdf_annot_MK_BC(self, n, color)
25733: (4)              def pdf_annot_MK_BC_rgb(self, rgb):
25734: (8)                  r""" Class-aware wrapper for `::pdf_annot_MK_BC_rgb()`."""
25735: (8)                  return _mupdf.PdfAnnot_pdf_annot_MK_BC_rgb(self, rgb)
25736: (4)              def pdf_annot_MK_BG(self, n, color):
25737: (8)                  r"""
25738: (8)                  Class-aware wrapper for `::pdf_annot_MK_BG()`.
25739: (8)                  This method has out-params. Python/C# wrappers look like:
25740: (12)                     `pdf_annot_MK_BG(float color[4])` => int n
25741: (8)                  """
25742: (8)                  return _mupdf.PdfAnnot_pdf_annot_MK_BG(self, n, color)
25743: (4)              def pdf_annot_MK_BG_rgb(self, rgb):
25744: (8)                  r""" Class-aware wrapper for `::pdf_annot_MK_BG_rgb()`."""
25745: (8)                  return _mupdf.PdfAnnot_pdf_annot_MK_BG_rgb(self, rgb)
25746: (4)              def pdf_annot_active(self):
25747: (8)                  r""" Class-aware wrapper for `::pdf_annot_active()`."""
25748: (8)                  return _mupdf.PdfAnnot_pdf_annot_active(self)
25749: (4)              def pdf_annot_ap(self):
25750: (8)                  r""" Class-aware wrapper for `::pdf_annot_ap()`."""
25751: (8)                  return _mupdf.PdfAnnot_pdf_annot_ap(self)
```

```
(4)             def pdf_annot_author(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_author()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_author(self)
(4)             def pdf_annot_border(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border(self)
(4)             def pdf_annot_border_dash_count(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border_dash_count()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_dash_count(self)
(4)             def pdf_annot_border_dash_item(self, i):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border_dash_item()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_dash_item(self, i)
(4)             def pdf_annot_border_effect(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border_effect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_effect(self)
(4)             def pdf_annot_border_effect_intensity(self):
(8)                 r""" Class-aware wrapper for
`::pdf_annot_border_effect_intensity()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_effect_intensity(self)
(4)             def pdf_annot_border_style(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_style(self)
(4)             def pdf_annot_border_width(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_border_width()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_border_width(self)
(4)             def pdf_annot_color(self, n, color):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_annot_color()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_annot_color(float color[4])` => int n
(8)                 """
(8)                 return _mupdf.PdfAnnot_pdf_annot_color(self, n, color)
(4)             def pdf_annot_contents(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_contents()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_contents(self)
(4)             def pdf_annot_creation_date(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_creation_date()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_creation_date(self)
(4)             def pdf_annot_default_appearance(self, font, size, n, color):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_annot_default_appearance()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_annot_default_appearance(float color[4])` => `(const char
*font, float size, int n)`
(8)                 """
(8)                 return _mupdf.PdfAnnot_pdf_annot_default_appearance(self, font, size,
n, color)
(4)             def pdf_annot_ensure_local_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_ensure_local_xref()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_ensure_local_xref(self)
(4)             def pdf_annot_event_blur(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_blur()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_blur(self)
(4)             def pdf_annot_event_down(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_down()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_down(self)
(4)             def pdf_annot_event_enter(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_enter()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_enter(self)
(4)             def pdf_annot_event_exit(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_exit()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_exit(self)
(4)             def pdf_annot_event_focus(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_focus()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_focus(self)
(4)             def pdf_annot_event_page_close(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_page_close()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_page_close(self)
(4)             def pdf_annot_event_page_invisible(self):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_page_invisible()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_page_invisible(self)
(4)             def pdf_annot_event_page_open(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_page_open()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_page_open(self)
(4)             def pdf_annot_event_page_visible(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_page_visible()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_page_visible(self)
(4)             def pdf_annot_event_up(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_event_up()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_event_up(self)
(4)             def pdf_annot_field_flags(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_field_flags()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_field_flags(self)
(4)             def pdf_annot_field_label(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_field_label()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_field_label(self)
(4)             def pdf_annot_field_value(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_field_value()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_field_value(self)
(4)             def pdf_annot_filespec(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_filespec()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_filespec(self)
(4)             def pdf_annot_flags(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_flags()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_flags(self)
(4)             def pdf_annot_has_author(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_author()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_author(self)
(4)             def pdf_annot_has_border(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_border()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_border(self)
(4)             def pdf_annot_has_border_effect(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_border_effect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_border_effect(self)
(4)             def pdf_annot_has_filespec(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_filespec()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_filespec(self)
(4)             def pdf_annot_has_icon_name(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_icon_name()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_icon_name(self)
(4)             def pdf_annot_has_ink_list(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_ink_list()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_ink_list(self)
(4)             def pdf_annot_has_intent(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_intent()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_intent(self)
(4)             def pdf_annot_has_interior_color(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_interior_color()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_interior_color(self)
(4)             def pdf_annot_has_line(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_line()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_line(self)
(4)             def pdf_annot_has_line_ending_styles(self):
(8)                 r""" Class-aware wrapper for
`::pdf_annot_has_line_ending_styles()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_line_ending_styles(self)
(4)             def pdf_annot_has_open(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_open()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_open(self)
(4)             def pdf_annot_has_quad_points(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_quad_points()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_quad_points(self)
(4)             def pdf_annot_has_quadding(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_quadding()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_quadding(self)
(4)             def pdf_annot_has_rect(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_rect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_rect(self)
```

```
(4)             def pdf_annot_has_vertices(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_has_vertices()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_has_vertices(self)
(4)             def pdf_annot_hidden_for_editing(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_hidden_for_editing()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_hidden_for_editing(self)
(4)             def pdf_annot_hot(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_hot()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_hot(self)
(4)             def pdf_annot_icon_name(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_icon_name()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_icon_name(self)
(4)             def pdf_annot_ink_list_count(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_ink_list_count()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_ink_list_count(self)
(4)             def pdf_annot_ink_list_stroke_count(self, i):
(8)                 r""" Class-aware wrapper for `::pdf_annot_ink_list_stroke_count()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_ink_list_stroke_count(self, i)
(4)             def pdf_annot_ink_list_stroke_vertex(self, i, k):
(8)                 r""" Class-aware wrapper for
`::pdf_annot_ink_list_stroke_vertex()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_ink_list_stroke_vertex(self, i, k)
(4)             def pdf_annot_intent(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_intent()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_intent(self)
(4)             def pdf_annot_interior_color(self, n, color):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_annot_interior_color()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_annot_interior_color(float color[4])` => int n
(8)                 """
(8)                 return _mupdf.PdfAnnot_pdf_annot_interior_color(self, n, color)
(4)             def pdf_annot_is_open(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_is_open()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_is_open(self)
(4)             def pdf_annot_is_standard_stamp(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_is_standard_stamp()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_is_standard_stamp(self)
(4)             def pdf_annot_line(self, a, b):
(8)                 r""" Class-aware wrapper for `::pdf_annot_line()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_line(self, a, b)
(4)             def pdf_annot_line_end_style(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_line_end_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_line_end_style(self)
(4)             def pdf_annot_line_ending_styles(self, start_style, end_style):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_annot_line_ending_styles()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_annot_line_ending_styles()` => `(enum pdf_line_ending
start_style, enum pdf_line_ending end_style)`
(8)                 """
(8)                 return _mupdf.PdfAnnot_pdf_annot_line_ending_styles(self, start_style,
end_style)
(4)             def pdf_annot_line_start_style(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_line_start_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_line_start_style(self)
(4)             def pdf_annot_modification_date(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_modification_date()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_modification_date(self)
(4)             def pdf_annot_needs_resynthesis(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_needs_resynthesis()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_needs_resynthesis(self)
(4)             def pdf_annot_obj(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_obj()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_obj(self)
(4)             def pdf_annot_opacity(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_opacity()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_opacity(self)
(4)             def pdf_annot_page(self):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_annot_page()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_page(self)
(4)             def pdf_annot_pop_and_discard_local_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_pop_and_discard_local_xref()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_pop_and_discard_local_xref(self)
(4)             def pdf_annot_pop_local_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_pop_local_xref()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_pop_local_xref(self)
(4)             def pdf_annot_popup(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_popup()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_popup(self)
(4)             def pdf_annot_push_local_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_push_local_xref()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_push_local_xref(self)
(4)             def pdf_annot_quad_point(self, i):
(8)                 r""" Class-aware wrapper for `::pdf_annot_quad_point()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_quad_point(self, i)
(4)             def pdf_annot_quad_point_count(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_quad_point_count()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_quad_point_count(self)
(4)             def pdf_annot_quadding(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_quadding()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_quadding(self)
(4)             def pdf_annot_rect(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_rect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_rect(self)
(4)             def pdf_annot_request_resynthesis(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_request_resynthesis()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_request_resynthesis(self)
(4)             def pdf_annot_request_synthesis(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_request_synthesis()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_request_synthesis(self)
(4)             def pdf_annot_transform(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_transform()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_transform(self)
(4)             def pdf_annot_type(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_type()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_type(self)
(4)             def pdf_annot_vertex(self, i):
(8)                 r""" Class-aware wrapper for `::pdf_annot_vertex()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_vertex(self, i)
(4)             def pdf_annot_vertex_count(self):
(8)                 r""" Class-aware wrapper for `::pdf_annot_vertex_count()`."""
(8)                 return _mupdf.PdfAnnot_pdf_annot_vertex_count(self)
(4)             def pdf_apply_redaction(self, opts):
(8)                 r""" Class-aware wrapper for `::pdf_apply_redaction()`."""
(8)                 return _mupdf.PdfAnnot_pdf_apply_redaction(self, opts)
(4)             def pdf_bound_annot(self):
(8)                 r""" Class-aware wrapper for `::pdf_bound_annot()`."""
(8)                 return _mupdf.PdfAnnot_pdf_bound_annot(self)
(4)             def pdf_bound_widget(self):
(8)                 r""" Class-aware wrapper for `::pdf_bound_widget()`."""
(8)                 return _mupdf.PdfAnnot_pdf_bound_widget(self)
(4)             def pdf_choice_widget_is_multiselect(self):
(8)                 r""" Class-aware wrapper for `::pdf_choice_widget_is_multiselect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_choice_widget_is_multiselect(self)
(4)             def pdf_choice_widget_options(self, exportval, opts):
(8)                 r""" Class-aware wrapper for `::pdf_choice_widget_options()`."""
(8)                 return _mupdf.PdfAnnot_pdf_choice_widget_options(self, exportval, opts)
(4)             def pdf_choice_widget_options2(self, exportval):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_choice_widget_options2()`.   Swig-friendly wrapper for pdf_choice_widget_options(), returns the
(8)                 options directly in a vector.
(8)                 """
(8)                 return _mupdf.PdfAnnot_pdf_choice_widget_options2(self, exportval)
```

```
26017: (4)              def pdf_choice_widget_set_value(self, n, opts):
26018: (8)                  r""" Class-aware wrapper for `::pdf_choice_widget_set_value()`."""
26019: (8)                  return _mupdf.PdfAnnot_pdf_choice_widget_set_value(self, n, opts)
26020: (4)              def pdf_choice_widget_value(self, opts):
26021: (8)                  r""" Class-aware wrapper for `::pdf_choice_widget_value()`."""
26022: (8)                  return _mupdf.PdfAnnot_pdf_choice_widget_value(self, opts)
26023: (4)              def pdf_clear_annot_border_dash(self):
26024: (8)                  r""" Class-aware wrapper for `::pdf_clear_annot_border_dash()`."""
26025: (8)                  return _mupdf.PdfAnnot_pdf_clear_annot_border_dash(self)
26026: (4)              def pdf_clear_annot_ink_list(self):
26027: (8)                  r""" Class-aware wrapper for `::pdf_clear_annot_ink_list()`."""
26028: (8)                  return _mupdf.PdfAnnot_pdf_clear_annot_ink_list(self)
26029: (4)              def pdf_clear_annot_quad_points(self):
26030: (8)                  r""" Class-aware wrapper for `::pdf_clear_annot_quad_points()`."""
26031: (8)                  return _mupdf.PdfAnnot_pdf_clear_annot_quad_points(self)
26032: (4)              def pdf_clear_annot_vertices(self):
26033: (8)                  r""" Class-aware wrapper for `::pdf_clear_annot_vertices()`."""
26034: (8)                  return _mupdf.PdfAnnot_pdf_clear_annot_vertices(self)
26035: (4)              def pdf_clear_signature(self):
26036: (8)                  r""" Class-aware wrapper for `::pdf_clear_signature()`."""
26037: (8)                  return _mupdf.PdfAnnot_pdf_clear_signature(self)
26038: (4)              def pdf_dirty_annot(self):
26039: (8)                  r""" Class-aware wrapper for `::pdf_dirty_annot()`."""
26040: (8)                  return _mupdf.PdfAnnot_pdf_dirty_annot(self)
26041: (4)              def pdf_edit_text_field_value(self, value, change, selStart, selEnd,
newvalue):
26042: (8)                  r"""
26043: (8)                  Class-aware wrapper for `::pdf_edit_text_field_value()`.
26044: (8)                  This method has out-params. Python/C# wrappers look like:
26045: (12)                     `pdf_edit_text_field_value(const char *value, const char *change)`
=> `(int, int selStart, int selEnd, char *newvalue)`
26046: (8)                  """
26047: (8)                  return _mupdf.PdfAnnot_pdf_edit_text_field_value(self, value, change,
selStart, selEnd, newvalue)
26048: (4)              def pdf_get_widget_editing_state(self):
26049: (8)                  r""" Class-aware wrapper for `::pdf_get_widget_editing_state()`."""
26050: (8)                  return _mupdf.PdfAnnot_pdf_get_widget_editing_state(self)
26051: (4)              def pdf_new_display_list_from_annot(self):
26052: (8)                  r""" Class-aware wrapper for `::pdf_new_display_list_from_annot()`."""
26053: (8)                  return _mupdf.PdfAnnot_pdf_new_display_list_from_annot(self)
26054: (4)              def pdf_new_pixmap_from_annot(self, ctm, cs, seps, alpha):
26055: (8)                  r""" Class-aware wrapper for `::pdf_new_pixmap_from_annot()`."""
26056: (8)                  return _mupdf.PdfAnnot_pdf_new_pixmap_from_annot(self, ctm, cs, seps,
alpha)
26057: (4)              def pdf_next_annot(self):
26058: (8)                  r""" Class-aware wrapper for `::pdf_next_annot()`."""
26059: (8)                  return _mupdf.PdfAnnot_pdf_next_annot(self)
26060: (4)              def pdf_next_widget(self):
26061: (8)                  r""" Class-aware wrapper for `::pdf_next_widget()`."""
26062: (8)                  return _mupdf.PdfAnnot_pdf_next_widget(self)
26063: (4)              def pdf_run_annot(self, dev, ctm, cookie):
26064: (8)                  r""" Class-aware wrapper for `::pdf_run_annot()`."""
26065: (8)                  return _mupdf.PdfAnnot_pdf_run_annot(self, dev, ctm, cookie)
26066: (4)              def pdf_set_annot_active(self, active):
26067: (8)                  r""" Class-aware wrapper for `::pdf_set_annot_active()`."""
26068: (8)                  return _mupdf.PdfAnnot_pdf_set_annot_active(self, active)
26069: (4)              def pdf_set_annot_appearance(self, appearance, state, ctm, bbox, res,
contents):
26070: (8)                  r""" Class-aware wrapper for `::pdf_set_annot_appearance()`."""
26071: (8)                  return _mupdf.PdfAnnot_pdf_set_annot_appearance(self, appearance,
state, ctm, bbox, res, contents)
26072: (4)              def pdf_set_annot_appearance_from_display_list(self, appearance, state,
ctm, list):
26073: (8)                  r""" Class-aware wrapper for
`::pdf_set_annot_appearance_from_display_list()`."""
26074: (8)                  return
_mupdf.PdfAnnot_pdf_set_annot_appearance_from_display_list(self, appearance, state, ctm, list)
26075: (4)              def pdf_set_annot_author(self, author):
26076: (8)                  r""" Class-aware wrapper for `::pdf_set_annot_author()`."""
```

```
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_author(self, author)
(4)             def pdf_set_annot_border(self, width):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_border()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_border(self, width)
(4)             def pdf_set_annot_border_effect(self, effect):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_border_effect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_border_effect(self, effect)
(4)             def pdf_set_annot_border_effect_intensity(self, intensity):
(8)                 r""" Class-aware wrapper for
`::pdf_set_annot_border_effect_intensity()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_border_effect_intensity(self,
intensity)
(4)             def pdf_set_annot_border_style(self, style):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_border_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_border_style(self, style)
(4)             def pdf_set_annot_border_width(self, width):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_border_width()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_border_width(self, width)
(4)             def pdf_set_annot_color(self, n, color):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_color()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_color(self, n, color)
(4)             def pdf_set_annot_contents(self, text):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_contents()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_contents(self, text)
(4)             def pdf_set_annot_creation_date(self, time):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_creation_date()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_creation_date(self, time)
(4)             def pdf_set_annot_default_appearance(self, font, size, n, color):
(8)                 r""" Class-aware wrapper for
`::pdf_set_annot_default_appearance()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_default_appearance(self, font,
size, n, color)
(4)             def pdf_set_annot_filespec(self, obj):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_filespec()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_filespec(self, obj)
(4)             def pdf_set_annot_flags(self, flags):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_flags()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_flags(self, flags)
(4)             def pdf_set_annot_hidden_for_editing(self, hidden):
(8)                 r""" Class-aware wrapper for
`::pdf_set_annot_hidden_for_editing()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_hidden_for_editing(self, hidden)
(4)             def pdf_set_annot_hot(self, hot):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_hot()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_hot(self, hot)
(4)             def pdf_set_annot_icon_name(self, name):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_icon_name()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_icon_name(self, name)
(4)             def pdf_set_annot_ink_list(self, n, count, v):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_ink_list()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_ink_list(self, n, count, v)
(4)             def pdf_set_annot_intent(self, it):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_intent()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_intent(self, it)
(4)             def pdf_set_annot_interior_color(self, n, color):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_interior_color()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_interior_color(self, n, color)
(4)             def pdf_set_annot_is_open(self, is_open):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_is_open()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_is_open(self, is_open)
(4)             def pdf_set_annot_language(self, lang):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_language()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_language(self, lang)
(4)             def pdf_set_annot_line(self, a, b):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_line()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_line(self, a, b)
(4)             def pdf_set_annot_line_end_style(self, e):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_line_end_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_line_end_style(self, e)
```

```
(4)             def pdf_set_annot_line_ending_styles(self, start_style, end_style):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_line_ending_styles()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_line_ending_styles(self, start_style, end_style)
(4)             def pdf_set_annot_line_start_style(self, s):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_line_start_style()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_line_start_style(self, s)
(4)             def pdf_set_annot_modification_date(self, time):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_modification_date()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_modification_date(self, time)
(4)             def pdf_set_annot_opacity(self, opacity):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_opacity()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_opacity(self, opacity)
(4)             def pdf_set_annot_popup(self, rect):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_popup()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_popup(self, rect)
(4)             def pdf_set_annot_quad_points(self, n, qv):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_quad_points()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_quad_points(self, n, qv)
(4)             def pdf_set_annot_quadding(self, q):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_quadding()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_quadding(self, q)
(4)             def pdf_set_annot_rect(self, rect):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_rect()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_rect(self, rect)
(4)             def pdf_set_annot_resynthesised(self):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_resynthesised()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_resynthesised(self)
(4)             def pdf_set_annot_stamp_image(self, image):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_stamp_image()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_stamp_image(self, image)
(4)             def pdf_set_annot_vertex(self, i, p):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_vertex()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_vertex(self, i, p)
(4)             def pdf_set_annot_vertices(self, n, v):
(8)                 r""" Class-aware wrapper for `::pdf_set_annot_vertices()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_annot_vertices(self, n, v)
(4)             def pdf_set_choice_field_value(self, value):
(8)                 r""" Class-aware wrapper for `::pdf_set_choice_field_value()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_choice_field_value(self, value)
(4)             def pdf_set_text_field_value(self, value):
(8)                 r""" Class-aware wrapper for `::pdf_set_text_field_value()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_text_field_value(self, value)
(4)             def pdf_set_widget_editing_state(self, editing):
(8)                 r""" Class-aware wrapper for `::pdf_set_widget_editing_state()`."""
(8)                 return _mupdf.PdfAnnot_pdf_set_widget_editing_state(self, editing)
(4)             def pdf_sign_signature(self, signer, appearance_flags, graphic, reason, location):
(8)                 r""" Class-aware wrapper for `::pdf_sign_signature()`."""
(8)                 return _mupdf.PdfAnnot_pdf_sign_signature(self, signer, appearance_flags, graphic, reason, location)
(4)             def pdf_sign_signature_with_appearance(self, signer, date, disp_list):
(8)                 r""" Class-aware wrapper for `::pdf_sign_signature_with_appearance()`."""
(8)                 return _mupdf.PdfAnnot_pdf_sign_signature_with_appearance(self, signer, date, disp_list)
(4)             def pdf_text_widget_format(self):
(8)                 r""" Class-aware wrapper for `::pdf_text_widget_format()`."""
(8)                 return _mupdf.PdfAnnot_pdf_text_widget_format(self)
(4)             def pdf_text_widget_max_len(self):
(8)                 r""" Class-aware wrapper for `::pdf_text_widget_max_len()`."""
(8)                 return _mupdf.PdfAnnot_pdf_text_widget_max_len(self)
(4)             def pdf_toggle_widget(self):
(8)                 r""" Class-aware wrapper for `::pdf_toggle_widget()`."""
(8)                 return _mupdf.PdfAnnot_pdf_toggle_widget(self)
(4)             def pdf_update_annot(self):
(8)                 r""" Class-aware wrapper for `::pdf_update_annot()`."""
(8)                 return _mupdf.PdfAnnot_pdf_update_annot(self)
```

```
26204: (4)              def pdf_update_widget(self):
26205: (8)                  r""" Class-aware wrapper for `::pdf_update_widget()`."""
26206: (8)                  return _mupdf.PdfAnnot_pdf_update_widget(self)
26207: (4)              def pdf_validate_signature(self):
26208: (8)                  r""" Class-aware wrapper for `::pdf_validate_signature()`."""
26209: (8)                  return _mupdf.PdfAnnot_pdf_validate_signature(self)
26210: (4)              def pdf_widget_is_readonly(self):
26211: (8)                  r""" Class-aware wrapper for `::pdf_widget_is_readonly()`."""
26212: (8)                  return _mupdf.PdfAnnot_pdf_widget_is_readonly(self)
26213: (4)              def pdf_widget_is_signed(self):
26214: (8)                  r""" Class-aware wrapper for `::pdf_widget_is_signed()`."""
26215: (8)                  return _mupdf.PdfAnnot_pdf_widget_is_signed(self)
26216: (4)              def pdf_widget_type(self):
26217: (8)                  r""" Class-aware wrapper for `::pdf_widget_type()`."""
26218: (8)                  return _mupdf.PdfAnnot_pdf_widget_type(self)
26219: (4)              def __init__(self, *args):
26220: (8)                  r"""
26221: (8)                  *Overload 1:*
26222: (8)                  Copy constructor using `pdf_keep_annot()`.
26223: (8)                  |
26224: (8)                  *Overload 2:*
26225: (8)                  Constructor using raw copy of pre-existing `::pdf_annot`.
26226: (8)                  |
26227: (8)                  *Overload 3:*
26228: (8)                  Constructor using raw copy of pre-existing `::pdf_annot`.
26229: (8)                  """
26230: (8)                  _mupdf.PdfAnnot_swiginit(self, _mupdf.new_PdfAnnot(*args))
26231: (4)              __swig_destroy__ = _mupdf.delete_PdfAnnot
26232: (4)              def m_internal_value(self):
26233: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26234: (8)                  return _mupdf.PdfAnnot_m_internal_value(self)
26235: (4)              m_internal = property(_mupdf.PdfAnnot_m_internal_get,
_mupdf.PdfAnnot_m_internal_set, doc=r""" Pointer to wrapped data.""")
26236: (4)              s_num_instances = property(_mupdf.PdfAnnot_s_num_instances_get,
_mupdf.PdfAnnot_s_num_instances_set)
26237: (0)          _mupdf.PdfAnnot_swigregister(PdfAnnot)
26238: (0)          class PdfCleanOptions(object):
26239: (4)              r""" Wrapper class for struct `pdf_clean_options`."""
26240: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26241: (4)              __repr__ = _swig_repr
26242: (4)              def write_opwd_utf8_set(self, text):
26243: (8)                  r""" Copies <text> into write.opwd_utf8[]."""
26244: (8)                  return _mupdf.PdfCleanOptions_write_opwd_utf8_set(self, text)
26245: (4)              def write_upwd_utf8_set(self, text):
26246: (8)                  r""" Copies <text> into upwd_utf8[]."""
26247: (8)                  return _mupdf.PdfCleanOptions_write_upwd_utf8_set(self, text)
26248: (4)              def __init__(self, *args):
26249: (8)                  r"""
26250: (8)                  *Overload 1:*
26251: (8)                  Default constructor, makes copy of pdf_default_write_options.
26252: (8)                  |
26253: (8)                  *Overload 2:*
26254: (8)                  Copy constructor using raw memcopy().
26255: (8)                  |
26256: (8)                  *Overload 3:*
26257: (8)                  Constructor using raw copy of pre-existing `::pdf_clean_options`.
26258: (8)                  |
26259: (8)                  *Overload 4:*
26260: (8)                  Constructor using raw copy of pre-existing `::pdf_clean_options`.
26261: (8)                  """
26262: (8)                  _mupdf.PdfCleanOptions_swiginit(self,
_mupdf.new_PdfCleanOptions(*args))
26263: (4)              def internal(self, *args):
26264: (8)                  r"""
26265: (8)                  *Overload 1:*
26266: (8)                  Access as underlying struct.
26267: (8)                  |
```

```
26268: (8)                  *Overload 2:*
26269: (8)                  Access as underlying struct.
26270: (8)                  """
26271: (8)                  return _mupdf.PdfCleanOptions_internal(self, *args)
26272: (4)              __swig_destroy__ = _mupdf.delete_PdfCleanOptions
26273: (4)              write = property(_mupdf.PdfCleanOptions_write_get,
_mupdf.PdfCleanOptions_write_set)
26274: (4)              image = property(_mupdf.PdfCleanOptions_image_get,
_mupdf.PdfCleanOptions_image_set)
26275: (4)              subset_fonts = property(_mupdf.PdfCleanOptions_subset_fonts_get,
_mupdf.PdfCleanOptions_subset_fonts_set)
26276: (4)              s_num_instances = property(_mupdf.PdfCleanOptions_s_num_instances_get,
_mupdf.PdfCleanOptions_s_num_instances_set)
26277: (4)              def to_string(self):
26278: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
26279: (8)                  return _mupdf.PdfCleanOptions_to_string(self)
26280: (4)              def __eq__(self, rhs):
26281: (8)                  r""" Comparison method."""
26282: (8)                  return _mupdf.PdfCleanOptions___eq__(self, rhs)
26283: (4)              def __ne__(self, rhs):
26284: (8)                  r""" Comparison method."""
26285: (8)                  return _mupdf.PdfCleanOptions___ne__(self, rhs)
26286: (0)          _mupdf.PdfCleanOptions_swigregister(PdfCleanOptions)
26287: (0)          class PdfCmap(object):
26288: (4)              r""" Wrapper class for struct `pdf_cmap`."""
26289: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26290: (4)              __repr__ = _swig_repr
26291: (4)              def pdf_add_codespace(self, low, high, n):
26292: (8)                  r""" Class-aware wrapper for `::pdf_add_codespace()`."""
26293: (8)                  return _mupdf.PdfCmap_pdf_add_codespace(self, low, high, n)
26294: (4)              def pdf_cmap_size(self):
26295: (8)                  r""" Class-aware wrapper for `::pdf_cmap_size()`."""
26296: (8)                  return _mupdf.PdfCmap_pdf_cmap_size(self)
26297: (4)              def pdf_cmap_wmode(self):
26298: (8)                  r""" Class-aware wrapper for `::pdf_cmap_wmode()`."""
26299: (8)                  return _mupdf.PdfCmap_pdf_cmap_wmode(self)
26300: (4)              def pdf_decode_cmap(self, s, e, cpt):
26301: (8)                  r"""
26302: (8)                  Class-aware wrapper for `::pdf_decode_cmap()`.
26303: (8)                  This method has out-params. Python/C# wrappers look like:
26304: (12)                     `pdf_decode_cmap(unsigned char *s, unsigned char *e)` => `(int,
unsigned int cpt)`
26305: (8)                  """
26306: (8)                  return _mupdf.PdfCmap_pdf_decode_cmap(self, s, e, cpt)
26307: (4)              def pdf_lookup_cmap(self, cpt):
26308: (8)                  r""" Class-aware wrapper for `::pdf_lookup_cmap()`."""
26309: (8)                  return _mupdf.PdfCmap_pdf_lookup_cmap(self, cpt)
26310: (4)              def pdf_lookup_cmap_full(self, cpt, out):
26311: (8)                  r"""
26312: (8)                  Class-aware wrapper for `::pdf_lookup_cmap_full()`.
26313: (8)                  This method has out-params. Python/C# wrappers look like:
26314: (12)                     `pdf_lookup_cmap_full(unsigned int cpt)` => `(int, int out)`
26315: (8)                  """
26316: (8)                  return _mupdf.PdfCmap_pdf_lookup_cmap_full(self, cpt, out)
26317: (4)              def pdf_map_one_to_many(self, one, many, len):
26318: (8)                  r"""
26319: (8)                  Class-aware wrapper for `::pdf_map_one_to_many()`.
26320: (8)                  This method has out-params. Python/C# wrappers look like:
26321: (12)                     `pdf_map_one_to_many(unsigned int one, size_t len)` => int many
26322: (8)                  """
26323: (8)                  return _mupdf.PdfCmap_pdf_map_one_to_many(self, one, many, len)
26324: (4)              def pdf_map_range_to_range(self, srclo, srchi, dstlo):
26325: (8)                  r""" Class-aware wrapper for `::pdf_map_range_to_range()`."""
26326: (8)                  return _mupdf.PdfCmap_pdf_map_range_to_range(self, srclo, srchi,
dstlo)
26327: (4)              def pdf_set_cmap_wmode(self, wmode):
26328: (8)                  r""" Class-aware wrapper for `::pdf_set_cmap_wmode()`."""
```

```
26329: (8)                  return _mupdf.PdfCmap_pdf_set_cmap_wmode(self, wmode)
26330: (4)              def pdf_set_usecmap(self, usecmap):
26331: (8)                  r""" Class-aware wrapper for `::pdf_set_usecmap()`."""
26332: (8)                  return _mupdf.PdfCmap_pdf_set_usecmap(self, usecmap)
26333: (4)              def pdf_sort_cmap(self):
26334: (8)                  r""" Class-aware wrapper for `::pdf_sort_cmap()`."""
26335: (8)                  return _mupdf.PdfCmap_pdf_sort_cmap(self)
26336: (4)              def __init__(self, *args):
26337: (8)                  r"""
26338: (8)                  *Overload 1:*
26339: (8)                  == Constructors.  Constructor using `pdf_new_cmap()`.
26340: (8)                  |
26341: (8)                  *Overload 2:*
26342: (8)                  Constructor using `pdf_new_identity_cmap()`.
26343: (8)                  |
26344: (8)                  *Overload 3:*
26345: (8)                  Copy constructor using `pdf_keep_cmap()`.
26346: (8)                  |
26347: (8)                  *Overload 4:*
26348: (8)                  Constructor using raw copy of pre-existing `::pdf_cmap`.
26349: (8)                  """
26350: (8)                  _mupdf.PdfCmap_swiginit(self, _mupdf.new_PdfCmap(*args))
26351: (4)              __swig_destroy__ = _mupdf.delete_PdfCmap
26352: (4)              def m_internal_value(self):
26353: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26354: (8)                  return _mupdf.PdfCmap_m_internal_value(self)
26355: (4)              m_internal = property(_mupdf.PdfCmap_m_internal_get,
_mupdf.PdfCmap_m_internal_set, doc=r""" Pointer to wrapped data.""")
26356: (4)              s_num_instances = property(_mupdf.PdfCmap_s_num_instances_get,
_mupdf.PdfCmap_s_num_instances_set)
26357: (0)          _mupdf.PdfCmap_swigregister(PdfCmap)
26358: (0)          class PdfColorFilterOptions(object):
26359: (4)              r""" Wrapper class for struct `pdf_color_filter_options`. Not copyable or
assignable."""
26360: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26361: (4)              __repr__ = _swig_repr
26362: (4)              def __init__(self, *args):
26363: (8)                  r"""
26364: (8)                  *Overload 1:*
26365: (8)                  Default constructor, sets `m_internal` to null.
26366: (8)                  |
26367: (8)                  *Overload 2:*
26368: (8)                  Constructor using raw copy of pre-existing
`::pdf_color_filter_options`.
26369: (8)                  """
26370: (8)                  _mupdf.PdfColorFilterOptions_swiginit(self,
_mupdf.new_PdfColorFilterOptions(*args))
26371: (4)              __swig_destroy__ = _mupdf.delete_PdfColorFilterOptions
26372: (4)              def m_internal_value(self):
26373: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26374: (8)                  return _mupdf.PdfColorFilterOptions_m_internal_value(self)
26375: (4)              m_internal = property(_mupdf.PdfColorFilterOptions_m_internal_get,
_mupdf.PdfColorFilterOptions_m_internal_set, doc=r""" Pointer to wrapped data.""")
26376: (4)              s_num_instances =
property(_mupdf.PdfColorFilterOptions_s_num_instances_get,
_mupdf.PdfColorFilterOptions_s_num_instances_set)
26377: (0)          _mupdf.PdfColorFilterOptions_swigregister(PdfColorFilterOptions)
26378: (0)          class PdfCrypt(object):
26379: (4)              r""" Wrapper class for struct `pdf_crypt`. Not copyable or assignable."""
26380: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26381: (4)              __repr__ = _swig_repr
26382: (4)              def pdf_crypt_encrypt_metadata(self):
26383: (8)                  r""" Class-aware wrapper for `::pdf_crypt_encrypt_metadata()`."""
26384: (8)                  return _mupdf.PdfCrypt_pdf_crypt_encrypt_metadata(self)
26385: (4)              def pdf_crypt_key(self):
```

```
26386: (8)                  r""" Class-aware wrapper for `::pdf_crypt_key()`."""
26387: (8)                  return _mupdf.PdfCrypt_pdf_crypt_key(self)
26388: (4)              def pdf_crypt_length(self):
26389: (8)                  r""" Class-aware wrapper for `::pdf_crypt_length()`."""
26390: (8)                  return _mupdf.PdfCrypt_pdf_crypt_length(self)
26391: (4)              def pdf_crypt_method(self):
26392: (8)                  r""" Class-aware wrapper for `::pdf_crypt_method()`."""
26393: (8)                  return _mupdf.PdfCrypt_pdf_crypt_method(self)
26394: (4)              def pdf_crypt_obj(self, obj, num, gen):
26395: (8)                  r""" Class-aware wrapper for `::pdf_crypt_obj()`."""
26396: (8)                  return _mupdf.PdfCrypt_pdf_crypt_obj(self, obj, num, gen)
26397: (4)              def pdf_crypt_owner_encryption(self):
26398: (8)                  r""" Class-aware wrapper for `::pdf_crypt_owner_encryption()`."""
26399: (8)                  return _mupdf.PdfCrypt_pdf_crypt_owner_encryption(self)
26400: (4)              def pdf_crypt_owner_password(self):
26401: (8)                  r""" Class-aware wrapper for `::pdf_crypt_owner_password()`."""
26402: (8)                  return _mupdf.PdfCrypt_pdf_crypt_owner_password(self)
26403: (4)              def pdf_crypt_permissions(self):
26404: (8)                  r""" Class-aware wrapper for `::pdf_crypt_permissions()`."""
26405: (8)                  return _mupdf.PdfCrypt_pdf_crypt_permissions(self)
26406: (4)              def pdf_crypt_permissions_encryption(self):
26407: (8)                  r""" Class-aware wrapper for
`::pdf_crypt_permissions_encryption()`."""
26408: (8)                  return _mupdf.PdfCrypt_pdf_crypt_permissions_encryption(self)
26409: (4)              def pdf_crypt_revision(self):
26410: (8)                  r""" Class-aware wrapper for `::pdf_crypt_revision()`."""
26411: (8)                  return _mupdf.PdfCrypt_pdf_crypt_revision(self)
26412: (4)              def pdf_crypt_stream_method(self):
26413: (8)                  r""" Class-aware wrapper for `::pdf_crypt_stream_method()`."""
26414: (8)                  return _mupdf.PdfCrypt_pdf_crypt_stream_method(self)
26415: (4)              def pdf_crypt_string_method(self):
26416: (8)                  r""" Class-aware wrapper for `::pdf_crypt_string_method()`."""
26417: (8)                  return _mupdf.PdfCrypt_pdf_crypt_string_method(self)
26418: (4)              def pdf_crypt_user_encryption(self):
26419: (8)                  r""" Class-aware wrapper for `::pdf_crypt_user_encryption()`."""
26420: (8)                  return _mupdf.PdfCrypt_pdf_crypt_user_encryption(self)
26421: (4)              def pdf_crypt_user_password(self):
26422: (8)                  r""" Class-aware wrapper for `::pdf_crypt_user_password()`."""
26423: (8)                  return _mupdf.PdfCrypt_pdf_crypt_user_password(self)
26424: (4)              def pdf_crypt_version(self):
26425: (8)                  r""" Class-aware wrapper for `::pdf_crypt_version()`."""
26426: (8)                  return _mupdf.PdfCrypt_pdf_crypt_version(self)
26427: (4)              def pdf_encrypt_data(self, num, gen, fmt_str_out, arg, s, n):
26428: (8)                  r""" Class-aware wrapper for `::pdf_encrypt_data()`."""
26429: (8)                  return _mupdf.PdfCrypt_pdf_encrypt_data(self, num, gen, fmt_str_out,
arg, s, n)
26430: (4)              def pdf_encrypted_len(self, num, gen, len):
26431: (8)                  r""" Class-aware wrapper for `::pdf_encrypted_len()`."""
26432: (8)                  return _mupdf.PdfCrypt_pdf_encrypted_len(self, num, gen, len)
26433: (4)              def __init__(self, *args):
26434: (8)                  r"""
26435: (8)                  *Overload 1:*
26436: (8)                  == Constructors.  Constructor using `pdf_new_crypt()`.
26437: (8)                  |
26438: (8)                  *Overload 2:*
26439: (8)                  Constructor using `pdf_new_encrypt()`.
26440: (8)                  |
26441: (8)                  *Overload 3:*
26442: (8)                  Default constructor, sets `m_internal` to null.
26443: (8)                  |
26444: (8)                  *Overload 4:*
26445: (8)                  Constructor using raw copy of pre-existing `::pdf_crypt`.
26446: (8)                  """
26447: (8)                  _mupdf.PdfCrypt_swiginit(self, _mupdf.new_PdfCrypt(*args))
26448: (4)              __swig_destroy__ = _mupdf.delete_PdfCrypt
26449: (4)              def m_internal_value(self):
26450: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26451: (8)                  return _mupdf.PdfCrypt_m_internal_value(self)
```

```
26452: (4)              m_internal = property(_mupdf.PdfCrypt_m_internal_get,
_mupdf.PdfCrypt_m_internal_set, doc=r""" Pointer to wrapped data.""")
26453: (4)              s_num_instances = property(_mupdf.PdfCrypt_s_num_instances_get,
_mupdf.PdfCrypt_s_num_instances_set)
26454: (0)          _mupdf.PdfCrypt_swigregister(PdfCrypt)
26455: (0)          class PdfCsi(object):
26456: (4)              r""" Wrapper class for struct `pdf_csi`. Not copyable or assignable."""
26457: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26458: (4)              __repr__ = _swig_repr
26459: (4)              def __init__(self, *args):
26460: (8)                  r"""
26461: (8)                  *Overload 1:*
26462: (8)                  Default constructor, sets `m_internal` to null.
26463: (8)                  |
26464: (8)                  *Overload 2:*
26465: (8)                  Constructor using raw copy of pre-existing `::pdf_csi`.
26466: (8)                  """
26467: (8)                  _mupdf.PdfCsi_swiginit(self, _mupdf.new_PdfCsi(*args))
26468: (4)              __swig_destroy__ = _mupdf.delete_PdfCsi
26469: (4)              def m_internal_value(self):
26470: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26471: (8)                  return _mupdf.PdfCsi_m_internal_value(self)
26472: (4)              m_internal = property(_mupdf.PdfCsi_m_internal_get,
_mupdf.PdfCsi_m_internal_set, doc=r""" Pointer to wrapped data.""")
26473: (4)              s_num_instances = property(_mupdf.PdfCsi_s_num_instances_get,
_mupdf.PdfCsi_s_num_instances_set)
26474: (0)          _mupdf.PdfCsi_swigregister(PdfCsi)
26475: (0)          class PdfCycleList(object):
26476: (4)              r""" Wrapper class for struct `pdf_cycle_list`. Not copyable or
assignable."""
26477: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26478: (4)              __repr__ = _swig_repr
26479: (4)              def pdf_cycle(self, prev, obj):
26480: (8)                  r""" Class-aware wrapper for `::pdf_cycle()`."""
26481: (8)                  return _mupdf.PdfCycleList_pdf_cycle(self, prev, obj)
26482: (4)              def __init__(self, *args):
26483: (8)                  r"""
26484: (8)                  *Overload 1:*
26485: (8)                  Default constructor, sets `m_internal` to null.
26486: (8)                  |
26487: (8)                  *Overload 2:*
26488: (8)                  Constructor using raw copy of pre-existing `::pdf_cycle_list`.
26489: (8)                  """
26490: (8)                  _mupdf.PdfCycleList_swiginit(self, _mupdf.new_PdfCycleList(*args))
26491: (4)              __swig_destroy__ = _mupdf.delete_PdfCycleList
26492: (4)              def m_internal_value(self):
26493: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26494: (8)                  return _mupdf.PdfCycleList_m_internal_value(self)
26495: (4)              m_internal = property(_mupdf.PdfCycleList_m_internal_get,
_mupdf.PdfCycleList_m_internal_set, doc=r""" Pointer to wrapped data.""")
26496: (4)              s_num_instances = property(_mupdf.PdfCycleList_s_num_instances_get,
_mupdf.PdfCycleList_s_num_instances_set)
26497: (0)          _mupdf.PdfCycleList_swigregister(PdfCycleList)
26498: (0)          class PdfDocEvent(object):
26499: (4)              r""" Wrapper class for struct `pdf_doc_event`. Not copyable or
assignable."""
26500: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26501: (4)              __repr__ = _swig_repr
26502: (4)              def pdf_access_exec_menu_item_event(self):
26503: (8)                  r""" Class-aware wrapper for `::pdf_access_exec_menu_item_event()`."""
26504: (8)                  return _mupdf.PdfDocEvent_pdf_access_exec_menu_item_event(self)
26505: (4)              def __init__(self, *args):
26506: (8)                  r"""
26507: (8)                  *Overload 1:*
```

```
26508: (8)                  Default constructor, sets `m_internal` to null.
26509: (8)                  |
26510: (8)                  *Overload 2:*
26511: (8)                  Constructor using raw copy of pre-existing `::pdf_doc_event`.
26512: (8)                  """
26513: (8)                  _mupdf.PdfDocEvent_swiginit(self, _mupdf.new_PdfDocEvent(*args))
26514: (4)              __swig_destroy__ = _mupdf.delete_PdfDocEvent
26515: (4)              def m_internal_value(self):
26516: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
26517: (8)                  return _mupdf.PdfDocEvent_m_internal_value(self)
26518: (4)              m_internal = property(_mupdf.PdfDocEvent_m_internal_get,
_mupdf.PdfDocEvent_m_internal_set, doc=r""" Pointer to wrapped data.""")
26519: (4)              s_num_instances = property(_mupdf.PdfDocEvent_s_num_instances_get,
_mupdf.PdfDocEvent_s_num_instances_set)
26520: (0)          _mupdf.PdfDocEvent_swigregister(PdfDocEvent)
26521: (0)          class PdfDocument(object):
26522: (4)              r""" Wrapper class for struct `pdf_document`."""
26523: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
26524: (4)              __repr__ = _swig_repr
26525: (4)              def pdf_abandon_operation(self):
26526: (8)                  r""" Class-aware wrapper for `::pdf_abandon_operation()`."""
26527: (8)                  return _mupdf.PdfDocument_pdf_abandon_operation(self)
26528: (4)              def pdf_add_cid_font(self, font):
26529: (8)                  r""" Class-aware wrapper for `::pdf_add_cid_font()`."""
26530: (8)                  return _mupdf.PdfDocument_pdf_add_cid_font(self, font)
26531: (4)              def pdf_add_cjk_font(self, font, script, wmode, serif):
26532: (8)                  r""" Class-aware wrapper for `::pdf_add_cjk_font()`."""
26533: (8)                  return _mupdf.PdfDocument_pdf_add_cjk_font(self, font, script, wmode,
serif)
26534: (4)              def pdf_add_embedded_file(self, filename, mimetype, contents, created,
modifed, add_checksum):
26535: (8)                  r""" Class-aware wrapper for `::pdf_add_embedded_file()`."""
26536: (8)                  return _mupdf.PdfDocument_pdf_add_embedded_file(self, filename,
mimetype, contents, created, modifed, add_checksum)
26537: (4)              def pdf_add_image(self, image):
26538: (8)                  r""" Class-aware wrapper for `::pdf_add_image()`."""
26539: (8)                  return _mupdf.PdfDocument_pdf_add_image(self, image)
26540: (4)              def pdf_add_journal_fragment(self, parent, copy, copy_stream, newobj):
26541: (8)                  r""" Class-aware wrapper for `::pdf_add_journal_fragment()`."""
26542: (8)                  return _mupdf.PdfDocument_pdf_add_journal_fragment(self, parent, copy,
copy_stream, newobj)
26543: (4)              def pdf_add_new_array(self, initial):
26544: (8)                  r""" Class-aware wrapper for `::pdf_add_new_array()`."""
26545: (8)                  return _mupdf.PdfDocument_pdf_add_new_array(self, initial)
26546: (4)              def pdf_add_new_dict(self, initial):
26547: (8)                  r""" Class-aware wrapper for `::pdf_add_new_dict()`."""
26548: (8)                  return _mupdf.PdfDocument_pdf_add_new_dict(self, initial)
26549: (4)              def pdf_add_object(self, obj):
26550: (8)                  r""" Class-aware wrapper for `::pdf_add_object()`."""
26551: (8)                  return _mupdf.PdfDocument_pdf_add_object(self, obj)
26552: (4)              def pdf_add_page(self, mediabox, rotate, resources, contents):
26553: (8)                  r""" Class-aware wrapper for `::pdf_add_page()`."""
26554: (8)                  return _mupdf.PdfDocument_pdf_add_page(self, mediabox, rotate,
resources, contents)
26555: (4)              def pdf_add_simple_font(self, font, encoding):
26556: (8)                  r""" Class-aware wrapper for `::pdf_add_simple_font()`."""
26557: (8)                  return _mupdf.PdfDocument_pdf_add_simple_font(self, font, encoding)
26558: (4)              def pdf_add_stream(self, buf, obj, compressed):
26559: (8)                  r""" Class-aware wrapper for `::pdf_add_stream()`."""
26560: (8)                  return _mupdf.PdfDocument_pdf_add_stream(self, buf, obj, compressed)
26561: (4)              def pdf_add_substitute_font(self, font):
26562: (8)                  r""" Class-aware wrapper for `::pdf_add_substitute_font()`."""
26563: (8)                  return _mupdf.PdfDocument_pdf_add_substitute_font(self, font)
26564: (4)              def pdf_annot_field_event_keystroke(self, annot, evt):
26565: (8)                  r""" Class-aware wrapper for `::pdf_annot_field_event_keystroke()`."""
26566: (8)                  return _mupdf.PdfDocument_pdf_annot_field_event_keystroke(self, annot,
evt)
```

```
26567: (4)              def pdf_authenticate_password(self, pw):
26568: (8)                  r""" Class-aware wrapper for `::pdf_authenticate_password()`."""
26569: (8)                  return _mupdf.PdfDocument_pdf_authenticate_password(self, pw)
26570: (4)              def pdf_bake_document(self, bake_annots, bake_widgets):
26571: (8)                  r""" Class-aware wrapper for `::pdf_bake_document()`."""
26572: (8)                  return _mupdf.PdfDocument_pdf_bake_document(self, bake_annots,
bake_widgets)
26573: (4)              def pdf_begin_implicit_operation(self):
26574: (8)                  r""" Class-aware wrapper for `::pdf_begin_implicit_operation()`."""
26575: (8)                  return _mupdf.PdfDocument_pdf_begin_implicit_operation(self)
26576: (4)              def pdf_begin_operation(self, operation):
26577: (8)                  r""" Class-aware wrapper for `::pdf_begin_operation()`."""
26578: (8)                  return _mupdf.PdfDocument_pdf_begin_operation(self, operation)
26579: (4)              def pdf_calculate_form(self):
26580: (8)                  r""" Class-aware wrapper for `::pdf_calculate_form()`."""
26581: (8)                  return _mupdf.PdfDocument_pdf_calculate_form(self)
26582: (4)              def pdf_can_be_saved_incrementally(self):
26583: (8)                  r""" Class-aware wrapper for `::pdf_can_be_saved_incrementally()`."""
26584: (8)                  return _mupdf.PdfDocument_pdf_can_be_saved_incrementally(self)
26585: (4)              def pdf_can_redo(self):
26586: (8)                  r""" Class-aware wrapper for `::pdf_can_redo()`."""
26587: (8)                  return _mupdf.PdfDocument_pdf_can_redo(self)
26588: (4)              def pdf_can_undo(self):
26589: (8)                  r""" Class-aware wrapper for `::pdf_can_undo()`."""
26590: (8)                  return _mupdf.PdfDocument_pdf_can_undo(self)
26591: (4)              def pdf_clear_xref(self):
26592: (8)                  r""" Class-aware wrapper for `::pdf_clear_xref()`."""
26593: (8)                  return _mupdf.PdfDocument_pdf_clear_xref(self)
26594: (4)              def pdf_clear_xref_to_mark(self):
26595: (8)                  r""" Class-aware wrapper for `::pdf_clear_xref_to_mark()`."""
26596: (8)                  return _mupdf.PdfDocument_pdf_clear_xref_to_mark(self)
26597: (4)              def pdf_count_layer_config_ui(self):
26598: (8)                  r""" Class-aware wrapper for `::pdf_count_layer_config_ui()`."""
26599: (8)                  return _mupdf.PdfDocument_pdf_count_layer_config_ui(self)
26600: (4)              def pdf_count_layer_configs(self):
26601: (8)                  r""" Class-aware wrapper for `::pdf_count_layer_configs()`."""
26602: (8)                  return _mupdf.PdfDocument_pdf_count_layer_configs(self)
26603: (4)              def pdf_count_layers(self):
26604: (8)                  r""" Class-aware wrapper for `::pdf_count_layers()`."""
26605: (8)                  return _mupdf.PdfDocument_pdf_count_layers(self)
26606: (4)              def pdf_count_objects(self):
26607: (8)                  r""" Class-aware wrapper for `::pdf_count_objects()`."""
26608: (8)                  return _mupdf.PdfDocument_pdf_count_objects(self)
26609: (4)              def pdf_count_pages(self):
26610: (8)                  r""" Class-aware wrapper for `::pdf_count_pages()`."""
26611: (8)                  return _mupdf.PdfDocument_pdf_count_pages(self)
26612: (4)              def pdf_count_q_balance(self, res, stm, prepend, append):
26613: (8)                  r"""
26614: (8)                  Class-aware wrapper for `::pdf_count_q_balance()`.
26615: (8)                  This method has out-params. Python/C# wrappers look like:
26616: (12)                     `pdf_count_q_balance(::pdf_obj *res, ::pdf_obj *stm)` => `(int
prepend, int append)`
26617: (8)                  """
26618: (8)                  return _mupdf.PdfDocument_pdf_count_q_balance(self, res, stm, prepend,
append)
26619: (4)              def pdf_count_signatures(self):
26620: (8)                  r""" Class-aware wrapper for `::pdf_count_signatures()`."""
26621: (8)                  return _mupdf.PdfDocument_pdf_count_signatures(self)
26622: (4)              def pdf_count_unsaved_versions(self):
26623: (8)                  r""" Class-aware wrapper for `::pdf_count_unsaved_versions()`."""
26624: (8)                  return _mupdf.PdfDocument_pdf_count_unsaved_versions(self)
26625: (4)              def pdf_count_versions(self):
26626: (8)                  r""" Class-aware wrapper for `::pdf_count_versions()`."""
26627: (8)                  return _mupdf.PdfDocument_pdf_count_versions(self)
26628: (4)              def pdf_create_field_name(self, prefix, buf, len):
26629: (8)                  r""" Class-aware wrapper for `::pdf_create_field_name()`."""
26630: (8)                  return _mupdf.PdfDocument_pdf_create_field_name(self, prefix, buf,
len)
26631: (4)              def pdf_create_object(self):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_create_object()`."""
(8)                 return _mupdf.PdfDocument_pdf_create_object(self)
(4)             def pdf_debug_doc_changes(self):
(8)                 r""" Class-aware wrapper for `::pdf_debug_doc_changes()`."""
(8)                 return _mupdf.PdfDocument_pdf_debug_doc_changes(self)
(4)             def pdf_delete_object(self, num):
(8)                 r""" Class-aware wrapper for `::pdf_delete_object()`."""
(8)                 return _mupdf.PdfDocument_pdf_delete_object(self, num)
(4)             def pdf_delete_page(self, number):
(8)                 r""" Class-aware wrapper for `::pdf_delete_page()`."""
(8)                 return _mupdf.PdfDocument_pdf_delete_page(self, number)
(4)             def pdf_delete_page_labels(self, index):
(8)                 r""" Class-aware wrapper for `::pdf_delete_page_labels()`."""
(8)                 return _mupdf.PdfDocument_pdf_delete_page_labels(self, index)
(4)             def pdf_delete_page_range(self, start, end):
(8)                 r""" Class-aware wrapper for `::pdf_delete_page_range()`."""
(8)                 return _mupdf.PdfDocument_pdf_delete_page_range(self, start, end)
(4)             def pdf_deselect_layer_config_ui(self, ui):
(8)                 r""" Class-aware wrapper for `::pdf_deselect_layer_config_ui()`."""
(8)                 return _mupdf.PdfDocument_pdf_deselect_layer_config_ui(self, ui)
(4)             def pdf_deserialise_journal(self, stm):
(8)                 r""" Class-aware wrapper for `::pdf_deserialise_journal()`."""
(8)                 return _mupdf.PdfDocument_pdf_deserialise_journal(self, stm)
(4)             def pdf_disable_js(self):
(8)                 r""" Class-aware wrapper for `::pdf_disable_js()`."""
(8)                 return _mupdf.PdfDocument_pdf_disable_js(self)
(4)             def pdf_doc_was_linearized(self):
(8)                 r""" Class-aware wrapper for `::pdf_doc_was_linearized()`."""
(8)                 return _mupdf.PdfDocument_pdf_doc_was_linearized(self)
(4)             def pdf_document_event_did_print(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_event_did_print()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_event_did_print(self)
(4)             def pdf_document_event_did_save(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_event_did_save()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_event_did_save(self)
(4)             def pdf_document_event_will_close(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_event_will_close()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_event_will_close(self)
(4)             def pdf_document_event_will_print(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_event_will_print()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_event_will_print(self)
(4)             def pdf_document_event_will_save(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_event_will_save()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_event_will_save(self)
(4)             def pdf_document_output_intent(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_output_intent()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_output_intent(self)
(4)             def pdf_document_permissions(self):
(8)                 r""" Class-aware wrapper for `::pdf_document_permissions()`."""
(8)                 return _mupdf.PdfDocument_pdf_document_permissions(self)
(4)             def pdf_empty_store(self):
(8)                 r""" Class-aware wrapper for `::pdf_empty_store()`."""
(8)                 return _mupdf.PdfDocument_pdf_empty_store(self)
(4)             def pdf_enable_journal(self):
(8)                 r""" Class-aware wrapper for `::pdf_enable_journal()`."""
(8)                 return _mupdf.PdfDocument_pdf_enable_journal(self)
(4)             def pdf_enable_js(self):
(8)                 r""" Class-aware wrapper for `::pdf_enable_js()`."""
(8)                 return _mupdf.PdfDocument_pdf_enable_js(self)
(4)             def pdf_enable_layer(self, layer, enabled):
(8)                 r""" Class-aware wrapper for `::pdf_enable_layer()`."""
(8)                 return _mupdf.PdfDocument_pdf_enable_layer(self, layer, enabled)
(4)             def pdf_end_operation(self):
(8)                 r""" Class-aware wrapper for `::pdf_end_operation()`."""
(8)                 return _mupdf.PdfDocument_pdf_end_operation(self)
(4)             def pdf_ensure_solid_xref(self, num):
(8)                 r""" Class-aware wrapper for `::pdf_ensure_solid_xref()`."""
(8)                 return _mupdf.PdfDocument_pdf_ensure_solid_xref(self, num)
(4)             def pdf_event_issue_alert(self, evt):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_event_issue_alert()`."""
(8)                 return _mupdf.PdfDocument_pdf_event_issue_alert(self, evt)
(4)             def pdf_event_issue_exec_menu_item(self, item):
(8)                 r""" Class-aware wrapper for `::pdf_event_issue_exec_menu_item()`."""
(8)                 return _mupdf.PdfDocument_pdf_event_issue_exec_menu_item(self, item)
(4)             def pdf_event_issue_launch_url(self, url, new_frame):
(8)                 r""" Class-aware wrapper for `::pdf_event_issue_launch_url()`."""
(8)                 return _mupdf.PdfDocument_pdf_event_issue_launch_url(self, url,
new_frame)
(4)             def pdf_event_issue_mail_doc(self, evt):
(8)                 r""" Class-aware wrapper for `::pdf_event_issue_mail_doc()`."""
(8)                 return _mupdf.PdfDocument_pdf_event_issue_mail_doc(self, evt)
(4)             def pdf_event_issue_print(self):
(8)                 r""" Class-aware wrapper for `::pdf_event_issue_print()`."""
(8)                 return _mupdf.PdfDocument_pdf_event_issue_print(self)
(4)             def pdf_field_event_calculate(self, field):
(8)                 r""" Class-aware wrapper for `::pdf_field_event_calculate()`."""
(8)                 return _mupdf.PdfDocument_pdf_field_event_calculate(self, field)
(4)             def pdf_field_event_format(self, field):
(8)                 r""" Class-aware wrapper for `::pdf_field_event_format()`."""
(8)                 return _mupdf.PdfDocument_pdf_field_event_format(self, field)
(4)             def pdf_field_event_keystroke(self, field, evt):
(8)                 r""" Class-aware wrapper for `::pdf_field_event_keystroke()`."""
(8)                 return _mupdf.PdfDocument_pdf_field_event_keystroke(self, field, evt)
(4)             def pdf_field_event_validate(self, field, value, newvalue):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_field_event_validate()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_field_event_validate(::pdf_obj *field, const char *value)` =>
`(int, char *newvalue)`
(8)                 """
(8)                 return _mupdf.PdfDocument_pdf_field_event_validate(self, field, value,
newvalue)
(4)             def pdf_field_reset(self, field):
(8)                 r""" Class-aware wrapper for `::pdf_field_reset()`."""
(8)                 return _mupdf.PdfDocument_pdf_field_reset(self, field)
(4)             def pdf_filter_annot_contents(self, annot, options):
(8)                 r""" Class-aware wrapper for `::pdf_filter_annot_contents()`."""
(8)                 return _mupdf.PdfDocument_pdf_filter_annot_contents(self, annot,
options)
(4)             def pdf_filter_page_contents(self, page, options):
(8)                 r""" Class-aware wrapper for `::pdf_filter_page_contents()`."""
(8)                 return _mupdf.PdfDocument_pdf_filter_page_contents(self, page,
options)
(4)             def pdf_find_font_resource(self, type, encoding, item, key):
(8)                 r""" Class-aware wrapper for `::pdf_find_font_resource()`."""
(8)                 return _mupdf.PdfDocument_pdf_find_font_resource(self, type, encoding,
item, key)
(4)             def pdf_find_version_for_obj(self, obj):
(8)                 r""" Class-aware wrapper for `::pdf_find_version_for_obj()`."""
(8)                 return _mupdf.PdfDocument_pdf_find_version_for_obj(self, obj)
(4)             def pdf_forget_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_forget_xref()`."""
(8)                 return _mupdf.PdfDocument_pdf_forget_xref(self)
(4)             def pdf_get_doc_event_callback_data(self):
(8)                 r""" Class-aware wrapper for `::pdf_get_doc_event_callback_data()`."""
(8)                 return _mupdf.PdfDocument_pdf_get_doc_event_callback_data(self)
(4)             def pdf_graft_object(self, obj):
(8)                 r""" Class-aware wrapper for `::pdf_graft_object()`."""
(8)                 return _mupdf.PdfDocument_pdf_graft_object(self, obj)
(4)             def pdf_graft_page(self, page_to, src, page_from):
(8)                 r""" Class-aware wrapper for `::pdf_graft_page()`."""
(8)                 return _mupdf.PdfDocument_pdf_graft_page(self, page_to, src,
page_from)
(4)             def pdf_has_permission(self, p):
(8)                 r""" Class-aware wrapper for `::pdf_has_permission()`."""
(8)                 return _mupdf.PdfDocument_pdf_has_permission(self, p)
(4)             def pdf_has_unsaved_changes(self):
(8)                 r""" Class-aware wrapper for `::pdf_has_unsaved_changes()`."""
```

```
26763: (8)                  return _mupdf.PdfDocument_pdf_has_unsaved_changes(self)
26764: (4)              def pdf_has_unsaved_sigs(self):
26765: (8)                  r""" Class-aware wrapper for `::pdf_has_unsaved_sigs()`."""
26766: (8)                  return _mupdf.PdfDocument_pdf_has_unsaved_sigs(self)
26767: (4)              def pdf_insert_font_resource(self, key, obj):
26768: (8)                  r""" Class-aware wrapper for `::pdf_insert_font_resource()`."""
26769: (8)                  return _mupdf.PdfDocument_pdf_insert_font_resource(self, key, obj)
26770: (4)              def pdf_insert_page(self, at, page):
26771: (8)                  r""" Class-aware wrapper for `::pdf_insert_page()`."""
26772: (8)                  return _mupdf.PdfDocument_pdf_insert_page(self, at, page)
26773: (4)              def pdf_invalidate_xfa(self):
26774: (8)                  r""" Class-aware wrapper for `::pdf_invalidate_xfa()`."""
26775: (8)                  return _mupdf.PdfDocument_pdf_invalidate_xfa(self)
26776: (4)              def pdf_is_local_object(self, obj):
26777: (8)                  r""" Class-aware wrapper for `::pdf_is_local_object()`."""
26778: (8)                  return _mupdf.PdfDocument_pdf_is_local_object(self, obj)
26779: (4)              def pdf_is_ocg_hidden(self, rdb, usage, ocg):
26780: (8)                  r""" Class-aware wrapper for `::pdf_is_ocg_hidden()`."""
26781: (8)                  return _mupdf.PdfDocument_pdf_is_ocg_hidden(self, rdb, usage, ocg)
26782: (4)              def pdf_js_set_console(self, console, user):
26783: (8)                  r""" Class-aware wrapper for `::pdf_js_set_console()`."""
26784: (8)                  return _mupdf.PdfDocument_pdf_js_set_console(self, console, user)
26785: (4)              def pdf_js_supported(self):
26786: (8)                  r""" Class-aware wrapper for `::pdf_js_supported()`."""
26787: (8)                  return _mupdf.PdfDocument_pdf_js_supported(self)
26788: (4)              def pdf_layer_config_info(self, config_num, info):
26789: (8)                  r""" Class-aware wrapper for `::pdf_layer_config_info()`."""
26790: (8)                  return _mupdf.PdfDocument_pdf_layer_config_info(self, config_num,
info)
26791: (4)              def pdf_layer_config_ui_info(self, ui, info):
26792: (8)                  r""" Class-aware wrapper for `::pdf_layer_config_ui_info()`."""
26793: (8)                  return _mupdf.PdfDocument_pdf_layer_config_ui_info(self, ui, info)
26794: (4)              def pdf_layer_is_enabled(self, layer):
26795: (8)                  r""" Class-aware wrapper for `::pdf_layer_is_enabled()`."""
26796: (8)                  return _mupdf.PdfDocument_pdf_layer_is_enabled(self, layer)
26797: (4)              def pdf_layer_name(self, layer):
26798: (8)                  r""" Class-aware wrapper for `::pdf_layer_name()`."""
26799: (8)                  return _mupdf.PdfDocument_pdf_layer_name(self, layer)
26800: (4)              def pdf_load_compressed_inline_image(self, dict, length, cstm, indexed,
image):
26801: (8)                  r""" Class-aware wrapper for
`::pdf_load_compressed_inline_image()`."""
26802: (8)                  return _mupdf.PdfDocument_pdf_load_compressed_inline_image(self, dict,
length, cstm, indexed, image)
26803: (4)              def pdf_load_compressed_stream(self, num, worst_case):
26804: (8)                  r""" Class-aware wrapper for `::pdf_load_compressed_stream()`."""
26805: (8)                  return _mupdf.PdfDocument_pdf_load_compressed_stream(self, num,
worst_case)
26806: (4)              def pdf_load_default_colorspaces(self, page):
26807: (8)                  r""" Class-aware wrapper for `::pdf_load_default_colorspaces()`."""
26808: (8)                  return _mupdf.PdfDocument_pdf_load_default_colorspaces(self, page)
26809: (4)              def pdf_load_embedded_cmap(self, ref):
26810: (8)                  r""" Class-aware wrapper for `::pdf_load_embedded_cmap()`."""
26811: (8)                  return _mupdf.PdfDocument_pdf_load_embedded_cmap(self, ref)
26812: (4)              def pdf_load_image(self, obj):
26813: (8)                  r""" Class-aware wrapper for `::pdf_load_image()`."""
26814: (8)                  return _mupdf.PdfDocument_pdf_load_image(self, obj)
26815: (4)              def pdf_load_inline_image(self, rdb, dict, file):
26816: (8)                  r""" Class-aware wrapper for `::pdf_load_inline_image()`."""
26817: (8)                  return _mupdf.PdfDocument_pdf_load_inline_image(self, rdb, dict, file)
26818: (4)              def pdf_load_journal(self, filename):
26819: (8)                  r""" Class-aware wrapper for `::pdf_load_journal()`."""
26820: (8)                  return _mupdf.PdfDocument_pdf_load_journal(self, filename)
26821: (4)              def pdf_load_link_annots(self, arg_1, annots, pagenum, page_ctm):
26822: (8)                  r""" Class-aware wrapper for `::pdf_load_link_annots()`."""
26823: (8)                  return _mupdf.PdfDocument_pdf_load_link_annots(self, arg_1, annots,
pagenum, page_ctm)
26824: (4)              def pdf_load_name_tree(self, which):
26825: (8)                  r""" Class-aware wrapper for `::pdf_load_name_tree()`."""
```

```
26826: (8)                  return _mupdf.PdfDocument_pdf_load_name_tree(self, which)
26827: (4)              def pdf_load_object(self, num):
26828: (8)                  r"""
26829: (8)                  Class-aware wrapper for `::pdf_load_object()`.
26830: (12)                     Load a given object.
26831: (12)                     This can cause xref reorganisations (solidifications etc) due to
26832: (12)                     repairs, so all held pdf_xref_entries should be considered
26833: (12)                     invalid after this call (other than the returned one).
26834: (8)                  """
26835: (8)                  return _mupdf.PdfDocument_pdf_load_object(self, num)
26836: (4)              def pdf_load_outline(self):
26837: (8)                  r""" Class-aware wrapper for `::pdf_load_outline()`."""
26838: (8)                  return _mupdf.PdfDocument_pdf_load_outline(self)
26839: (4)              def pdf_load_page(self, number):
26840: (8)                  r""" Class-aware wrapper for `::pdf_load_page()`."""
26841: (8)                  return _mupdf.PdfDocument_pdf_load_page(self, number)
26842: (4)              def pdf_load_page_tree(self):
26843: (8)                  r""" Class-aware wrapper for `::pdf_load_page_tree()`."""
26844: (8)                  return _mupdf.PdfDocument_pdf_load_page_tree(self)
26845: (4)              def pdf_load_pattern(self, obj):
26846: (8)                  r""" Class-aware wrapper for `::pdf_load_pattern()`."""
26847: (8)                  return _mupdf.PdfDocument_pdf_load_pattern(self, obj)
26848: (4)              def pdf_load_raw_stream_number(self, num):
26849: (8)                  r""" Class-aware wrapper for `::pdf_load_raw_stream_number()`."""
26850: (8)                  return _mupdf.PdfDocument_pdf_load_raw_stream_number(self, num)
26851: (4)              def pdf_load_shading(self, obj):
26852: (8)                  r""" Class-aware wrapper for `::pdf_load_shading()`."""
26853: (8)                  return _mupdf.PdfDocument_pdf_load_shading(self, obj)
26854: (4)              def pdf_load_stream_number(self, num):
26855: (8)                  r""" Class-aware wrapper for `::pdf_load_stream_number()`."""
26856: (8)                  return _mupdf.PdfDocument_pdf_load_stream_number(self, num)
26857: (4)              def pdf_load_to_unicode(self, font, strings, collection, cmapstm):
26858: (8)                  r"""
26859: (8)                  Class-aware wrapper for `::pdf_load_to_unicode()`.
26860: (8)                  This method has out-params. Python/C# wrappers look like:
26861: (12)                     `pdf_load_to_unicode(::pdf_font_desc *font, char *collection,
::pdf_obj *cmapstm)` => const char *strings
26862: (8)                  """
26863: (8)                  return _mupdf.PdfDocument_pdf_load_to_unicode(self, font, strings,
collection, cmapstm)
26864: (4)              def pdf_load_type3_glyphs(self, fontdesc):
26865: (8)                  r""" Class-aware wrapper for `::pdf_load_type3_glyphs()`."""
26866: (8)                  return _mupdf.PdfDocument_pdf_load_type3_glyphs(self, fontdesc)
26867: (4)              def pdf_load_unencrypted_object(self, num):
26868: (8)                  r""" Class-aware wrapper for `::pdf_load_unencrypted_object()`."""
26869: (8)                  return _mupdf.PdfDocument_pdf_load_unencrypted_object(self, num)
26870: (4)              def pdf_lookup_dest(self, needle):
26871: (8)                  r""" Class-aware wrapper for `::pdf_lookup_dest()`."""
26872: (8)                  return _mupdf.PdfDocument_pdf_lookup_dest(self, needle)
26873: (4)              def pdf_lookup_metadata(self, key, ptr, size):
26874: (8)                  r""" Class-aware wrapper for `::pdf_lookup_metadata()`."""
26875: (8)                  return _mupdf.PdfDocument_pdf_lookup_metadata(self, key, ptr, size)
26876: (4)              def pdf_lookup_metadata2(self, key):
26877: (8)                  r"""
26878: (8)                  Class-aware wrapper for `::pdf_lookup_metadata2()`.
26879: (8)                  C++ alternative to `pdf_lookup_metadata()` that returns a
`std::string`
26880: (8)                  or calls `fz_throw()` if not found.
26881: (8)                  """
26882: (8)                  return _mupdf.PdfDocument_pdf_lookup_metadata2(self, key)
26883: (4)              def pdf_lookup_name(self, which, needle):
26884: (8)                  r""" Class-aware wrapper for `::pdf_lookup_name()`."""
26885: (8)                  return _mupdf.PdfDocument_pdf_lookup_name(self, which, needle)
26886: (4)              def pdf_lookup_page_loc(self, needle, parentp, indexp):
26887: (8)                  r"""
26888: (8)                  Class-aware wrapper for `::pdf_lookup_page_loc()`.
26889: (8)                  This method has out-params. Python/C# wrappers look like:
26890: (12)                     `pdf_lookup_page_loc(int needle, ::pdf_obj **parentp)` =>
`(pdf_obj *, int indexp)`
```

```
(8)                 """
(8)                 return _mupdf.PdfDocument_pdf_lookup_page_loc(self, needle, parentp, indexp)
(4)             def pdf_lookup_page_number(self, pageobj):
(8)                 r""" Class-aware wrapper for `::pdf_lookup_page_number()`."""
(8)                 return _mupdf.PdfDocument_pdf_lookup_page_number(self, pageobj)
(4)             def pdf_lookup_page_obj(self, needle):
(8)                 r""" Class-aware wrapper for `::pdf_lookup_page_obj()`."""
(8)                 return _mupdf.PdfDocument_pdf_lookup_page_obj(self, needle)
(4)             def pdf_mark_xref(self):
(8)                 r""" Class-aware wrapper for `::pdf_mark_xref()`."""
(8)                 return _mupdf.PdfDocument_pdf_mark_xref(self)
(4)             def pdf_metadata(self):
(8)                 r""" Class-aware wrapper for `::pdf_metadata()`."""
(8)                 return _mupdf.PdfDocument_pdf_metadata(self)
(4)             def pdf_minimize_document(self):
(8)                 r""" Class-aware wrapper for `::pdf_minimize_document()`."""
(8)                 return _mupdf.PdfDocument_pdf_minimize_document(self)
(4)             def pdf_needs_password(self):
(8)                 r""" Class-aware wrapper for `::pdf_needs_password()`."""
(8)                 return _mupdf.PdfDocument_pdf_needs_password(self)
(4)             def pdf_new_action_from_link(self, uri):
(8)                 r""" Class-aware wrapper for `::pdf_new_action_from_link()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_action_from_link(self, uri)
(4)             def pdf_new_array(self, initialcap):
(8)                 r""" Class-aware wrapper for `::pdf_new_array()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_array(self, initialcap)
(4)             def pdf_new_color_filter(self, chain, struct_parents, transform, options, copts):
(8)                 r""" Class-aware wrapper for `::pdf_new_color_filter()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_color_filter(self, chain, struct_parents, transform, options, copts)
(4)             def pdf_new_date(self, time):
(8)                 r""" Class-aware wrapper for `::pdf_new_date()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_date(self, time)
(4)             def pdf_new_dest_from_link(self, uri, is_remote):
(8)                 r""" Class-aware wrapper for `::pdf_new_dest_from_link()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_dest_from_link(self, uri, is_remote)
(4)             def pdf_new_dict(self, initialcap):
(8)                 r""" Class-aware wrapper for `::pdf_new_dict()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_dict(self, initialcap)
(4)             def pdf_new_graft_map(self):
(8)                 r""" Class-aware wrapper for `::pdf_new_graft_map()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_graft_map(self)
(4)             def pdf_new_indirect(self, num, gen):
(8)                 r""" Class-aware wrapper for `::pdf_new_indirect()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_indirect(self, num, gen)
(4)             def pdf_new_matrix(self, mtx):
(8)                 r""" Class-aware wrapper for `::pdf_new_matrix()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_matrix(self, mtx)
(4)             def pdf_new_pdf_device(self, topctm, resources, contents):
(8)                 r""" Class-aware wrapper for `::pdf_new_pdf_device()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_pdf_device(self, topctm, resources, contents)
(4)             def pdf_new_rect(self, rect):
(8)                 r""" Class-aware wrapper for `::pdf_new_rect()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_rect(self, rect)
(4)             def pdf_new_run_processor(self, dev, ctm, struct_parent, usage, gstate, default_cs, cookie):
(8)                 r""" Class-aware wrapper for `::pdf_new_run_processor()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_run_processor(self, dev, ctm, struct_parent, usage, gstate, default_cs, cookie)
(4)             def pdf_new_sanitize_filter(self, chain, struct_parents, transform, options, sopts):
(8)                 r""" Class-aware wrapper for `::pdf_new_sanitize_filter()`."""
(8)                 return _mupdf.PdfDocument_pdf_new_sanitize_filter(self, chain, struct_parents, transform, options, sopts)
(4)             def pdf_new_xobject(self, bbox, matrix, res, buffer):
(8)                 r""" Class-aware wrapper for `::pdf_new_xobject()`."""
```

```
26952: (8)                  return _mupdf.PdfDocument_pdf_new_xobject(self, bbox, matrix, res,
buffer)
26953: (4)              def pdf_obj_num_is_stream(self, num):
26954: (8)                  r""" Class-aware wrapper for `::pdf_obj_num_is_stream()`."""
26955: (8)                  return _mupdf.PdfDocument_pdf_obj_num_is_stream(self, num)
26956: (4)              def pdf_open_contents_stream(self, obj):
26957: (8)                  r""" Class-aware wrapper for `::pdf_open_contents_stream()`."""
26958: (8)                  return _mupdf.PdfDocument_pdf_open_contents_stream(self, obj)
26959: (4)              def pdf_open_inline_stream(self, stmobj, length, chain, params):
26960: (8)                  r""" Class-aware wrapper for `::pdf_open_inline_stream()`."""
26961: (8)                  return _mupdf.PdfDocument_pdf_open_inline_stream(self, stmobj, length,
chain, params)
26962: (4)              def pdf_open_raw_stream_number(self, num):
26963: (8)                  r""" Class-aware wrapper for `::pdf_open_raw_stream_number()`."""
26964: (8)                  return _mupdf.PdfDocument_pdf_open_raw_stream_number(self, num)
26965: (4)              def pdf_open_stream_number(self, num):
26966: (8)                  r""" Class-aware wrapper for `::pdf_open_stream_number()`."""
26967: (8)                  return _mupdf.PdfDocument_pdf_open_stream_number(self, num)
26968: (4)              def pdf_open_stream_with_offset(self, num, dict, stm_ofs):
26969: (8)                  r""" Class-aware wrapper for `::pdf_open_stream_with_offset()`."""
26970: (8)                  return _mupdf.PdfDocument_pdf_open_stream_with_offset(self, num, dict,
stm_ofs)
26971: (4)              def pdf_page_label(self, page, buf, size):
26972: (8)                  r""" Class-aware wrapper for `::pdf_page_label()`."""
26973: (8)                  return _mupdf.PdfDocument_pdf_page_label(self, page, buf, size)
26974: (4)              def pdf_page_write(self, mediabox, presources, pcontents):
26975: (8)                  r"""
26976: (8)                  Class-aware wrapper for `::pdf_page_write()`.
26977: (8)                  This method has out-params. Python/C# wrappers look like:
26978: (12)                     `pdf_page_write(::fz_rect mediabox, ::pdf_obj **presources,
::fz_buffer **pcontents)` => `(fz_device *)`
26979: (8)                  """
26980: (8)                  return _mupdf.PdfDocument_pdf_page_write(self, mediabox, presources,
pcontents)
26981: (4)              def pdf_parse_array(self, f, buf):
26982: (8)                  r""" Class-aware wrapper for `::pdf_parse_array()`."""
26983: (8)                  return _mupdf.PdfDocument_pdf_parse_array(self, f, buf)
26984: (4)              def pdf_parse_dict(self, f, buf):
26985: (8)                  r""" Class-aware wrapper for `::pdf_parse_dict()`."""
26986: (8)                  return _mupdf.PdfDocument_pdf_parse_dict(self, f, buf)
26987: (4)              def pdf_parse_ind_obj(self, f, num, gen, stm_ofs, try_repair):
26988: (8)                  r"""
26989: (8)                  Class-aware wrapper for `::pdf_parse_ind_obj()`.
26990: (8)                  This method has out-params. Python/C# wrappers look like:
26991: (12)                     `pdf_parse_ind_obj(::fz_stream *f)` => `(pdf_obj *, int num, int
gen, int64_t stm_ofs, int try_repair)`
26992: (8)                  """
26993: (8)                  return _mupdf.PdfDocument_pdf_parse_ind_obj(self, f, num, gen,
stm_ofs, try_repair)
26994: (4)              def pdf_parse_journal_obj(self, stm, onum, ostm, newobj):
26995: (8)                  r"""
26996: (8)                  Class-aware wrapper for `::pdf_parse_journal_obj()`.
26997: (8)                  This method has out-params. Python/C# wrappers look like:
26998: (12)                     `pdf_parse_journal_obj(::fz_stream *stm, ::fz_buffer **ostm)` =>
`(pdf_obj *, int onum, int newobj)`
26999: (8)                  """
27000: (8)                  return _mupdf.PdfDocument_pdf_parse_journal_obj(self, stm, onum, ostm,
newobj)
27001: (4)              def pdf_parse_stm_obj(self, f, buf):
27002: (8)                  r""" Class-aware wrapper for `::pdf_parse_stm_obj()`."""
27003: (8)                  return _mupdf.PdfDocument_pdf_parse_stm_obj(self, f, buf)
27004: (4)              def pdf_progressive_advance(self, pagenum):
27005: (8)                  r""" Class-aware wrapper for `::pdf_progressive_advance()`."""
27006: (8)                  return _mupdf.PdfDocument_pdf_progressive_advance(self, pagenum)
27007: (4)              def pdf_purge_local_font_resources(self):
27008: (8)                  r""" Class-aware wrapper for `::pdf_purge_local_font_resources()`."""
27009: (8)                  return _mupdf.PdfDocument_pdf_purge_local_font_resources(self)
27010: (4)              def pdf_purge_locals_from_store(self):
27011: (8)                  r""" Class-aware wrapper for `::pdf_purge_locals_from_store()`."""
```

```
27012: (8)                  return _mupdf.PdfDocument_pdf_purge_locals_from_store(self)
27013: (4)              def pdf_read_journal(self, stm):
27014: (8)                  r""" Class-aware wrapper for `::pdf_read_journal()`."""
27015: (8)                  return _mupdf.PdfDocument_pdf_read_journal(self, stm)
27016: (4)              def pdf_rearrange_pages(self, count, pages):
27017: (8)                  r""" Class-aware wrapper for `::pdf_rearrange_pages()`."""
27018: (8)                  return _mupdf.PdfDocument_pdf_rearrange_pages(self, count, pages)
27019: (4)              def pdf_rearrange_pages2(self, pages):
27020: (8)                  r""" Class-aware wrapper for `::pdf_rearrange_pages2()`.   Swig-
friendly wrapper for pdf_rearrange_pages()."""
27021: (8)                  return _mupdf.PdfDocument_pdf_rearrange_pages2(self, pages)
27022: (4)              def pdf_redact_page(self, page, opts):
27023: (8)                  r""" Class-aware wrapper for `::pdf_redact_page()`."""
27024: (8)                  return _mupdf.PdfDocument_pdf_redact_page(self, page, opts)
27025: (4)              def pdf_redo(self):
27026: (8)                  r""" Class-aware wrapper for `::pdf_redo()`."""
27027: (8)                  return _mupdf.PdfDocument_pdf_redo(self)
27028: (4)              def pdf_repair_obj(self, buf, stmofsp, stmlenp, encrypt, id, page, tmpofs,
root):
27029: (8)                  r"""
27030: (8)                  Class-aware wrapper for `::pdf_repair_obj()`.
27031: (8)                  This method has out-params. Python/C# wrappers look like:
27032: (12)                     `pdf_repair_obj(::pdf_lexbuf *buf, ::pdf_obj **encrypt, ::pdf_obj
**id, ::pdf_obj **page, ::pdf_obj **root)` => `(int, int64_t stmofsp, int64_t stmlenp, int64_t
tmpofs)`
27033: (8)                  """
27034: (8)                  return _mupdf.PdfDocument_pdf_repair_obj(self, buf, stmofsp, stmlenp,
encrypt, id, page, tmpofs, root)
27035: (4)              def pdf_repair_obj_stms(self):
27036: (8)                  r""" Class-aware wrapper for `::pdf_repair_obj_stms()`."""
27037: (8)                  return _mupdf.PdfDocument_pdf_repair_obj_stms(self)
27038: (4)              def pdf_repair_trailer(self):
27039: (8)                  r""" Class-aware wrapper for `::pdf_repair_trailer()`."""
27040: (8)                  return _mupdf.PdfDocument_pdf_repair_trailer(self)
27041: (4)              def pdf_repair_xref(self):
27042: (8)                  r""" Class-aware wrapper for `::pdf_repair_xref()`."""
27043: (8)                  return _mupdf.PdfDocument_pdf_repair_xref(self)
27044: (4)              def pdf_replace_xref(self, entries, n):
27045: (8)                  r""" Class-aware wrapper for `::pdf_replace_xref()`."""
27046: (8)                  return _mupdf.PdfDocument_pdf_replace_xref(self, entries, n)
27047: (4)              def pdf_reset_form(self, fields, exclude):
27048: (8)                  r""" Class-aware wrapper for `::pdf_reset_form()`."""
27049: (8)                  return _mupdf.PdfDocument_pdf_reset_form(self, fields, exclude)
27050: (4)              def pdf_resolve_link(self, uri, xp, yp):
27051: (8)                  r"""
27052: (8)                  Class-aware wrapper for `::pdf_resolve_link()`.
27053: (8)                  This method has out-params. Python/C# wrappers look like:
27054: (12)                     `pdf_resolve_link(const char *uri)` => `(int, float xp, float yp)`
27055: (8)                  """
27056: (8)                  return _mupdf.PdfDocument_pdf_resolve_link(self, uri, xp, yp)
27057: (4)              def pdf_rewrite_images(self, opts):
27058: (8)                  r""" Class-aware wrapper for `::pdf_rewrite_images()`."""
27059: (8)                  return _mupdf.PdfDocument_pdf_rewrite_images(self, opts)
27060: (4)              def pdf_run_document_structure(self, dev, cookie):
27061: (8)                  r""" Class-aware wrapper for `::pdf_run_document_structure()`."""
27062: (8)                  return _mupdf.PdfDocument_pdf_run_document_structure(self, dev,
cookie)
27063: (4)              def pdf_run_glyph(self, resources, contents, dev, ctm, gstate,
default_cs):
27064: (8)                  r""" Class-aware wrapper for `::pdf_run_glyph()`."""
27065: (8)                  return _mupdf.PdfDocument_pdf_run_glyph(self, resources, contents,
dev, ctm, gstate, default_cs)
27066: (4)              def pdf_save_document(self, filename, opts):
27067: (8)                  r""" Class-aware wrapper for `::pdf_save_document()`."""
27068: (8)                  return _mupdf.PdfDocument_pdf_save_document(self, filename, opts)
27069: (4)              def pdf_save_journal(self, filename):
27070: (8)                  r""" Class-aware wrapper for `::pdf_save_journal()`."""
27071: (8)                  return _mupdf.PdfDocument_pdf_save_journal(self, filename)
27072: (4)              def pdf_save_snapshot(self, filename):
```

```
27073: (8)                  r""" Class-aware wrapper for `::pdf_save_snapshot()`."""
27074: (8)                  return _mupdf.PdfDocument_pdf_save_snapshot(self, filename)
27075: (4)              def pdf_select_layer_config(self, config_num):
27076: (8)                  r""" Class-aware wrapper for `::pdf_select_layer_config()`."""
27077: (8)                  return _mupdf.PdfDocument_pdf_select_layer_config(self, config_num)
27078: (4)              def pdf_select_layer_config_ui(self, ui):
27079: (8)                  r""" Class-aware wrapper for `::pdf_select_layer_config_ui()`."""
27080: (8)                  return _mupdf.PdfDocument_pdf_select_layer_config_ui(self, ui)
27081: (4)              def pdf_serialise_journal(self, out):
27082: (8)                  r""" Class-aware wrapper for `::pdf_serialise_journal()`."""
27083: (8)                  return _mupdf.PdfDocument_pdf_serialise_journal(self, out)
27084: (4)              def pdf_set_annot_field_value(self, widget, text, ignore_trigger_events):
27085: (8)                  r""" Class-aware wrapper for `::pdf_set_annot_field_value()`."""
27086: (8)                  return _mupdf.PdfDocument_pdf_set_annot_field_value(self, widget,
text, ignore_trigger_events)
27087: (4)              def pdf_set_doc_event_callback(self, event_cb, free_event_data_cb, data):
27088: (8)                  r""" Class-aware wrapper for `::pdf_set_doc_event_callback()`."""
27089: (8)                  return _mupdf.PdfDocument_pdf_set_doc_event_callback(self, event_cb,
free_event_data_cb, data)
27090: (4)              def pdf_set_document_language(self, lang):
27091: (8)                  r""" Class-aware wrapper for `::pdf_set_document_language()`."""
27092: (8)                  return _mupdf.PdfDocument_pdf_set_document_language(self, lang)
27093: (4)              def pdf_set_field_value(self, field, text, ignore_trigger_events):
27094: (8)                  r""" Class-aware wrapper for `::pdf_set_field_value()`."""
27095: (8)                  return _mupdf.PdfDocument_pdf_set_field_value(self, field, text,
ignore_trigger_events)
27096: (4)              def pdf_set_layer_config_as_default(self):
27097: (8)                  r""" Class-aware wrapper for `::pdf_set_layer_config_as_default()`."""
27098: (8)                  return _mupdf.PdfDocument_pdf_set_layer_config_as_default(self)
27099: (4)              def pdf_set_page_labels(self, index, style, prefix, start):
27100: (8)                  r""" Class-aware wrapper for `::pdf_set_page_labels()`."""
27101: (8)                  return _mupdf.PdfDocument_pdf_set_page_labels(self, index, style,
prefix, start)
27102: (4)              def pdf_set_populating_xref_trailer(self, trailer):
27103: (8)                  r""" Class-aware wrapper for `::pdf_set_populating_xref_trailer()`."""
27104: (8)                  return _mupdf.PdfDocument_pdf_set_populating_xref_trailer(self,
trailer)
27105: (4)              def pdf_signature_byte_range(self, signature, byte_range):
27106: (8)                  r""" Class-aware wrapper for `::pdf_signature_byte_range()`."""
27107: (8)                  return _mupdf.PdfDocument_pdf_signature_byte_range(self, signature,
byte_range)
27108: (4)              def pdf_signature_contents(self, signature, contents):
27109: (8)                  r"""
27110: (8)                  Class-aware wrapper for `::pdf_signature_contents()`.
27111: (8)                  This method has out-params. Python/C# wrappers look like:
27112: (12)                     `pdf_signature_contents(::pdf_obj *signature)` => `(size_t, char
*contents)`
27113: (8)                  """
27114: (8)                  return _mupdf.PdfDocument_pdf_signature_contents(self, signature,
contents)
27115: (4)              def pdf_signature_hash_bytes(self, signature):
27116: (8)                  r""" Class-aware wrapper for `::pdf_signature_hash_bytes()`."""
27117: (8)                  return _mupdf.PdfDocument_pdf_signature_hash_bytes(self, signature)
27118: (4)              def pdf_signature_incremental_change_since_signing(self, signature):
27119: (8)                  r""" Class-aware wrapper for
`::pdf_signature_incremental_change_since_signing()`."""
27120: (8)                  return
_mupdf.PdfDocument_pdf_signature_incremental_change_since_signing(self, signature)
27121: (4)              def pdf_signature_is_signed(self, field):
27122: (8)                  r""" Class-aware wrapper for `::pdf_signature_is_signed()`."""
27123: (8)                  return _mupdf.PdfDocument_pdf_signature_is_signed(self, field)
27124: (4)              def pdf_signature_set_value(self, field, signer, stime):
27125: (8)                  r""" Class-aware wrapper for `::pdf_signature_set_value()`."""
27126: (8)                  return _mupdf.PdfDocument_pdf_signature_set_value(self, field, signer,
stime)
27127: (4)              def pdf_subset_fonts(self, pages_len, pages):
27128: (8)                  r""" Class-aware wrapper for `::pdf_subset_fonts()`."""
27129: (8)                  return _mupdf.PdfDocument_pdf_subset_fonts(self, pages_len, pages)
27130: (4)              def pdf_subset_fonts2(self, pages):
```

```
27131: (8)                  r""" Class-aware wrapper for `::pdf_subset_fonts2()`.   Swig-friendly
wrapper for pdf_subset_fonts()."""
27132: (8)                  return _mupdf.PdfDocument_pdf_subset_fonts2(self, pages)
27133: (4)              def pdf_toggle_layer_config_ui(self, ui):
27134: (8)                  r""" Class-aware wrapper for `::pdf_toggle_layer_config_ui()`."""
27135: (8)                  return _mupdf.PdfDocument_pdf_toggle_layer_config_ui(self, ui)
27136: (4)              def pdf_trailer(self):
27137: (8)                  r""" Class-aware wrapper for `::pdf_trailer()`."""
27138: (8)                  return _mupdf.PdfDocument_pdf_trailer(self)
27139: (4)              def pdf_undo(self):
27140: (8)                  r""" Class-aware wrapper for `::pdf_undo()`."""
27141: (8)                  return _mupdf.PdfDocument_pdf_undo(self)
27142: (4)              def pdf_undoredo_state(self, steps):
27143: (8)                  r"""
27144: (8)                  Class-aware wrapper for `::pdf_undoredo_state()`.
27145: (8)                  This method has out-params. Python/C# wrappers look like:
27146: (12)                     `pdf_undoredo_state()` => `(int, int steps)`
27147: (8)                  """
27148: (8)                  return _mupdf.PdfDocument_pdf_undoredo_state(self, steps)
27149: (4)              def pdf_undoredo_step(self, step):
27150: (8)                  r""" Class-aware wrapper for `::pdf_undoredo_step()`."""
27151: (8)                  return _mupdf.PdfDocument_pdf_undoredo_step(self, step)
27152: (4)              def pdf_update_object(self, num, obj):
27153: (8)                  r""" Class-aware wrapper for `::pdf_update_object()`."""
27154: (8)                  return _mupdf.PdfDocument_pdf_update_object(self, num, obj)
27155: (4)              def pdf_update_stream(self, ref, buf, compressed):
27156: (8)                  r""" Class-aware wrapper for `::pdf_update_stream()`."""
27157: (8)                  return _mupdf.PdfDocument_pdf_update_stream(self, ref, buf,
compressed)
27158: (4)              def pdf_update_xobject(self, xobj, bbox, mat, res, buffer):
27159: (8)                  r""" Class-aware wrapper for `::pdf_update_xobject()`."""
27160: (8)                  return _mupdf.PdfDocument_pdf_update_xobject(self, xobj, bbox, mat,
res, buffer)
27161: (4)              def pdf_validate_change_history(self):
27162: (8)                  r""" Class-aware wrapper for `::pdf_validate_change_history()`."""
27163: (8)                  return _mupdf.PdfDocument_pdf_validate_change_history(self)
27164: (4)              def pdf_validate_changes(self, version):
27165: (8)                  r""" Class-aware wrapper for `::pdf_validate_changes()`."""
27166: (8)                  return _mupdf.PdfDocument_pdf_validate_changes(self, version)
27167: (4)              def pdf_version(self):
27168: (8)                  r""" Class-aware wrapper for `::pdf_version()`."""
27169: (8)                  return _mupdf.PdfDocument_pdf_version(self)
27170: (4)              def pdf_was_pure_xfa(self):
27171: (8)                  r""" Class-aware wrapper for `::pdf_was_pure_xfa()`."""
27172: (8)                  return _mupdf.PdfDocument_pdf_was_pure_xfa(self)
27173: (4)              def pdf_was_repaired(self):
27174: (8)                  r""" Class-aware wrapper for `::pdf_was_repaired()`."""
27175: (8)                  return _mupdf.PdfDocument_pdf_was_repaired(self)
27176: (4)              def pdf_write_document(self, out, opts):
27177: (8)                  r""" Class-aware wrapper for `::pdf_write_document()`."""
27178: (8)                  return _mupdf.PdfDocument_pdf_write_document(self, out, opts)
27179: (4)              def pdf_write_journal(self, out):
27180: (8)                  r""" Class-aware wrapper for `::pdf_write_journal()`."""
27181: (8)                  return _mupdf.PdfDocument_pdf_write_journal(self, out)
27182: (4)              def pdf_write_snapshot(self, out):
27183: (8)                  r""" Class-aware wrapper for `::pdf_write_snapshot()`."""
27184: (8)                  return _mupdf.PdfDocument_pdf_write_snapshot(self, out)
27185: (4)              def pdf_xref_ensure_incremental_object(self, num):
27186: (8)                  r""" Class-aware wrapper for
`::pdf_xref_ensure_incremental_object()`."""
27187: (8)                  return _mupdf.PdfDocument_pdf_xref_ensure_incremental_object(self,
num)
27188: (4)              def pdf_xref_ensure_local_object(self, num):
27189: (8)                  r""" Class-aware wrapper for `::pdf_xref_ensure_local_object()`."""
27190: (8)                  return _mupdf.PdfDocument_pdf_xref_ensure_local_object(self, num)
27191: (4)              def pdf_xref_entry_map(self, fn, arg):
27192: (8)                  r""" Class-aware wrapper for `::pdf_xref_entry_map()`."""
27193: (8)                  return _mupdf.PdfDocument_pdf_xref_entry_map(self, fn, arg)
27194: (4)              def pdf_xref_is_incremental(self, num):
```

```
27195: (8)                  r""" Class-aware wrapper for `::pdf_xref_is_incremental()`."""
27196: (8)                  return _mupdf.PdfDocument_pdf_xref_is_incremental(self, num)
27197: (4)              def pdf_xref_len(self):
27198: (8)                  r""" Class-aware wrapper for `::pdf_xref_len()`."""
27199: (8)                  return _mupdf.PdfDocument_pdf_xref_len(self)
27200: (4)              def pdf_xref_obj_is_unsaved_signature(self, obj):
27201: (8)                  r""" Class-aware wrapper for
`::pdf_xref_obj_is_unsaved_signature()`."""
27202: (8)                  return _mupdf.PdfDocument_pdf_xref_obj_is_unsaved_signature(self, obj)
27203: (4)              def pdf_xref_remove_unsaved_signature(self, field):
27204: (8)                  r""" Class-aware wrapper for
`::pdf_xref_remove_unsaved_signature()`."""
27205: (8)                  return _mupdf.PdfDocument_pdf_xref_remove_unsaved_signature(self,
field)
27206: (4)              def pdf_xref_store_unsaved_signature(self, field, signer):
27207: (8)                  r""" Class-aware wrapper for
`::pdf_xref_store_unsaved_signature()`."""
27208: (8)                  return _mupdf.PdfDocument_pdf_xref_store_unsaved_signature(self,
field, signer)
27209: (4)              def super(self):
27210: (8)                  r""" Returns wrapper for .super member."""
27211: (8)                  return _mupdf.PdfDocument_super(self)
27212: (4)              def __init__(self, *args):
27213: (8)                  r"""
27214: (8)                  *Overload 1:*
27215: (8)                  == Constructors.  Constructor using `pdf_create_document()`.
27216: (8)                  |
27217: (8)                  *Overload 2:*
27218: (8)                  Constructor using `pdf_document_from_fz_document()`.
27219: (8)                  |
27220: (8)                  *Overload 3:*
27221: (8)                  Constructor using `pdf_open_document()`.
27222: (8)                  |
27223: (8)                  *Overload 4:*
27224: (8)                  Constructor using `pdf_open_document_with_stream()`.
27225: (8)                  |
27226: (8)                  *Overload 5:*
27227: (8)                  Copy constructor using `pdf_keep_document()`.
27228: (8)                  |
27229: (8)                  *Overload 6:*
27230: (8)                  Constructor using raw copy of pre-existing `::pdf_document`.
27231: (8)                  """
27232: (8)                  _mupdf.PdfDocument_swiginit(self, _mupdf.new_PdfDocument(*args))
27233: (4)              __swig_destroy__ = _mupdf.delete_PdfDocument
27234: (4)              def m_internal_value(self):
27235: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27236: (8)                  return _mupdf.PdfDocument_m_internal_value(self)
27237: (4)              m_internal = property(_mupdf.PdfDocument_m_internal_get,
_mupdf.PdfDocument_m_internal_set, doc=r""" Pointer to wrapped data.""")
27238: (4)              s_num_instances = property(_mupdf.PdfDocument_s_num_instances_get,
_mupdf.PdfDocument_s_num_instances_set)
27239: (0)          _mupdf.PdfDocument_swigregister(PdfDocument)
27240: (0)          class PdfEmbeddedFileParams(object):
27241: (4)              r""" Wrapper class for struct `pdf_embedded_file_params`. Not copyable or
assignable."""
27242: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27243: (4)              __repr__ = _swig_repr
27244: (4)              def __init__(self, *args):
27245: (8)                  r"""
27246: (8)                  *Overload 1:*
27247: (8)                  Default constructor, sets `m_internal` to null.
27248: (8)                  |
27249: (8)                  *Overload 2:*
27250: (8)                  Constructor using raw copy of pre-existing
`::pdf_embedded_file_params`.
27251: (8)                  """
27252: (8)                  _mupdf.PdfEmbeddedFileParams_swiginit(self,
```

```
_mupdf.new_PdfEmbeddedFileParams(*args))
27253: (4)              __swig_destroy__ = _mupdf.delete_PdfEmbeddedFileParams
27254: (4)              def m_internal_value(self):
27255: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27256: (8)                  return _mupdf.PdfEmbeddedFileParams_m_internal_value(self)
27257: (4)              m_internal = property(_mupdf.PdfEmbeddedFileParams_m_internal_get,
_mupdf.PdfEmbeddedFileParams_m_internal_set, doc=r""" Pointer to wrapped data.""")
27258: (4)              s_num_instances =
property(_mupdf.PdfEmbeddedFileParams_s_num_instances_get,
_mupdf.PdfEmbeddedFileParams_s_num_instances_set)
27259: (0)          _mupdf.PdfEmbeddedFileParams_swigregister(PdfEmbeddedFileParams)
27260: (0)          class PdfFilterFactory(object):
27261: (4)              r""" Wrapper class for struct `pdf_filter_factory`."""
27262: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27263: (4)              __repr__ = _swig_repr
27264: (4)              def __init__(self, *args):
27265: (8)                  r"""
27266: (8)                  *Overload 1:*
27267: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
27268: (8)                  |
27269: (8)                  *Overload 2:*
27270: (8)                  Constructor using raw copy of pre-existing `::pdf_filter_factory`.
27271: (8)                  |
27272: (8)                  *Overload 3:*
27273: (8)                  Constructor using raw copy of pre-existing `::pdf_filter_factory`.
27274: (8)                  """
27275: (8)                  _mupdf.PdfFilterFactory_swiginit(self,
_mupdf.new_PdfFilterFactory(*args))
27276: (4)              def internal(self, *args):
27277: (8)                  r"""
27278: (8)                  *Overload 1:*
27279: (8)                  Access as underlying struct.
27280: (8)                  |
27281: (8)                  *Overload 2:*
27282: (8)                  Access as underlying struct.
27283: (8)                  """
27284: (8)                  return _mupdf.PdfFilterFactory_internal(self, *args)
27285: (4)              __swig_destroy__ = _mupdf.delete_PdfFilterFactory
27286: (4)              filter = property(_mupdf.PdfFilterFactory_filter_get,
_mupdf.PdfFilterFactory_filter_set)
27287: (4)              options = property(_mupdf.PdfFilterFactory_options_get,
_mupdf.PdfFilterFactory_options_set)
27288: (4)              s_num_instances = property(_mupdf.PdfFilterFactory_s_num_instances_get,
_mupdf.PdfFilterFactory_s_num_instances_set)
27289: (4)              def to_string(self):
27290: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
27291: (8)                  return _mupdf.PdfFilterFactory_to_string(self)
27292: (4)              def __eq__(self, rhs):
27293: (8)                  r""" Comparison method."""
27294: (8)                  return _mupdf.PdfFilterFactory___eq__(self, rhs)
27295: (4)              def __ne__(self, rhs):
27296: (8)                  r""" Comparison method."""
27297: (8)                  return _mupdf.PdfFilterFactory___ne__(self, rhs)
27298: (0)          _mupdf.PdfFilterFactory_swigregister(PdfFilterFactory)
27299: (0)          class PdfFilterFactory2(PdfFilterFactory):
27300: (4)              r""" Wrapper class for struct pdf_filter_factory with virtual fns for each
fnptr; this is for use as a SWIG Director class."""
27301: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27302: (4)              __repr__ = _swig_repr
27303: (4)              def __init__(self):
27304: (8)                  r""" == Constructor."""
27305: (8)                  if self.__class__ == PdfFilterFactory2:
27306: (12)                     _self = None
27307: (8)                  else:
```

```
27308: (12)                     _self = self
27309: (8)                  _mupdf.PdfFilterFactory2_swiginit(self,
_mupdf.new_PdfFilterFactory2(_self, ))
27310: (4)              def use_virtual_filter(self, use=True):
27311: (8)                  r"""
27312: (8)                  These methods set the function pointers in *m_internal
27313: (8)                  to point to internal callbacks that call our virtual methods.
27314: (8)                  """
27315: (8)                  return _mupdf.PdfFilterFactory2_use_virtual_filter(self, use)
27316: (4)              def filter(self, arg_0, arg_1, arg_2, arg_3, arg_4, arg_5):
27317: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
27318: (8)                  return _mupdf.PdfFilterFactory2_filter(self, arg_0, arg_1, arg_2,
arg_3, arg_4, arg_5)
27319: (4)              __swig_destroy__ = _mupdf.delete_PdfFilterFactory2
27320: (4)              def __disown__(self):
27321: (8)                  self.this.disown()
27322: (8)                  _mupdf.disown_PdfFilterFactory2(self)
27323: (8)                  return weakref.proxy(self)
27324: (0)          _mupdf.PdfFilterFactory2_swigregister(PdfFilterFactory2)
27325: (0)          class PdfFilterOptions(object):
27326: (4)              r""" Wrapper class for struct `pdf_filter_options`."""
27327: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27328: (4)              __repr__ = _swig_repr
27329: (4)              def add_factory(self, factory):
27330: (8)                  r""" We use default copy constructor and operator=.  Appends `factory`
to internal vector and updates this->filters."""
27331: (8)                  return _mupdf.PdfFilterOptions_add_factory(self, factory)
27332: (4)              def __init__(self, *args):
27333: (8)                  r"""
27334: (8)                  *Overload 1:*
27335: (8)                  Default constructor initialises all fields to null/zero.
27336: (8)                  |
27337: (8)                  *Overload 2:*
27338: (8)                  Constructor using raw copy of pre-existing `::pdf_filter_options`.
27339: (8)                  |
27340: (8)                  *Overload 3:*
27341: (8)                  Constructor using raw copy of pre-existing `::pdf_filter_options`.
27342: (8)                  """
27343: (8)                  _mupdf.PdfFilterOptions_swiginit(self,
_mupdf.new_PdfFilterOptions(*args))
27344: (4)              def internal(self, *args):
27345: (8)                  r"""
27346: (8)                  *Overload 1:*
27347: (8)                  Access as underlying struct.
27348: (8)                  |
27349: (8)                  *Overload 2:*
27350: (8)                  Access as underlying struct.
27351: (8)                  """
27352: (8)                  return _mupdf.PdfFilterOptions_internal(self, *args)
27353: (4)              __swig_destroy__ = _mupdf.delete_PdfFilterOptions
27354: (4)              recurse = property(_mupdf.PdfFilterOptions_recurse_get,
_mupdf.PdfFilterOptions_recurse_set)
27355: (4)              instance_forms = property(_mupdf.PdfFilterOptions_instance_forms_get,
_mupdf.PdfFilterOptions_instance_forms_set)
27356: (4)              ascii = property(_mupdf.PdfFilterOptions_ascii_get,
_mupdf.PdfFilterOptions_ascii_set)
27357: (4)              no_update = property(_mupdf.PdfFilterOptions_no_update_get,
_mupdf.PdfFilterOptions_no_update_set)
27358: (4)              opaque = property(_mupdf.PdfFilterOptions_opaque_get,
_mupdf.PdfFilterOptions_opaque_set)
27359: (4)              complete = property(_mupdf.PdfFilterOptions_complete_get,
_mupdf.PdfFilterOptions_complete_set)
27360: (4)              filters = property(_mupdf.PdfFilterOptions_filters_get,
_mupdf.PdfFilterOptions_filters_set)
27361: (4)              newlines = property(_mupdf.PdfFilterOptions_newlines_get,
_mupdf.PdfFilterOptions_newlines_set)
27362: (4)              s_num_instances = property(_mupdf.PdfFilterOptions_s_num_instances_get,
```

```
_mupdf.PdfFilterOptions_s_num_instances_set)
27363: (4)              def to_string(self):
27364: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
27365: (8)                  return _mupdf.PdfFilterOptions_to_string(self)
27366: (4)              def __eq__(self, rhs):
27367: (8)                  r""" Comparison method."""
27368: (8)                  return _mupdf.PdfFilterOptions___eq__(self, rhs)
27369: (4)              def __ne__(self, rhs):
27370: (8)                  r""" Comparison method."""
27371: (8)                  return _mupdf.PdfFilterOptions___ne__(self, rhs)
27372: (4)              m_filters = property(_mupdf.PdfFilterOptions_m_filters_get,
_mupdf.PdfFilterOptions_m_filters_set)
27373: (0)          _mupdf.PdfFilterOptions_swigregister(PdfFilterOptions)
27374: (0)          class PdfFilterOptions2(PdfFilterOptions):
27375: (4)              r""" Wrapper class for struct pdf_filter_options with virtual fns for each
fnptr; this is for use as a SWIG Director class."""
27376: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27377: (4)              __repr__ = _swig_repr
27378: (4)              def __init__(self):
27379: (8)                  r""" == Constructor."""
27380: (8)                  if self.__class__ == PdfFilterOptions2:
27381: (12)                     _self = None
27382: (8)                  else:
27383: (12)                     _self = self
27384: (8)                  _mupdf.PdfFilterOptions2_swiginit(self,
_mupdf.new_PdfFilterOptions2(_self, ))
27385: (4)              def use_virtual_complete(self, use=True):
27386: (8)                  r"""
27387: (8)                  These methods set the function pointers in *m_internal
27388: (8)                  to point to internal callbacks that call our virtual methods.
27389: (8)                  """
27390: (8)                  return _mupdf.PdfFilterOptions2_use_virtual_complete(self, use)
27391: (4)              def complete(self, arg_0, arg_1):
27392: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
27393: (8)                  return _mupdf.PdfFilterOptions2_complete(self, arg_0, arg_1)
27394: (4)              __swig_destroy__ = _mupdf.delete_PdfFilterOptions2
27395: (4)              def __disown__(self):
27396: (8)                  self.this.disown()
27397: (8)                  _mupdf.disown_PdfFilterOptions2(self)
27398: (8)                  return weakref.proxy(self)
27399: (0)          _mupdf.PdfFilterOptions2_swigregister(PdfFilterOptions2)
27400: (0)          class PdfFontDesc(object):
27401: (4)              r""" Wrapper class for struct `pdf_font_desc`. Not copyable or
assignable."""
27402: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27403: (4)              __repr__ = _swig_repr
27404: (4)              def pdf_add_hmtx(self, lo, hi, w):
27405: (8)                  r""" Class-aware wrapper for `::pdf_add_hmtx()`."""
27406: (8)                  return _mupdf.PdfFontDesc_pdf_add_hmtx(self, lo, hi, w)
27407: (4)              def pdf_add_vmtx(self, lo, hi, x, y, w):
27408: (8)                  r""" Class-aware wrapper for `::pdf_add_vmtx()`."""
27409: (8)                  return _mupdf.PdfFontDesc_pdf_add_vmtx(self, lo, hi, x, y, w)
27410: (4)              def pdf_end_hmtx(self):
27411: (8)                  r""" Class-aware wrapper for `::pdf_end_hmtx()`."""
27412: (8)                  return _mupdf.PdfFontDesc_pdf_end_hmtx(self)
27413: (4)              def pdf_end_vmtx(self):
27414: (8)                  r""" Class-aware wrapper for `::pdf_end_vmtx()`."""
27415: (8)                  return _mupdf.PdfFontDesc_pdf_end_vmtx(self)
27416: (4)              def pdf_font_cid_to_gid(self, cid):
27417: (8)                  r""" Class-aware wrapper for `::pdf_font_cid_to_gid()`."""
27418: (8)                  return _mupdf.PdfFontDesc_pdf_font_cid_to_gid(self, cid)
27419: (4)              def pdf_set_default_hmtx(self, w):
27420: (8)                  r""" Class-aware wrapper for `::pdf_set_default_hmtx()`."""
27421: (8)                  return _mupdf.PdfFontDesc_pdf_set_default_hmtx(self, w)
27422: (4)              def pdf_set_default_vmtx(self, y, w):
```

```
27423: (8)                  r""" Class-aware wrapper for `::pdf_set_default_vmtx()`."""
27424: (8)                  return _mupdf.PdfFontDesc_pdf_set_default_vmtx(self, y, w)
27425: (4)              def pdf_set_font_wmode(self, wmode):
27426: (8)                  r""" Class-aware wrapper for `::pdf_set_font_wmode()`."""
27427: (8)                  return _mupdf.PdfFontDesc_pdf_set_font_wmode(self, wmode)
27428: (4)              def __init__(self, *args):
27429: (8)                  r"""
27430: (8)                  *Overload 1:*
27431: (8)                  == Constructors.  Constructor using `pdf_new_font_desc()`.
27432: (8)                  |
27433: (8)                  *Overload 2:*
27434: (8)                  Constructor using raw copy of pre-existing `::pdf_font_desc`.
27435: (8)                  """
27436: (8)                  _mupdf.PdfFontDesc_swiginit(self, _mupdf.new_PdfFontDesc(*args))
27437: (4)              __swig_destroy__ = _mupdf.delete_PdfFontDesc
27438: (4)              def m_internal_value(self):
27439: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27440: (8)                  return _mupdf.PdfFontDesc_m_internal_value(self)
27441: (4)              m_internal = property(_mupdf.PdfFontDesc_m_internal_get,
_mupdf.PdfFontDesc_m_internal_set, doc=r""" Pointer to wrapped data.""")
27442: (4)              s_num_instances = property(_mupdf.PdfFontDesc_s_num_instances_get,
_mupdf.PdfFontDesc_s_num_instances_set)
27443: (0)          _mupdf.PdfFontDesc_swigregister(PdfFontDesc)
27444: (0)          class PdfFontResourceKey(object):
27445: (4)              r""" Wrapper class for struct `pdf_font_resource_key`. Not copyable or
assignable."""
27446: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27447: (4)              __repr__ = _swig_repr
27448: (4)              def __init__(self, *args):
27449: (8)                  r"""
27450: (8)                  *Overload 1:*
27451: (8)                  Default constructor, sets `m_internal` to null.
27452: (8)                  |
27453: (8)                  *Overload 2:*
27454: (8)                  Constructor using raw copy of pre-existing `::pdf_font_resource_key`.
27455: (8)                  """
27456: (8)                  _mupdf.PdfFontResourceKey_swiginit(self,
_mupdf.new_PdfFontResourceKey(*args))
27457: (4)              __swig_destroy__ = _mupdf.delete_PdfFontResourceKey
27458: (4)              def m_internal_value(self):
27459: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27460: (8)                  return _mupdf.PdfFontResourceKey_m_internal_value(self)
27461: (4)              m_internal = property(_mupdf.PdfFontResourceKey_m_internal_get,
_mupdf.PdfFontResourceKey_m_internal_set, doc=r""" Pointer to wrapped data.""")
27462: (4)              s_num_instances = property(_mupdf.PdfFontResourceKey_s_num_instances_get,
_mupdf.PdfFontResourceKey_s_num_instances_set)
27463: (0)          _mupdf.PdfFontResourceKey_swigregister(PdfFontResourceKey)
27464: (0)          class PdfFunction(object):
27465: (4)              r""" Wrapper class for struct `pdf_function`."""
27466: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27467: (4)              __repr__ = _swig_repr
27468: (4)              def pdf_eval_function(self, _in, inlen, out, outlen):
27469: (8)                  r"""
27470: (8)                  Class-aware wrapper for `::pdf_eval_function()`.
27471: (8)                  This method has out-params. Python/C# wrappers look like:
27472: (12)                     `pdf_eval_function(const float *in, int inlen, int outlen)` =>
float out
27473: (8)                  """
27474: (8)                  return _mupdf.PdfFunction_pdf_eval_function(self, _in, inlen, out,
outlen)
27475: (4)              def pdf_function_size(self):
27476: (8)                  r""" Class-aware wrapper for `::pdf_function_size()`."""
27477: (8)                  return _mupdf.PdfFunction_pdf_function_size(self)
27478: (4)              def __init__(self, *args):
27479: (8)                  r"""
```

```
27480: (8)                  *Overload 1:*
27481: (8)                  Copy constructor using `pdf_keep_function()`.
27482: (8)                  |
27483: (8)                  *Overload 2:*
27484: (8)                  Default constructor, sets `m_internal` to null.
27485: (8)                  |
27486: (8)                  *Overload 3:*
27487: (8)                  Constructor using raw copy of pre-existing `::pdf_function`.
27488: (8)                  """
27489: (8)                  _mupdf.PdfFunction_swiginit(self, _mupdf.new_PdfFunction(*args))
27490: (4)              __swig_destroy__ = _mupdf.delete_PdfFunction
27491: (4)              def m_internal_value(self):
27492: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27493: (8)                  return _mupdf.PdfFunction_m_internal_value(self)
27494: (4)              m_internal = property(_mupdf.PdfFunction_m_internal_get,
_mupdf.PdfFunction_m_internal_set, doc=r""" Pointer to wrapped data.""")
27495: (4)              s_num_instances = property(_mupdf.PdfFunction_s_num_instances_get,
_mupdf.PdfFunction_s_num_instances_set)
27496: (0)          _mupdf.PdfFunction_swigregister(PdfFunction)
27497: (0)          class PdfGraftMap(object):
27498: (4)              r""" Wrapper class for struct `pdf_graft_map`."""
27499: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27500: (4)              __repr__ = _swig_repr
27501: (4)              def pdf_graft_mapped_object(self, obj):
27502: (8)                  r""" Class-aware wrapper for `::pdf_graft_mapped_object()`."""
27503: (8)                  return _mupdf.PdfGraftMap_pdf_graft_mapped_object(self, obj)
27504: (4)              def pdf_graft_mapped_page(self, page_to, src, page_from):
27505: (8)                  r""" Class-aware wrapper for `::pdf_graft_mapped_page()`."""
27506: (8)                  return _mupdf.PdfGraftMap_pdf_graft_mapped_page(self, page_to, src,
page_from)
27507: (4)              def __init__(self, *args):
27508: (8)                  r"""
27509: (8)                  *Overload 1:*
27510: (8)                  == Constructors.  Constructor using `pdf_new_graft_map()`.
27511: (8)                  |
27512: (8)                  *Overload 2:*
27513: (8)                  Copy constructor using `pdf_keep_graft_map()`.
27514: (8)                  |
27515: (8)                  *Overload 3:*
27516: (8)                  Default constructor, sets `m_internal` to null.
27517: (8)                  |
27518: (8)                  *Overload 4:*
27519: (8)                  Constructor using raw copy of pre-existing `::pdf_graft_map`.
27520: (8)                  """
27521: (8)                  _mupdf.PdfGraftMap_swiginit(self, _mupdf.new_PdfGraftMap(*args))
27522: (4)              __swig_destroy__ = _mupdf.delete_PdfGraftMap
27523: (4)              def m_internal_value(self):
27524: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27525: (8)                  return _mupdf.PdfGraftMap_m_internal_value(self)
27526: (4)              m_internal = property(_mupdf.PdfGraftMap_m_internal_get,
_mupdf.PdfGraftMap_m_internal_set, doc=r""" Pointer to wrapped data.""")
27527: (4)              s_num_instances = property(_mupdf.PdfGraftMap_s_num_instances_get,
_mupdf.PdfGraftMap_s_num_instances_set)
27528: (0)          _mupdf.PdfGraftMap_swigregister(PdfGraftMap)
27529: (0)          class PdfGstate(object):
27530: (4)              r""" Wrapper class for struct `pdf_gstate`. Not copyable or assignable."""
27531: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27532: (4)              __repr__ = _swig_repr
27533: (4)              def __init__(self, *args):
27534: (8)                  r"""
27535: (8)                  *Overload 1:*
27536: (8)                  Default constructor, sets `m_internal` to null.
27537: (8)                  |
27538: (8)                  *Overload 2:*
27539: (8)                  Constructor using raw copy of pre-existing `::pdf_gstate`.
```

```
27540: (8)                  """
27541: (8)                  _mupdf.PdfGstate_swiginit(self, _mupdf.new_PdfGstate(*args))
27542: (4)              __swig_destroy__ = _mupdf.delete_PdfGstate
27543: (4)              def m_internal_value(self):
27544: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27545: (8)                  return _mupdf.PdfGstate_m_internal_value(self)
27546: (4)              m_internal = property(_mupdf.PdfGstate_m_internal_get,
_mupdf.PdfGstate_m_internal_set, doc=r""" Pointer to wrapped data.""")
27547: (4)              s_num_instances = property(_mupdf.PdfGstate_s_num_instances_get,
_mupdf.PdfGstate_s_num_instances_set)
27548: (0)          _mupdf.PdfGstate_swigregister(PdfGstate)
27549: (0)          class PdfHintPage(object):
27550: (4)              r""" Wrapper class for struct `pdf_hint_page`. Not copyable or
assignable."""
27551: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27552: (4)              __repr__ = _swig_repr
27553: (4)              def __init__(self, *args):
27554: (8)                  r"""
27555: (8)                  *Overload 1:*
27556: (8)                  Default constructor, sets `m_internal` to null.
27557: (8)                  |
27558: (8)                  *Overload 2:*
27559: (8)                  Constructor using raw copy of pre-existing `::pdf_hint_page`.
27560: (8)                  """
27561: (8)                  _mupdf.PdfHintPage_swiginit(self, _mupdf.new_PdfHintPage(*args))
27562: (4)              __swig_destroy__ = _mupdf.delete_PdfHintPage
27563: (4)              def m_internal_value(self):
27564: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27565: (8)                  return _mupdf.PdfHintPage_m_internal_value(self)
27566: (4)              m_internal = property(_mupdf.PdfHintPage_m_internal_get,
_mupdf.PdfHintPage_m_internal_set, doc=r""" Pointer to wrapped data.""")
27567: (4)              s_num_instances = property(_mupdf.PdfHintPage_s_num_instances_get,
_mupdf.PdfHintPage_s_num_instances_set)
27568: (0)          _mupdf.PdfHintPage_swigregister(PdfHintPage)
27569: (0)          class PdfHintShared(object):
27570: (4)              r""" Wrapper class for struct `pdf_hint_shared`. Not copyable or
assignable."""
27571: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27572: (4)              __repr__ = _swig_repr
27573: (4)              def __init__(self, *args):
27574: (8)                  r"""
27575: (8)                  *Overload 1:*
27576: (8)                  Default constructor, sets `m_internal` to null.
27577: (8)                  |
27578: (8)                  *Overload 2:*
27579: (8)                  Constructor using raw copy of pre-existing `::pdf_hint_shared`.
27580: (8)                  """
27581: (8)                  _mupdf.PdfHintShared_swiginit(self, _mupdf.new_PdfHintShared(*args))
27582: (4)              __swig_destroy__ = _mupdf.delete_PdfHintShared
27583: (4)              def m_internal_value(self):
27584: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27585: (8)                  return _mupdf.PdfHintShared_m_internal_value(self)
27586: (4)              m_internal = property(_mupdf.PdfHintShared_m_internal_get,
_mupdf.PdfHintShared_m_internal_set, doc=r""" Pointer to wrapped data.""")
27587: (4)              s_num_instances = property(_mupdf.PdfHintShared_s_num_instances_get,
_mupdf.PdfHintShared_s_num_instances_set)
27588: (0)          _mupdf.PdfHintShared_swigregister(PdfHintShared)
27589: (0)          class PdfHmtx(object):
27590: (4)              r""" Wrapper class for struct `pdf_hmtx`. Not copyable or assignable."""
27591: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27592: (4)              __repr__ = _swig_repr
27593: (4)              def __init__(self, *args):
27594: (8)                  r"""
```

```
27595: (8)                  *Overload 1:*
27596: (8)                  Default constructor, sets `m_internal` to null.
27597: (8)                  |
27598: (8)                  *Overload 2:*
27599: (8)                  Constructor using raw copy of pre-existing `::pdf_hmtx`.
27600: (8)                  """
27601: (8)                  _mupdf.PdfHmtx_swiginit(self, _mupdf.new_PdfHmtx(*args))
27602: (4)              __swig_destroy__ = _mupdf.delete_PdfHmtx
27603: (4)              def m_internal_value(self):
27604: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27605: (8)                  return _mupdf.PdfHmtx_m_internal_value(self)
27606: (4)              m_internal = property(_mupdf.PdfHmtx_m_internal_get,
_mupdf.PdfHmtx_m_internal_set, doc=r""" Pointer to wrapped data.""")
27607: (4)              s_num_instances = property(_mupdf.PdfHmtx_s_num_instances_get,
_mupdf.PdfHmtx_s_num_instances_set)
27608: (0)          _mupdf.PdfHmtx_swigregister(PdfHmtx)
27609: (0)          class PdfImageRewriterOptions(object):
27610: (4)              r""" Wrapper class for struct `pdf_image_rewriter_options`."""
27611: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27612: (4)              __repr__ = _swig_repr
27613: (4)              def __init__(self, *args):
27614: (8)                  r"""
27615: (8)                  *Overload 1:*
27616: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
27617: (8)                  |
27618: (8)                  *Overload 2:*
27619: (8)                  Constructor using raw copy of pre-existing
`::pdf_image_rewriter_options`.
27620: (8)                  |
27621: (8)                  *Overload 3:*
27622: (8)                  Constructor using raw copy of pre-existing
`::pdf_image_rewriter_options`.
27623: (8)                  """
27624: (8)                  _mupdf.PdfImageRewriterOptions_swiginit(self,
_mupdf.new_PdfImageRewriterOptions(*args))
27625: (4)              def internal(self, *args):
27626: (8)                  r"""
27627: (8)                  *Overload 1:*
27628: (8)                  Access as underlying struct.
27629: (8)                  |
27630: (8)                  *Overload 2:*
27631: (8)                  Access as underlying struct.
27632: (8)                  """
27633: (8)                  return _mupdf.PdfImageRewriterOptions_internal(self, *args)
27634: (4)              __swig_destroy__ = _mupdf.delete_PdfImageRewriterOptions
27635: (4)              color_lossless_image_subsample_method =
property(_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_method_get,
_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_method_set)
27636: (4)              color_lossy_image_subsample_method =
property(_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_method_get,
_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_method_set)
27637: (4)              color_lossless_image_subsample_threshold =
property(_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_threshold_get,
_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_threshold_set)
27638: (4)              color_lossless_image_subsample_to =
property(_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_to_get,
_mupdf.PdfImageRewriterOptions_color_lossless_image_subsample_to_set)
27639: (4)              color_lossy_image_subsample_threshold =
property(_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_threshold_get,
_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_threshold_set)
27640: (4)              color_lossy_image_subsample_to =
property(_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_to_get,
_mupdf.PdfImageRewriterOptions_color_lossy_image_subsample_to_set)
27641: (4)              color_lossless_image_recompress_method =
property(_mupdf.PdfImageRewriterOptions_color_lossless_image_recompress_method_get,
_mupdf.PdfImageRewriterOptions_color_lossless_image_recompress_method_set)
```

```
27642: (4)              color_lossy_image_recompress_method =
property(_mupdf.PdfImageRewriterOptions_color_lossy_image_recompress_method_get,
_mupdf.PdfImageRewriterOptions_color_lossy_image_recompress_method_set)
27643: (4)              color_lossy_image_recompress_quality =
property(_mupdf.PdfImageRewriterOptions_color_lossy_image_recompress_quality_get,
_mupdf.PdfImageRewriterOptions_color_lossy_image_recompress_quality_set)
27644: (4)              color_lossless_image_recompress_quality =
property(_mupdf.PdfImageRewriterOptions_color_lossless_image_recompress_quality_get,
_mupdf.PdfImageRewriterOptions_color_lossless_image_recompress_quality_set)
27645: (4)              gray_lossless_image_subsample_method =
property(_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_method_get,
_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_method_set)
27646: (4)              gray_lossy_image_subsample_method =
property(_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_method_get,
_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_method_set)
27647: (4)              gray_lossless_image_subsample_threshold =
property(_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_threshold_get,
_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_threshold_set)
27648: (4)              gray_lossless_image_subsample_to =
property(_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_to_get,
_mupdf.PdfImageRewriterOptions_gray_lossless_image_subsample_to_set)
27649: (4)              gray_lossy_image_subsample_threshold =
property(_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_threshold_get,
_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_threshold_set)
27650: (4)              gray_lossy_image_subsample_to =
property(_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_to_get,
_mupdf.PdfImageRewriterOptions_gray_lossy_image_subsample_to_set)
27651: (4)              gray_lossless_image_recompress_method =
property(_mupdf.PdfImageRewriterOptions_gray_lossless_image_recompress_method_get,
_mupdf.PdfImageRewriterOptions_gray_lossless_image_recompress_method_set)
27652: (4)              gray_lossy_image_recompress_method =
property(_mupdf.PdfImageRewriterOptions_gray_lossy_image_recompress_method_get,
_mupdf.PdfImageRewriterOptions_gray_lossy_image_recompress_method_set)
27653: (4)              gray_lossy_image_recompress_quality =
property(_mupdf.PdfImageRewriterOptions_gray_lossy_image_recompress_quality_get,
_mupdf.PdfImageRewriterOptions_gray_lossy_image_recompress_quality_set)
27654: (4)              gray_lossless_image_recompress_quality =
property(_mupdf.PdfImageRewriterOptions_gray_lossless_image_recompress_quality_get,
_mupdf.PdfImageRewriterOptions_gray_lossless_image_recompress_quality_set)
27655: (4)              bitonal_image_subsample_method =
property(_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_method_get,
_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_method_set)
27656: (4)              bitonal_image_subsample_threshold =
property(_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_threshold_get,
_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_threshold_set)
27657: (4)              bitonal_image_subsample_to =
property(_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_to_get,
_mupdf.PdfImageRewriterOptions_bitonal_image_subsample_to_set)
27658: (4)              bitonal_image_recompress_method =
property(_mupdf.PdfImageRewriterOptions_bitonal_image_recompress_method_get,
_mupdf.PdfImageRewriterOptions_bitonal_image_recompress_method_set)
27659: (4)              bitonal_image_recompress_quality =
property(_mupdf.PdfImageRewriterOptions_bitonal_image_recompress_quality_get,
_mupdf.PdfImageRewriterOptions_bitonal_image_recompress_quality_set)
27660: (4)              s_num_instances =
property(_mupdf.PdfImageRewriterOptions_s_num_instances_get,
_mupdf.PdfImageRewriterOptions_s_num_instances_set)
27661: (4)              def to_string(self):
27662: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
27663: (8)                  return _mupdf.PdfImageRewriterOptions_to_string(self)
27664: (4)              def __eq__(self, rhs):
27665: (8)                  r""" Comparison method."""
27666: (8)                  return _mupdf.PdfImageRewriterOptions___eq__(self, rhs)
27667: (4)              def __ne__(self, rhs):
27668: (8)                  r""" Comparison method."""
27669: (8)                  return _mupdf.PdfImageRewriterOptions___ne__(self, rhs)
27670: (0)          _mupdf.PdfImageRewriterOptions_swigregister(PdfImageRewriterOptions)
27671: (0)          class PdfJournal(object):
```

```
27672: (4)              r""" Wrapper class for struct `pdf_journal`. Not copyable or
assignable."""
27673: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27674: (4)              __repr__ = _swig_repr
27675: (4)              def pdf_discard_journal(self):
27676: (8)                  r""" Class-aware wrapper for `::pdf_discard_journal()`."""
27677: (8)                  return _mupdf.PdfJournal_pdf_discard_journal(self)
27678: (4)              def __init__(self, *args):
27679: (8)                  r"""
27680: (8)                  *Overload 1:*
27681: (8)                  Default constructor, sets `m_internal` to null.
27682: (8)                  |
27683: (8)                  *Overload 2:*
27684: (8)                  Constructor using raw copy of pre-existing `::pdf_journal`.
27685: (8)                  """
27686: (8)                  _mupdf.PdfJournal_swiginit(self, _mupdf.new_PdfJournal(*args))
27687: (4)              __swig_destroy__ = _mupdf.delete_PdfJournal
27688: (4)              def m_internal_value(self):
27689: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27690: (8)                  return _mupdf.PdfJournal_m_internal_value(self)
27691: (4)              m_internal = property(_mupdf.PdfJournal_m_internal_get,
_mupdf.PdfJournal_m_internal_set, doc=r""" Pointer to wrapped data.""")
27692: (4)              s_num_instances = property(_mupdf.PdfJournal_s_num_instances_get,
_mupdf.PdfJournal_s_num_instances_set)
27693: (0)          _mupdf.PdfJournal_swigregister(PdfJournal)
27694: (0)          class PdfJs(object):
27695: (4)              r""" Wrapper class for struct `pdf_js`. Not copyable or assignable."""
27696: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27697: (4)              __repr__ = _swig_repr
27698: (4)              def pdf_js_event_init(self, target, value, willCommit):
27699: (8)                  r""" Class-aware wrapper for `::pdf_js_event_init()`."""
27700: (8)                  return _mupdf.PdfJs_pdf_js_event_init(self, target, value, willCommit)
27701: (4)              def pdf_js_event_init_keystroke(self, target, evt):
27702: (8)                  r""" Class-aware wrapper for `::pdf_js_event_init_keystroke()`."""
27703: (8)                  return _mupdf.PdfJs_pdf_js_event_init_keystroke(self, target, evt)
27704: (4)              def pdf_js_event_result(self):
27705: (8)                  r""" Class-aware wrapper for `::pdf_js_event_result()`."""
27706: (8)                  return _mupdf.PdfJs_pdf_js_event_result(self)
27707: (4)              def pdf_js_event_result_keystroke(self, evt):
27708: (8)                  r""" Class-aware wrapper for `::pdf_js_event_result_keystroke()`."""
27709: (8)                  return _mupdf.PdfJs_pdf_js_event_result_keystroke(self, evt)
27710: (4)              def pdf_js_event_result_validate(self, newvalue):
27711: (8)                  r"""
27712: (8)                  Class-aware wrapper for `::pdf_js_event_result_validate()`.
27713: (8)                  This method has out-params. Python/C# wrappers look like:
27714: (12)                     `pdf_js_event_result_validate()` => `(int, char *newvalue)`
27715: (8)                  """
27716: (8)                  return _mupdf.PdfJs_pdf_js_event_result_validate(self, newvalue)
27717: (4)              def pdf_js_event_value(self):
27718: (8)                  r""" Class-aware wrapper for `::pdf_js_event_value()`."""
27719: (8)                  return _mupdf.PdfJs_pdf_js_event_value(self)
27720: (4)              def pdf_js_execute(self, name, code, result):
27721: (8)                  r"""
27722: (8)                  Class-aware wrapper for `::pdf_js_execute()`.
27723: (8)                  This method has out-params. Python/C# wrappers look like:
27724: (12)                     `pdf_js_execute(const char *name, const char *code)` => char
*result
27725: (8)                  """
27726: (8)                  return _mupdf.PdfJs_pdf_js_execute(self, name, code, result)
27727: (4)              def __init__(self, *args):
27728: (8)                  r"""
27729: (8)                  *Overload 1:*
27730: (8)                  Default constructor, sets `m_internal` to null.
27731: (8)                  |
27732: (8)                  *Overload 2:*
27733: (8)                  Constructor using raw copy of pre-existing `::pdf_js`.
```

```
27734: (8)                  """
27735: (8)                  _mupdf.PdfJs_swiginit(self, _mupdf.new_PdfJs(*args))
27736: (4)              __swig_destroy__ = _mupdf.delete_PdfJs
27737: (4)              def m_internal_value(self):
27738: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27739: (8)                  return _mupdf.PdfJs_m_internal_value(self)
27740: (4)              m_internal = property(_mupdf.PdfJs_m_internal_get,
_mupdf.PdfJs_m_internal_set, doc=r""" Pointer to wrapped data.""")
27741: (4)              s_num_instances = property(_mupdf.PdfJs_s_num_instances_get,
_mupdf.PdfJs_s_num_instances_set)
27742: (0)          _mupdf.PdfJs_swigregister(PdfJs)
27743: (0)          class PdfJsConsole(object):
27744: (4)              r""" Wrapper class for struct `pdf_js_console`. Not copyable or
assignable."""
27745: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27746: (4)              __repr__ = _swig_repr
27747: (4)              def __init__(self, *args):
27748: (8)                  r"""
27749: (8)                  *Overload 1:*
27750: (8)                  Default constructor, sets `m_internal` to null.
27751: (8)                  |
27752: (8)                  *Overload 2:*
27753: (8)                  Constructor using raw copy of pre-existing `::pdf_js_console`.
27754: (8)                  """
27755: (8)                  _mupdf.PdfJsConsole_swiginit(self, _mupdf.new_PdfJsConsole(*args))
27756: (4)              __swig_destroy__ = _mupdf.delete_PdfJsConsole
27757: (4)              def m_internal_value(self):
27758: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27759: (8)                  return _mupdf.PdfJsConsole_m_internal_value(self)
27760: (4)              m_internal = property(_mupdf.PdfJsConsole_m_internal_get,
_mupdf.PdfJsConsole_m_internal_set, doc=r""" Pointer to wrapped data.""")
27761: (4)              s_num_instances = property(_mupdf.PdfJsConsole_s_num_instances_get,
_mupdf.PdfJsConsole_s_num_instances_set)
27762: (0)          _mupdf.PdfJsConsole_swigregister(PdfJsConsole)
27763: (0)          class PdfKeystrokeEvent(object):
27764: (4)              r""" Wrapper class for struct `pdf_keystroke_event`. Not copyable or
assignable."""
27765: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27766: (4)              __repr__ = _swig_repr
27767: (4)              def __init__(self, *args):
27768: (8)                  r"""
27769: (8)                  *Overload 1:*
27770: (8)                  Default constructor, sets `m_internal` to null.
27771: (8)                  |
27772: (8)                  *Overload 2:*
27773: (8)                  Constructor using raw copy of pre-existing `::pdf_keystroke_event`.
27774: (8)                  """
27775: (8)                  _mupdf.PdfKeystrokeEvent_swiginit(self,
_mupdf.new_PdfKeystrokeEvent(*args))
27776: (4)              __swig_destroy__ = _mupdf.delete_PdfKeystrokeEvent
27777: (4)              def m_internal_value(self):
27778: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27779: (8)                  return _mupdf.PdfKeystrokeEvent_m_internal_value(self)
27780: (4)              m_internal = property(_mupdf.PdfKeystrokeEvent_m_internal_get,
_mupdf.PdfKeystrokeEvent_m_internal_set, doc=r""" Pointer to wrapped data.""")
27781: (4)              s_num_instances = property(_mupdf.PdfKeystrokeEvent_s_num_instances_get,
_mupdf.PdfKeystrokeEvent_s_num_instances_set)
27782: (0)          _mupdf.PdfKeystrokeEvent_swigregister(PdfKeystrokeEvent)
27783: (0)          class PdfLaunchUrlEvent(object):
27784: (4)              r""" Wrapper class for struct `pdf_launch_url_event`. Not copyable or
assignable."""
27785: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27786: (4)              __repr__ = _swig_repr
```

```
27787: (4)              def __init__(self, *args):
27788: (8)                  r"""
27789: (8)                  *Overload 1:*
27790: (8)                  Default constructor, sets `m_internal` to null.
27791: (8)                  |
27792: (8)                  *Overload 2:*
27793: (8)                  Constructor using raw copy of pre-existing `::pdf_launch_url_event`.
27794: (8)                  """
27795: (8)                  _mupdf.PdfLaunchUrlEvent_swiginit(self,
_mupdf.new_PdfLaunchUrlEvent(*args))
27796: (4)              __swig_destroy__ = _mupdf.delete_PdfLaunchUrlEvent
27797: (4)              def m_internal_value(self):
27798: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27799: (8)                  return _mupdf.PdfLaunchUrlEvent_m_internal_value(self)
27800: (4)              m_internal = property(_mupdf.PdfLaunchUrlEvent_m_internal_get,
_mupdf.PdfLaunchUrlEvent_m_internal_set, doc=r""" Pointer to wrapped data.""")
27801: (4)              s_num_instances = property(_mupdf.PdfLaunchUrlEvent_s_num_instances_get,
_mupdf.PdfLaunchUrlEvent_s_num_instances_set)
27802: (0)          _mupdf.PdfLaunchUrlEvent_swigregister(PdfLaunchUrlEvent)
27803: (0)          class PdfLayerConfig(object):
27804: (4)              r""" Wrapper class for struct `pdf_layer_config`."""
27805: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27806: (4)              __repr__ = _swig_repr
27807: (4)              def __init__(self, *args):
27808: (8)                  r"""
27809: (8)                  *Overload 1:*
27810: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
27811: (8)                  |
27812: (8)                  *Overload 2:*
27813: (8)                  Constructor using raw copy of pre-existing `::pdf_layer_config`.
27814: (8)                  |
27815: (8)                  *Overload 3:*
27816: (8)                  Constructor using raw copy of pre-existing `::pdf_layer_config`.
27817: (8)                  """
27818: (8)                  _mupdf.PdfLayerConfig_swiginit(self, _mupdf.new_PdfLayerConfig(*args))
27819: (4)              def internal(self, *args):
27820: (8)                  r"""
27821: (8)                  *Overload 1:*
27822: (8)                  Access as underlying struct.
27823: (8)                  |
27824: (8)                  *Overload 2:*
27825: (8)                  Access as underlying struct.
27826: (8)                  """
27827: (8)                  return _mupdf.PdfLayerConfig_internal(self, *args)
27828: (4)              __swig_destroy__ = _mupdf.delete_PdfLayerConfig
27829: (4)              name = property(_mupdf.PdfLayerConfig_name_get,
_mupdf.PdfLayerConfig_name_set)
27830: (4)              creator = property(_mupdf.PdfLayerConfig_creator_get,
_mupdf.PdfLayerConfig_creator_set)
27831: (4)              s_num_instances = property(_mupdf.PdfLayerConfig_s_num_instances_get,
_mupdf.PdfLayerConfig_s_num_instances_set)
27832: (4)              def to_string(self):
27833: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
27834: (8)                  return _mupdf.PdfLayerConfig_to_string(self)
27835: (4)              def __eq__(self, rhs):
27836: (8)                  r""" Comparison method."""
27837: (8)                  return _mupdf.PdfLayerConfig___eq__(self, rhs)
27838: (4)              def __ne__(self, rhs):
27839: (8)                  r""" Comparison method."""
27840: (8)                  return _mupdf.PdfLayerConfig___ne__(self, rhs)
27841: (0)          _mupdf.PdfLayerConfig_swigregister(PdfLayerConfig)
27842: (0)          class PdfLayerConfigUi(object):
27843: (4)              r""" Wrapper class for struct `pdf_layer_config_ui`."""
27844: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
27845: (4)              __repr__ = _swig_repr
27846: (4)              def __init__(self, *args):
27847: (8)                  r"""
27848: (8)                  *Overload 1:*
27849: (8)                  Default constructor sets .text to null, .type to PDF_LAYER_UI_LABEL,
and other fields to zero.
27850: (8)                  |
27851: (8)                  *Overload 2:*
27852: (8)                  We use default copy constructor and operator=.  Constructor using raw
copy of pre-existing `::pdf_layer_config_ui`.
27853: (8)                  |
27854: (8)                  *Overload 3:*
27855: (8)                  Constructor using raw copy of pre-existing `::pdf_layer_config_ui`.
27856: (8)                  """
27857: (8)                  _mupdf.PdfLayerConfigUi_swiginit(self,
_mupdf.new_PdfLayerConfigUi(*args))
27858: (4)              def internal(self, *args):
27859: (8)                  r"""
27860: (8)                  *Overload 1:*
27861: (8)                  Access as underlying struct.
27862: (8)                  |
27863: (8)                  *Overload 2:*
27864: (8)                  Access as underlying struct.
27865: (8)                  """
27866: (8)                  return _mupdf.PdfLayerConfigUi_internal(self, *args)
27867: (4)              __swig_destroy__ = _mupdf.delete_PdfLayerConfigUi
27868: (4)              text = property(_mupdf.PdfLayerConfigUi_text_get,
_mupdf.PdfLayerConfigUi_text_set)
27869: (4)              depth = property(_mupdf.PdfLayerConfigUi_depth_get,
_mupdf.PdfLayerConfigUi_depth_set)
27870: (4)              type = property(_mupdf.PdfLayerConfigUi_type_get,
_mupdf.PdfLayerConfigUi_type_set)
27871: (4)              selected = property(_mupdf.PdfLayerConfigUi_selected_get,
_mupdf.PdfLayerConfigUi_selected_set)
27872: (4)              locked = property(_mupdf.PdfLayerConfigUi_locked_get,
_mupdf.PdfLayerConfigUi_locked_set)
27873: (4)              s_num_instances = property(_mupdf.PdfLayerConfigUi_s_num_instances_get,
_mupdf.PdfLayerConfigUi_s_num_instances_set)
27874: (4)              def to_string(self):
27875: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
27876: (8)                  return _mupdf.PdfLayerConfigUi_to_string(self)
27877: (4)              def __eq__(self, rhs):
27878: (8)                  r""" Comparison method."""
27879: (8)                  return _mupdf.PdfLayerConfigUi___eq__(self, rhs)
27880: (4)              def __ne__(self, rhs):
27881: (8)                  r""" Comparison method."""
27882: (8)                  return _mupdf.PdfLayerConfigUi___ne__(self, rhs)
27883: (0)          _mupdf.PdfLayerConfigUi_swigregister(PdfLayerConfigUi)
27884: (0)          class PdfLexbuf(object):
27885: (4)              r""" Wrapper class for struct `pdf_lexbuf`. Not copyable or assignable."""
27886: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27887: (4)              __repr__ = _swig_repr
27888: (4)              def pdf_lexbuf_fin(self):
27889: (8)                  r""" Class-aware wrapper for `::pdf_lexbuf_fin()`."""
27890: (8)                  return _mupdf.PdfLexbuf_pdf_lexbuf_fin(self)
27891: (4)              def pdf_lexbuf_grow(self):
27892: (8)                  r""" Class-aware wrapper for `::pdf_lexbuf_grow()`."""
27893: (8)                  return _mupdf.PdfLexbuf_pdf_lexbuf_grow(self)
27894: (4)              def pdf_lexbuf_init(self, size):
27895: (8)                  r""" Class-aware wrapper for `::pdf_lexbuf_init()`."""
27896: (8)                  return _mupdf.PdfLexbuf_pdf_lexbuf_init(self, size)
27897: (4)              __swig_destroy__ = _mupdf.delete_PdfLexbuf
27898: (4)              def __init__(self, *args):
27899: (8)                  r"""
27900: (8)                  *Overload 1:*
27901: (8)                  Constructor that calls pdf_lexbuf_init(size).
27902: (8)                  |
```

```
27903: (8)                  *Overload 2:*
27904: (8)                  Default constructor, sets `m_internal` to null.
27905: (8)                  |
27906: (8)                  *Overload 3:*
27907: (8)                  Constructor using raw copy of pre-existing `::pdf_lexbuf`.
27908: (8)                  """
27909: (8)                  _mupdf.PdfLexbuf_swiginit(self, _mupdf.new_PdfLexbuf(*args))
27910: (4)              def m_internal_value(self):
27911: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27912: (8)                  return _mupdf.PdfLexbuf_m_internal_value(self)
27913: (4)              m_internal = property(_mupdf.PdfLexbuf_m_internal_get,
_mupdf.PdfLexbuf_m_internal_set, doc=r""" Pointer to wrapped data.""")
27914: (4)              s_num_instances = property(_mupdf.PdfLexbuf_s_num_instances_get,
_mupdf.PdfLexbuf_s_num_instances_set)
27915: (0)          _mupdf.PdfLexbuf_swigregister(PdfLexbuf)
27916: (0)          class PdfLexbufLarge(object):
27917: (4)              r""" Wrapper class for struct `pdf_lexbuf_large`. Not copyable or
assignable."""
27918: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27919: (4)              __repr__ = _swig_repr
27920: (4)              def __init__(self, *args):
27921: (8)                  r"""
27922: (8)                  *Overload 1:*
27923: (8)                  Default constructor, sets `m_internal` to null.
27924: (8)                  |
27925: (8)                  *Overload 2:*
27926: (8)                  Constructor using raw copy of pre-existing `::pdf_lexbuf_large`.
27927: (8)                  """
27928: (8)                  _mupdf.PdfLexbufLarge_swiginit(self, _mupdf.new_PdfLexbufLarge(*args))
27929: (4)              __swig_destroy__ = _mupdf.delete_PdfLexbufLarge
27930: (4)              def m_internal_value(self):
27931: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27932: (8)                  return _mupdf.PdfLexbufLarge_m_internal_value(self)
27933: (4)              m_internal = property(_mupdf.PdfLexbufLarge_m_internal_get,
_mupdf.PdfLexbufLarge_m_internal_set, doc=r""" Pointer to wrapped data.""")
27934: (4)              s_num_instances = property(_mupdf.PdfLexbufLarge_s_num_instances_get,
_mupdf.PdfLexbufLarge_s_num_instances_set)
27935: (0)          _mupdf.PdfLexbufLarge_swigregister(PdfLexbufLarge)
27936: (0)          class PdfLockedFields(object):
27937: (4)              r""" Wrapper class for struct `pdf_locked_fields`. Not copyable or
assignable."""
27938: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27939: (4)              __repr__ = _swig_repr
27940: (4)              def pdf_is_field_locked(self, name):
27941: (8)                  r""" Class-aware wrapper for `::pdf_is_field_locked()`."""
27942: (8)                  return _mupdf.PdfLockedFields_pdf_is_field_locked(self, name)
27943: (4)              def __init__(self, *args):
27944: (8)                  r"""
27945: (8)                  *Overload 1:*
27946: (8)                  Default constructor, sets `m_internal` to null.
27947: (8)                  |
27948: (8)                  *Overload 2:*
27949: (8)                  Constructor using raw copy of pre-existing `::pdf_locked_fields`.
27950: (8)                  """
27951: (8)                  _mupdf.PdfLockedFields_swiginit(self,
_mupdf.new_PdfLockedFields(*args))
27952: (4)              __swig_destroy__ = _mupdf.delete_PdfLockedFields
27953: (4)              def m_internal_value(self):
27954: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27955: (8)                  return _mupdf.PdfLockedFields_m_internal_value(self)
27956: (4)              m_internal = property(_mupdf.PdfLockedFields_m_internal_get,
_mupdf.PdfLockedFields_m_internal_set, doc=r""" Pointer to wrapped data.""")
27957: (4)              s_num_instances = property(_mupdf.PdfLockedFields_s_num_instances_get,
_mupdf.PdfLockedFields_s_num_instances_set)
```

```
27958: (0)          _mupdf.PdfLockedFields_swigregister(PdfLockedFields)
27959: (0)          class PdfMailDocEvent(object):
27960: (4)              r""" Wrapper class for struct `pdf_mail_doc_event`. Not copyable or
assignable."""
27961: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27962: (4)              __repr__ = _swig_repr
27963: (4)              def __init__(self, *args):
27964: (8)                  r"""
27965: (8)                  *Overload 1:*
27966: (8)                  Default constructor, sets `m_internal` to null.
27967: (8)                  |
27968: (8)                  *Overload 2:*
27969: (8)                  Constructor using raw copy of pre-existing `::pdf_mail_doc_event`.
27970: (8)                  """
27971: (8)                  _mupdf.PdfMailDocEvent_swiginit(self,
_mupdf.new_PdfMailDocEvent(*args))
27972: (4)              __swig_destroy__ = _mupdf.delete_PdfMailDocEvent
27973: (4)              def m_internal_value(self):
27974: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
27975: (8)                  return _mupdf.PdfMailDocEvent_m_internal_value(self)
27976: (4)              m_internal = property(_mupdf.PdfMailDocEvent_m_internal_get,
_mupdf.PdfMailDocEvent_m_internal_set, doc=r""" Pointer to wrapped data.""")
27977: (4)              s_num_instances = property(_mupdf.PdfMailDocEvent_s_num_instances_get,
_mupdf.PdfMailDocEvent_s_num_instances_set)
27978: (0)          _mupdf.PdfMailDocEvent_swigregister(PdfMailDocEvent)
27979: (0)          class PdfMarkBits(object):
27980: (4)              r""" Wrapper class for struct `pdf_mark_bits`. Not copyable or
assignable."""
27981: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
27982: (4)              __repr__ = _swig_repr
27983: (4)              def pdf_mark_bits_reset(self):
27984: (8)                  r""" Class-aware wrapper for `::pdf_mark_bits_reset()`."""
27985: (8)                  return _mupdf.PdfMarkBits_pdf_mark_bits_reset(self)
27986: (4)              def pdf_mark_bits_set(self, obj):
27987: (8)                  r""" Class-aware wrapper for `::pdf_mark_bits_set()`."""
27988: (8)                  return _mupdf.PdfMarkBits_pdf_mark_bits_set(self, obj)
27989: (4)              def __init__(self, *args):
27990: (8)                  r"""
27991: (8)                  *Overload 1:*
27992: (8)                  == Constructors.  Constructor using `pdf_new_mark_bits()`.
27993: (8)                  |
27994: (8)                  *Overload 2:*
27995: (8)                  Default constructor, sets `m_internal` to null.
27996: (8)                  |
27997: (8)                  *Overload 3:*
27998: (8)                  Constructor using raw copy of pre-existing `::pdf_mark_bits`.
27999: (8)                  """
28000: (8)                  _mupdf.PdfMarkBits_swiginit(self, _mupdf.new_PdfMarkBits(*args))
28001: (4)              __swig_destroy__ = _mupdf.delete_PdfMarkBits
28002: (4)              def m_internal_value(self):
28003: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28004: (8)                  return _mupdf.PdfMarkBits_m_internal_value(self)
28005: (4)              m_internal = property(_mupdf.PdfMarkBits_m_internal_get,
_mupdf.PdfMarkBits_m_internal_set, doc=r""" Pointer to wrapped data.""")
28006: (4)              s_num_instances = property(_mupdf.PdfMarkBits_s_num_instances_get,
_mupdf.PdfMarkBits_s_num_instances_set)
28007: (0)          _mupdf.PdfMarkBits_swigregister(PdfMarkBits)
28008: (0)          class PdfMarkList(object):
28009: (4)              r""" Wrapper class for struct `pdf_mark_list`. Not copyable or
assignable."""
28010: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28011: (4)              __repr__ = _swig_repr
28012: (4)              def pdf_mark_list_check(self, obj):
28013: (8)                  r""" Class-aware wrapper for `::pdf_mark_list_check()`."""
```

```
28014: (8)                 return _mupdf.PdfMarkList_pdf_mark_list_check(self, obj)
28015: (4)             def pdf_mark_list_free(self):
28016: (8)                 r""" Class-aware wrapper for `::pdf_mark_list_free()`."""
28017: (8)                 return _mupdf.PdfMarkList_pdf_mark_list_free(self)
28018: (4)             def pdf_mark_list_init(self):
28019: (8)                 r""" Class-aware wrapper for `::pdf_mark_list_init()`."""
28020: (8)                 return _mupdf.PdfMarkList_pdf_mark_list_init(self)
28021: (4)             def pdf_mark_list_pop(self):
28022: (8)                 r""" Class-aware wrapper for `::pdf_mark_list_pop()`."""
28023: (8)                 return _mupdf.PdfMarkList_pdf_mark_list_pop(self)
28024: (4)             def pdf_mark_list_push(self, obj):
28025: (8)                 r""" Class-aware wrapper for `::pdf_mark_list_push()`."""
28026: (8)                 return _mupdf.PdfMarkList_pdf_mark_list_push(self, obj)
28027: (4)             def __init__(self, *args):
28028: (8)                 r"""
28029: (8)                 *Overload 1:*
28030: (8)                 Default constructor, sets `m_internal` to null.
28031: (8)                 |
28032: (8)                 *Overload 2:*
28033: (8)                 Constructor using raw copy of pre-existing `::pdf_mark_list`.
28034: (8)                 """
28035: (8)                 _mupdf.PdfMarkList_swiginit(self, _mupdf.new_PdfMarkList(*args))
28036: (4)             __swig_destroy__ = _mupdf.delete_PdfMarkList
28037: (4)             def m_internal_value(self):
28038: (8)                 r""" Return numerical value of .m_internal; helps with Python
debugging."""
28039: (8)                 return _mupdf.PdfMarkList_m_internal_value(self)
28040: (4)             m_internal = property(_mupdf.PdfMarkList_m_internal_get,
_mupdf.PdfMarkList_m_internal_set, doc=r""" Pointer to wrapped data.""")
28041: (4)             s_num_instances = property(_mupdf.PdfMarkList_s_num_instances_get,
_mupdf.PdfMarkList_s_num_instances_set)
28042: (0)         _mupdf.PdfMarkList_swigregister(PdfMarkList)
28043: (0)         class PdfMrange(object):
28044: (4)             r""" Wrapper class for struct `pdf_mrange`. Not copyable or assignable."""
28045: (4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28046: (4)             __repr__ = _swig_repr
28047: (4)             def __init__(self, *args):
28048: (8)                 r"""
28049: (8)                 *Overload 1:*
28050: (8)                 Default constructor, sets `m_internal` to null.
28051: (8)                 |
28052: (8)                 *Overload 2:*
28053: (8)                 Constructor using raw copy of pre-existing `::pdf_mrange`.
28054: (8)                 """
28055: (8)                 _mupdf.PdfMrange_swiginit(self, _mupdf.new_PdfMrange(*args))
28056: (4)             __swig_destroy__ = _mupdf.delete_PdfMrange
28057: (4)             def m_internal_value(self):
28058: (8)                 r""" Return numerical value of .m_internal; helps with Python
debugging."""
28059: (8)                 return _mupdf.PdfMrange_m_internal_value(self)
28060: (4)             m_internal = property(_mupdf.PdfMrange_m_internal_get,
_mupdf.PdfMrange_m_internal_set, doc=r""" Pointer to wrapped data.""")
28061: (4)             s_num_instances = property(_mupdf.PdfMrange_s_num_instances_get,
_mupdf.PdfMrange_s_num_instances_set)
28062: (0)         _mupdf.PdfMrange_swigregister(PdfMrange)
28063: (0)         class PdfObj(object):
28064: (4)             r""" Wrapper class for struct `pdf_obj`."""
28065: (4)             thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28066: (4)             __repr__ = _swig_repr
28067: (4)             @staticmethod
28068: (4)             def pdf_new_text_string(s):
28069: (8)                 r""" Class-aware wrapper for `::pdf_new_text_string()`."""
28070: (8)                 return _mupdf.PdfObj_pdf_new_text_string(s)
28071: (4)             @staticmethod
28072: (4)             def pdf_new_dict(doc, initialcap):
28073: (8)                 r""" Class-aware wrapper for `::pdf_new_dict()`."""
28074: (8)                 return _mupdf.PdfObj_pdf_new_dict(doc, initialcap)
```

```
(4)     def pdf_array_contains(self, obj):
(8)         r""" Class-aware wrapper for `::pdf_array_contains()`."""
(8)         return _mupdf.PdfObj_pdf_array_contains(self, obj)
(4)     def pdf_array_delete(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_delete()`."""
(8)         return _mupdf.PdfObj_pdf_array_delete(self, index)
(4)     def pdf_array_find(self, obj):
(8)         r""" Class-aware wrapper for `::pdf_array_find()`."""
(8)         return _mupdf.PdfObj_pdf_array_find(self, obj)
(4)     def pdf_array_get(self, i):
(8)         r""" Class-aware wrapper for `::pdf_array_get()`."""
(8)         return _mupdf.PdfObj_pdf_array_get(self, i)
(4)     def pdf_array_get_bool(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_bool()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_bool(self, index)
(4)     def pdf_array_get_int(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_int()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_int(self, index)
(4)     def pdf_array_get_matrix(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_matrix()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_matrix(self, index)
(4)     def pdf_array_get_name(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_name()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_name(self, index)
(4)     def pdf_array_get_real(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_real()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_real(self, index)
(4)     def pdf_array_get_rect(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_rect()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_rect(self, index)
(4)     def pdf_array_get_string(self, index, sizep):
(8)         r"""
(8)         Class-aware wrapper for `::pdf_array_get_string()`.
(8)         This method has out-params. Python/C# wrappers look like:
(12)            `pdf_array_get_string(int index)` => `(const char *, size_t
sizep)`
(8)         """
(8)         return _mupdf.PdfObj_pdf_array_get_string(self, index, sizep)
(4)     def pdf_array_get_text_string(self, index):
(8)         r""" Class-aware wrapper for `::pdf_array_get_text_string()`."""
(8)         return _mupdf.PdfObj_pdf_array_get_text_string(self, index)
(4)     def pdf_array_insert(self, obj, index):
(8)         r""" Class-aware wrapper for `::pdf_array_insert()`."""
(8)         return _mupdf.PdfObj_pdf_array_insert(self, obj, index)
(4)     def pdf_array_len(self):
(8)         r""" Class-aware wrapper for `::pdf_array_len()`."""
(8)         return _mupdf.PdfObj_pdf_array_len(self)
(4)     def pdf_array_push(self, obj):
(8)         r""" Class-aware wrapper for `::pdf_array_push()`."""
(8)         return _mupdf.PdfObj_pdf_array_push(self, obj)
(4)     def pdf_array_push_array(self, initial):
(8)         r""" Class-aware wrapper for `::pdf_array_push_array()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_array(self, initial)
(4)     def pdf_array_push_bool(self, x):
(8)         r""" Class-aware wrapper for `::pdf_array_push_bool()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_bool(self, x)
(4)     def pdf_array_push_dict(self, initial):
(8)         r""" Class-aware wrapper for `::pdf_array_push_dict()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_dict(self, initial)
(4)     def pdf_array_push_int(self, x):
(8)         r""" Class-aware wrapper for `::pdf_array_push_int()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_int(self, x)
(4)     def pdf_array_push_name(self, x):
(8)         r""" Class-aware wrapper for `::pdf_array_push_name()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_name(self, x)
(4)     def pdf_array_push_real(self, x):
(8)         r""" Class-aware wrapper for `::pdf_array_push_real()`."""
(8)         return _mupdf.PdfObj_pdf_array_push_real(self, x)
(4)     def pdf_array_push_string(self, x, n):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_array_push_string()`."""
(8)                 return _mupdf.PdfObj_pdf_array_push_string(self, x, n)
(4)             def pdf_array_push_text_string(self, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_push_text_string()`."""
(8)                 return _mupdf.PdfObj_pdf_array_push_text_string(self, x)
(4)             def pdf_array_put(self, i, obj):
(8)                 r""" Class-aware wrapper for `::pdf_array_put()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put(self, i, obj)
(4)             def pdf_array_put_array(self, i, initial):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_array()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_array(self, i, initial)
(4)             def pdf_array_put_bool(self, i, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_bool()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_bool(self, i, x)
(4)             def pdf_array_put_dict(self, i, initial):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_dict()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_dict(self, i, initial)
(4)             def pdf_array_put_int(self, i, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_int()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_int(self, i, x)
(4)             def pdf_array_put_name(self, i, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_name()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_name(self, i, x)
(4)             def pdf_array_put_real(self, i, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_real()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_real(self, i, x)
(4)             def pdf_array_put_string(self, i, x, n):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_string()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_string(self, i, x, n)
(4)             def pdf_array_put_text_string(self, i, x):
(8)                 r""" Class-aware wrapper for `::pdf_array_put_text_string()`."""
(8)                 return _mupdf.PdfObj_pdf_array_put_text_string(self, i, x)
(4)             def pdf_button_field_on_state(self):
(8)                 r""" Class-aware wrapper for `::pdf_button_field_on_state()`."""
(8)                 return _mupdf.PdfObj_pdf_button_field_on_state(self)
(4)             def pdf_choice_field_option(self, exportval, i):
(8)                 r""" Class-aware wrapper for `::pdf_choice_field_option()`."""
(8)                 return _mupdf.PdfObj_pdf_choice_field_option(self, exportval, i)
(4)             def pdf_choice_field_option_count(self):
(8)                 r""" Class-aware wrapper for `::pdf_choice_field_option_count()`."""
(8)                 return _mupdf.PdfObj_pdf_choice_field_option_count(self)
(4)             def pdf_clean_obj(self):
(8)                 r""" Class-aware wrapper for `::pdf_clean_obj()`."""
(8)                 return _mupdf.PdfObj_pdf_clean_obj(self)
(4)             def pdf_copy_array(self):
(8)                 r""" Class-aware wrapper for `::pdf_copy_array()`."""
(8)                 return _mupdf.PdfObj_pdf_copy_array(self)
(4)             def pdf_copy_dict(self):
(8)                 r""" Class-aware wrapper for `::pdf_copy_dict()`."""
(8)                 return _mupdf.PdfObj_pdf_copy_dict(self)
(4)             def pdf_debug_obj(self):
(8)                 r""" Class-aware wrapper for `::pdf_debug_obj()`."""
(8)                 return _mupdf.PdfObj_pdf_debug_obj(self)
(4)             def pdf_debug_ref(self):
(8)                 r""" Class-aware wrapper for `::pdf_debug_ref()`."""
(8)                 return _mupdf.PdfObj_pdf_debug_ref(self)
(4)             def pdf_deep_copy_obj(self):
(8)                 r""" Class-aware wrapper for `::pdf_deep_copy_obj()`."""
(8)                 return _mupdf.PdfObj_pdf_deep_copy_obj(self)
(4)             def pdf_dict_del(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_del()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_del(self, key)
(4)             def pdf_dict_dels(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_dels()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_dels(self, key)
(4)             def pdf_dict_get_bool(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_get_bool()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_get_bool(self, key)
(4)             def pdf_dict_get_bool_default(self, key, _def):
```

```
28212: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_bool_default()`."""
28213: (8)                  return _mupdf.PdfObj_pdf_dict_get_bool_default(self, key, _def)
28214: (4)              def pdf_dict_get_date(self, key):
28215: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_date()`."""
28216: (8)                  return _mupdf.PdfObj_pdf_dict_get_date(self, key)
28217: (4)              def pdf_dict_get_inheritable(self, key):
28218: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable()`."""
28219: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable(self, key)
28220: (4)              def pdf_dict_get_inheritable_bool(self, key):
28221: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_bool()`."""
28222: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_bool(self, key)
28223: (4)              def pdf_dict_get_inheritable_date(self, key):
28224: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_date()`."""
28225: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_date(self, key)
28226: (4)              def pdf_dict_get_inheritable_int(self, key):
28227: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_int()`."""
28228: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_int(self, key)
28229: (4)              def pdf_dict_get_inheritable_int64(self, key):
28230: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_int64()`."""
28231: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_int64(self, key)
28232: (4)              def pdf_dict_get_inheritable_matrix(self, key):
28233: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_matrix()`."""
28234: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_matrix(self, key)
28235: (4)              def pdf_dict_get_inheritable_name(self, key):
28236: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_name()`."""
28237: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_name(self, key)
28238: (4)              def pdf_dict_get_inheritable_real(self, key):
28239: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_real()`."""
28240: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_real(self, key)
28241: (4)              def pdf_dict_get_inheritable_rect(self, key):
28242: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_inheritable_rect()`."""
28243: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_rect(self, key)
28244: (4)              def pdf_dict_get_inheritable_string(self, key, sizep):
28245: (8)                  r"""
28246: (8)                  Class-aware wrapper for `::pdf_dict_get_inheritable_string()`.
28247: (8)                  This method has out-params. Python/C# wrappers look like:
28248: (12)                     `pdf_dict_get_inheritable_string(::pdf_obj *key)` => `(const char
*, size_t sizep)`
28249: (8)                  """
28250: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_string(self, key, sizep)
28251: (4)              def pdf_dict_get_inheritable_text_string(self, key):
28252: (8)                  r""" Class-aware wrapper for
`::pdf_dict_get_inheritable_text_string()`."""
28253: (8)                  return _mupdf.PdfObj_pdf_dict_get_inheritable_text_string(self, key)
28254: (4)              def pdf_dict_get_int(self, key):
28255: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_int()`."""
28256: (8)                  return _mupdf.PdfObj_pdf_dict_get_int(self, key)
28257: (4)              def pdf_dict_get_int64(self, key):
28258: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_int64()`."""
28259: (8)                  return _mupdf.PdfObj_pdf_dict_get_int64(self, key)
28260: (4)              def pdf_dict_get_int_default(self, key, _def):
28261: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_int_default()`."""
28262: (8)                  return _mupdf.PdfObj_pdf_dict_get_int_default(self, key, _def)
28263: (4)              def pdf_dict_get_key(self, idx):
28264: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_key()`."""
28265: (8)                  return _mupdf.PdfObj_pdf_dict_get_key(self, idx)
28266: (4)              def pdf_dict_get_matrix(self, key):
28267: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_matrix()`."""
28268: (8)                  return _mupdf.PdfObj_pdf_dict_get_matrix(self, key)
28269: (4)              def pdf_dict_get_name(self, key):
28270: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_name()`."""
28271: (8)                  return _mupdf.PdfObj_pdf_dict_get_name(self, key)
28272: (4)              def pdf_dict_get_real(self, key):
28273: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_real()`."""
28274: (8)                  return _mupdf.PdfObj_pdf_dict_get_real(self, key)
28275: (4)              def pdf_dict_get_real_default(self, key, _def):
28276: (8)                  r""" Class-aware wrapper for `::pdf_dict_get_real_default()`."""
28277: (8)                  return _mupdf.PdfObj_pdf_dict_get_real_default(self, key, _def)
28278: (4)              def pdf_dict_get_rect(self, key):
```

```
(8)                 r""" Class-aware wrapper for `::pdf_dict_get_rect()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_get_rect(self, key)
(4)             def pdf_dict_get_string(self, key, sizep):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_dict_get_string()`.
(8)                 This method has out-params. Python/C# wrappers look like:
(12)                    `pdf_dict_get_string(::pdf_obj *key)` => `(const char *, size_t
sizep)`
(8)                 """
(8)                 return _mupdf.PdfObj_pdf_dict_get_string(self, key, sizep)
(4)             def pdf_dict_get_text_string(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_get_text_string()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_get_text_string(self, key)
(4)             def pdf_dict_get_val(self, idx):
(8)                 r""" Class-aware wrapper for `::pdf_dict_get_val()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_get_val(self, idx)
(4)             def pdf_dict_geta(self, key, abbrev):
(8)                 r""" Class-aware wrapper for `::pdf_dict_geta()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_geta(self, key, abbrev)
(4)             def pdf_dict_getp(self, path):
(8)                 r""" Class-aware wrapper for `::pdf_dict_getp()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_getp(self, path)
(4)             def pdf_dict_getp_inheritable(self, path):
(8)                 r""" Class-aware wrapper for `::pdf_dict_getp_inheritable()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_getp_inheritable(self, path)
(4)             def pdf_dict_gets(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_gets()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_gets(self, key)
(4)             def pdf_dict_gets_inheritable(self, key):
(8)                 r""" Class-aware wrapper for `::pdf_dict_gets_inheritable()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_gets_inheritable(self, key)
(4)             def pdf_dict_getsa(self, key, abbrev):
(8)                 r""" Class-aware wrapper for `::pdf_dict_getsa()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_getsa(self, key, abbrev)
(4)             def pdf_dict_len(self):
(8)                 r""" Class-aware wrapper for `::pdf_dict_len()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_len(self)
(4)             def pdf_dict_put(self, key, val):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put(self, key, val)
(4)             def pdf_dict_put_array(self, key, initial):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_array()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_array(self, key, initial)
(4)             def pdf_dict_put_bool(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_bool()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_bool(self, key, x)
(4)             def pdf_dict_put_date(self, key, time):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_date()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_date(self, key, time)
(4)             def pdf_dict_put_dict(self, key, initial):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_dict()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_dict(self, key, initial)
(4)             def pdf_dict_put_int(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_int()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_int(self, key, x)
(4)             def pdf_dict_put_matrix(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_matrix()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_matrix(self, key, x)
(4)             def pdf_dict_put_name(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_name()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_name(self, key, x)
(4)             def pdf_dict_put_real(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_real()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_real(self, key, x)
(4)             def pdf_dict_put_rect(self, key, x):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_rect()`."""
(8)                 return _mupdf.PdfObj_pdf_dict_put_rect(self, key, x)
(4)             def pdf_dict_put_string(self, key, x, n):
(8)                 r""" Class-aware wrapper for `::pdf_dict_put_string()`."""
```

```
28347: (8)                 return _mupdf.PdfObj_pdf_dict_put_string(self, key, x, n)
28348: (4)             def pdf_dict_put_text_string(self, key, x):
28349: (8)                 r""" Class-aware wrapper for `::pdf_dict_put_text_string()`."""
28350: (8)                 return _mupdf.PdfObj_pdf_dict_put_text_string(self, key, x)
28351: (4)             def pdf_dict_put_val_null(self, idx):
28352: (8)                 r""" Class-aware wrapper for `::pdf_dict_put_val_null()`."""
28353: (8)                 return _mupdf.PdfObj_pdf_dict_put_val_null(self, idx)
28354: (4)             def pdf_dict_putp(self, path, val):
28355: (8)                 r""" Class-aware wrapper for `::pdf_dict_putp()`."""
28356: (8)                 return _mupdf.PdfObj_pdf_dict_putp(self, path, val)
28357: (4)             def pdf_dict_puts(self, key, val):
28358: (8)                 r""" Class-aware wrapper for `::pdf_dict_puts()`."""
28359: (8)                 return _mupdf.PdfObj_pdf_dict_puts(self, key, val)
28360: (4)             def pdf_dict_puts_dict(self, key, initial):
28361: (8)                 r""" Class-aware wrapper for `::pdf_dict_puts_dict()`."""
28362: (8)                 return _mupdf.PdfObj_pdf_dict_puts_dict(self, key, initial)
28363: (4)             def pdf_dirty_obj(self):
28364: (8)                 r""" Class-aware wrapper for `::pdf_dirty_obj()`."""
28365: (8)                 return _mupdf.PdfObj_pdf_dirty_obj(self)
28366: (4)             def pdf_field_border_style(self):
28367: (8)                 r""" Class-aware wrapper for `::pdf_field_border_style()`."""
28368: (8)                 return _mupdf.PdfObj_pdf_field_border_style(self)
28369: (4)             def pdf_field_display(self):
28370: (8)                 r""" Class-aware wrapper for `::pdf_field_display()`."""
28371: (8)                 return _mupdf.PdfObj_pdf_field_display(self)
28372: (4)             def pdf_field_flags(self):
28373: (8)                 r""" Class-aware wrapper for `::pdf_field_flags()`."""
28374: (8)                 return _mupdf.PdfObj_pdf_field_flags(self)
28375: (4)             def pdf_field_label(self):
28376: (8)                 r""" Class-aware wrapper for `::pdf_field_label()`."""
28377: (8)                 return _mupdf.PdfObj_pdf_field_label(self)
28378: (4)             def pdf_field_set_border_style(self, text):
28379: (8)                 r""" Class-aware wrapper for `::pdf_field_set_border_style()`."""
28380: (8)                 return _mupdf.PdfObj_pdf_field_set_border_style(self, text)
28381: (4)             def pdf_field_set_button_caption(self, text):
28382: (8)                 r""" Class-aware wrapper for `::pdf_field_set_button_caption()`."""
28383: (8)                 return _mupdf.PdfObj_pdf_field_set_button_caption(self, text)
28384: (4)             def pdf_field_set_display(self, d):
28385: (8)                 r""" Class-aware wrapper for `::pdf_field_set_display()`."""
28386: (8)                 return _mupdf.PdfObj_pdf_field_set_display(self, d)
28387: (4)             def pdf_field_set_fill_color(self, col):
28388: (8)                 r""" Class-aware wrapper for `::pdf_field_set_fill_color()`."""
28389: (8)                 return _mupdf.PdfObj_pdf_field_set_fill_color(self, col)
28390: (4)             def pdf_field_set_text_color(self, col):
28391: (8)                 r""" Class-aware wrapper for `::pdf_field_set_text_color()`."""
28392: (8)                 return _mupdf.PdfObj_pdf_field_set_text_color(self, col)
28393: (4)             def pdf_field_type(self):
28394: (8)                 r""" Class-aware wrapper for `::pdf_field_type()`."""
28395: (8)                 return _mupdf.PdfObj_pdf_field_type(self)
28396: (4)             def pdf_field_type_string(self):
28397: (8)                 r""" Class-aware wrapper for `::pdf_field_type_string()`."""
28398: (8)                 return _mupdf.PdfObj_pdf_field_type_string(self)
28399: (4)             def pdf_field_value(self):
28400: (8)                 r""" Class-aware wrapper for `::pdf_field_value()`."""
28401: (8)                 return _mupdf.PdfObj_pdf_field_value(self)
28402: (4)             def pdf_filter_xobject_instance(self, page_res, ctm, options, cycle_up):
28403: (8)                 r""" Class-aware wrapper for `::pdf_filter_xobject_instance()`."""
28404: (8)                 return _mupdf.PdfObj_pdf_filter_xobject_instance(self, page_res, ctm,
options, cycle_up)
28405: (4)             def pdf_flatten_inheritable_page_items(self):
28406: (8)                 r""" Class-aware wrapper for
`::pdf_flatten_inheritable_page_items()`."""
28407: (8)                 return _mupdf.PdfObj_pdf_flatten_inheritable_page_items(self)
28408: (4)             def pdf_get_bound_document(self):
28409: (8)                 r""" Class-aware wrapper for `::pdf_get_bound_document()`."""
28410: (8)                 return _mupdf.PdfObj_pdf_get_bound_document(self)
28411: (4)             def pdf_get_embedded_file_params(self, out):
28412: (8)                 r""" Class-aware wrapper for `::pdf_get_embedded_file_params()`."""
28413: (8)                 return _mupdf.PdfObj_pdf_get_embedded_file_params(self, out)
```

```
28414: (4)                def pdf_get_indirect_document(self):
28415: (8)                    r""" Class-aware wrapper for `::pdf_get_indirect_document()`."""
28416: (8)                    return _mupdf.PdfObj_pdf_get_indirect_document(self)
28417: (4)                def pdf_intent_from_name(self):
28418: (8)                    r""" Class-aware wrapper for `::pdf_intent_from_name()`."""
28419: (8)                    return _mupdf.PdfObj_pdf_intent_from_name(self)
28420: (4)                def pdf_is_array(self):
28421: (8)                    r""" Class-aware wrapper for `::pdf_is_array()`."""
28422: (8)                    return _mupdf.PdfObj_pdf_is_array(self)
28423: (4)                def pdf_is_bool(self):
28424: (8)                    r""" Class-aware wrapper for `::pdf_is_bool()`."""
28425: (8)                    return _mupdf.PdfObj_pdf_is_bool(self)
28426: (4)                def pdf_is_dict(self):
28427: (8)                    r""" Class-aware wrapper for `::pdf_is_dict()`."""
28428: (8)                    return _mupdf.PdfObj_pdf_is_dict(self)
28429: (4)                def pdf_is_embedded_file(self):
28430: (8)                    r""" Class-aware wrapper for `::pdf_is_embedded_file()`."""
28431: (8)                    return _mupdf.PdfObj_pdf_is_embedded_file(self)
28432: (4)                def pdf_is_indirect(self):
28433: (8)                    r""" Class-aware wrapper for `::pdf_is_indirect()`."""
28434: (8)                    return _mupdf.PdfObj_pdf_is_indirect(self)
28435: (4)                def pdf_is_int(self):
28436: (8)                    r""" Class-aware wrapper for `::pdf_is_int()`."""
28437: (8)                    return _mupdf.PdfObj_pdf_is_int(self)
28438: (4)                def pdf_is_jpx_image(self):
28439: (8)                    r""" Class-aware wrapper for `::pdf_is_jpx_image()`."""
28440: (8)                    return _mupdf.PdfObj_pdf_is_jpx_image(self)
28441: (4)                def pdf_is_name(self):
28442: (8)                    r""" Class-aware wrapper for `::pdf_is_name()`."""
28443: (8)                    return _mupdf.PdfObj_pdf_is_name(self)
28444: (4)                def pdf_is_null(self):
28445: (8)                    r""" Class-aware wrapper for `::pdf_is_null()`."""
28446: (8)                    return _mupdf.PdfObj_pdf_is_null(self)
28447: (4)                def pdf_is_number(self):
28448: (8)                    r""" Class-aware wrapper for `::pdf_is_number()`."""
28449: (8)                    return _mupdf.PdfObj_pdf_is_number(self)
28450: (4)                def pdf_is_real(self):
28451: (8)                    r""" Class-aware wrapper for `::pdf_is_real()`."""
28452: (8)                    return _mupdf.PdfObj_pdf_is_real(self)
28453: (4)                def pdf_is_stream(self):
28454: (8)                    r""" Class-aware wrapper for `::pdf_is_stream()`."""
28455: (8)                    return _mupdf.PdfObj_pdf_is_stream(self)
28456: (4)                def pdf_is_string(self):
28457: (8)                    r""" Class-aware wrapper for `::pdf_is_string()`."""
28458: (8)                    return _mupdf.PdfObj_pdf_is_string(self)
28459: (4)                def pdf_line_ending_from_name(self):
28460: (8)                    r""" Class-aware wrapper for `::pdf_line_ending_from_name()`."""
28461: (8)                    return _mupdf.PdfObj_pdf_line_ending_from_name(self)
28462: (4)                def pdf_load_colorspace(self):
28463: (8)                    r""" Class-aware wrapper for `::pdf_load_colorspace()`."""
28464: (8)                    return _mupdf.PdfObj_pdf_load_colorspace(self)
28465: (4)                def pdf_load_embedded_file_contents(self):
28466: (8)                    r""" Class-aware wrapper for `::pdf_load_embedded_file_contents()`."""
28467: (8)                    return _mupdf.PdfObj_pdf_load_embedded_file_contents(self)
28468: (4)                def pdf_load_field_name(self):
28469: (8)                    r""" Class-aware wrapper for `::pdf_load_field_name()`."""
28470: (8)                    return _mupdf.PdfObj_pdf_load_field_name(self)
28471: (4)                def pdf_load_function(self, _in, out):
28472: (8)                    r""" Class-aware wrapper for `::pdf_load_function()`."""
28473: (8)                    return _mupdf.PdfObj_pdf_load_function(self, _in, out)
28474: (4)                def pdf_load_raw_stream(self):
28475: (8)                    r""" Class-aware wrapper for `::pdf_load_raw_stream()`."""
28476: (8)                    return _mupdf.PdfObj_pdf_load_raw_stream(self)
28477: (4)                def pdf_load_stream(self):
28478: (8)                    r""" Class-aware wrapper for `::pdf_load_stream()`."""
28479: (8)                    return _mupdf.PdfObj_pdf_load_stream(self)
28480: (4)                def pdf_load_stream_or_string_as_utf8(self):
28481: (8)                    r""" Class-aware wrapper for
`::pdf_load_stream_or_string_as_utf8()`."""
```

```
28482: (8)                  return _mupdf.PdfObj_pdf_load_stream_or_string_as_utf8(self)
28483: (4)              def pdf_lookup_field(self, name):
28484: (8)                  r""" Class-aware wrapper for `::pdf_lookup_field()`."""
28485: (8)                  return _mupdf.PdfObj_pdf_lookup_field(self, name)
28486: (4)              def pdf_lookup_number(self, needle):
28487: (8)                  r""" Class-aware wrapper for `::pdf_lookup_number()`."""
28488: (8)                  return _mupdf.PdfObj_pdf_lookup_number(self, needle)
28489: (4)              def pdf_mark_obj(self):
28490: (8)                  r""" Class-aware wrapper for `::pdf_mark_obj()`."""
28491: (8)                  return _mupdf.PdfObj_pdf_mark_obj(self)
28492: (4)              def pdf_name_eq(self, b):
28493: (8)                  r""" Class-aware wrapper for `::pdf_name_eq()`."""
28494: (8)                  return _mupdf.PdfObj_pdf_name_eq(self, b)
28495: (4)              def pdf_new_utf8_from_pdf_stream_obj(self):
28496: (8)                  r""" Class-aware wrapper for
`::pdf_new_utf8_from_pdf_stream_obj()`."""
28497: (8)                  return _mupdf.PdfObj_pdf_new_utf8_from_pdf_stream_obj(self)
28498: (4)              def pdf_new_utf8_from_pdf_string_obj(self):
28499: (8)                  r""" Class-aware wrapper for
`::pdf_new_utf8_from_pdf_string_obj()`."""
28500: (8)                  return _mupdf.PdfObj_pdf_new_utf8_from_pdf_string_obj(self)
28501: (4)              def pdf_obj_is_dirty(self):
28502: (8)                  r""" Class-aware wrapper for `::pdf_obj_is_dirty()`."""
28503: (8)                  return _mupdf.PdfObj_pdf_obj_is_dirty(self)
28504: (4)              def pdf_obj_is_incremental(self):
28505: (8)                  r""" Class-aware wrapper for `::pdf_obj_is_incremental()`."""
28506: (8)                  return _mupdf.PdfObj_pdf_obj_is_incremental(self)
28507: (4)              def pdf_obj_marked(self):
28508: (8)                  r""" Class-aware wrapper for `::pdf_obj_marked()`."""
28509: (8)                  return _mupdf.PdfObj_pdf_obj_marked(self)
28510: (4)              def pdf_obj_memo(self, bit, memo):
28511: (8)                  r"""
28512: (8)                  Class-aware wrapper for `::pdf_obj_memo()`.
28513: (8)                  This method has out-params. Python/C# wrappers look like:
28514: (12)                     `pdf_obj_memo(int bit)` => `(int, int memo)`
28515: (8)                  """
28516: (8)                  return _mupdf.PdfObj_pdf_obj_memo(self, bit, memo)
28517: (4)              def pdf_obj_parent_num(self):
28518: (8)                  r""" Class-aware wrapper for `::pdf_obj_parent_num()`."""
28519: (8)                  return _mupdf.PdfObj_pdf_obj_parent_num(self)
28520: (4)              def pdf_obj_refs(self):
28521: (8)                  r""" Class-aware wrapper for `::pdf_obj_refs()`."""
28522: (8)                  return _mupdf.PdfObj_pdf_obj_refs(self)
28523: (4)              def pdf_objcmp(self, b):
28524: (8)                  r""" Class-aware wrapper for `::pdf_objcmp()`."""
28525: (8)                  return _mupdf.PdfObj_pdf_objcmp(self, b)
28526: (4)              def pdf_objcmp_deep(self, b):
28527: (8)                  r""" Class-aware wrapper for `::pdf_objcmp_deep()`."""
28528: (8)                  return _mupdf.PdfObj_pdf_objcmp_deep(self, b)
28529: (4)              def pdf_objcmp_resolve(self, b):
28530: (8)                  r""" Class-aware wrapper for `::pdf_objcmp_resolve()`."""
28531: (8)                  return _mupdf.PdfObj_pdf_objcmp_resolve(self, b)
28532: (4)              def pdf_open_raw_stream(self):
28533: (8)                  r""" Class-aware wrapper for `::pdf_open_raw_stream()`."""
28534: (8)                  return _mupdf.PdfObj_pdf_open_raw_stream(self)
28535: (4)              def pdf_open_stream(self):
28536: (8)                  r""" Class-aware wrapper for `::pdf_open_stream()`."""
28537: (8)                  return _mupdf.PdfObj_pdf_open_stream(self)
28538: (4)              def pdf_page_obj_transform(self, outbox, outctm):
28539: (8)                  r""" Class-aware wrapper for `::pdf_page_obj_transform()`."""
28540: (8)                  return _mupdf.PdfObj_pdf_page_obj_transform(self, outbox, outctm)
28541: (4)              def pdf_page_obj_transform_box(self, outbox, out, box):
28542: (8)                  r""" Class-aware wrapper for `::pdf_page_obj_transform_box()`."""
28543: (8)                  return _mupdf.PdfObj_pdf_page_obj_transform_box(self, outbox, out,
box)
28544: (4)              def pdf_pin_document(self):
28545: (8)                  r""" Class-aware wrapper for `::pdf_pin_document()`."""
28546: (8)                  return _mupdf.PdfObj_pdf_pin_document(self)
28547: (4)              def pdf_recolor_shade(self, reshade, opaque):
```

```
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_recolor_shade()`.
(12)                    Recolor a shade.
(8)                 """
(8)                 return _mupdf.PdfObj_pdf_recolor_shade(self, reshade, opaque)
(4)             def pdf_resolve_indirect(self):
(8)                 r"""
(8)                 Class-aware wrapper for `::pdf_resolve_indirect()`.
(12)                    Resolve an indirect object (or chain of objects).
(12)                    This can cause xref reorganisations (solidifications etc) due to
(12)                    repairs, so all held pdf_xref_entries should be considered
(12)                    invalid after this call (other than the returned one).
(8)                 """
(8)                 return _mupdf.PdfObj_pdf_resolve_indirect(self)
(4)             def pdf_resolve_indirect_chain(self):
(8)                 r""" Class-aware wrapper for `::pdf_resolve_indirect_chain()`."""
(8)                 return _mupdf.PdfObj_pdf_resolve_indirect_chain(self)
(4)             def pdf_set_int(self, i):
(8)                 r""" Class-aware wrapper for `::pdf_set_int()`."""
(8)                 return _mupdf.PdfObj_pdf_set_int(self, i)
(4)             def pdf_set_obj_memo(self, bit, memo):
(8)                 r""" Class-aware wrapper for `::pdf_set_obj_memo()`."""
(8)                 return _mupdf.PdfObj_pdf_set_obj_memo(self, bit, memo)
(4)             def pdf_set_obj_parent(self, num):
(8)                 r""" Class-aware wrapper for `::pdf_set_obj_parent()`."""
(8)                 return _mupdf.PdfObj_pdf_set_obj_parent(self, num)
(4)             def pdf_set_str_len(self, newlen):
(8)                 r""" Class-aware wrapper for `::pdf_set_str_len()`."""
(8)                 return _mupdf.PdfObj_pdf_set_str_len(self, newlen)
(4)             def pdf_sort_dict(self):
(8)                 r""" Class-aware wrapper for `::pdf_sort_dict()`."""
(8)                 return _mupdf.PdfObj_pdf_sort_dict(self)
(4)             def pdf_store_item(self, val, itemsize):
(8)                 r""" Class-aware wrapper for `::pdf_store_item()`."""
(8)                 return _mupdf.PdfObj_pdf_store_item(self, val, itemsize)
(4)             def pdf_to_bool(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_bool()`."""
(8)                 return _mupdf.PdfObj_pdf_to_bool(self)
(4)             def pdf_to_bool_default(self, _def):
(8)                 r""" Class-aware wrapper for `::pdf_to_bool_default()`."""
(8)                 return _mupdf.PdfObj_pdf_to_bool_default(self, _def)
(4)             def pdf_to_date(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_date()`."""
(8)                 return _mupdf.PdfObj_pdf_to_date(self)
(4)             def pdf_to_gen(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_gen()`."""
(8)                 return _mupdf.PdfObj_pdf_to_gen(self)
(4)             def pdf_to_int(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_int()`."""
(8)                 return _mupdf.PdfObj_pdf_to_int(self)
(4)             def pdf_to_int64(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_int64()`."""
(8)                 return _mupdf.PdfObj_pdf_to_int64(self)
(4)             def pdf_to_int_default(self, _def):
(8)                 r""" Class-aware wrapper for `::pdf_to_int_default()`."""
(8)                 return _mupdf.PdfObj_pdf_to_int_default(self, _def)
(4)             def pdf_to_matrix(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_matrix()`."""
(8)                 return _mupdf.PdfObj_pdf_to_matrix(self)
(4)             def pdf_to_name(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_name()`."""
(8)                 return _mupdf.PdfObj_pdf_to_name(self)
(4)             def pdf_to_num(self):
(8)                 r""" Class-aware wrapper for `::pdf_to_num()`."""
(8)                 return _mupdf.PdfObj_pdf_to_num(self)
(4)             def pdf_to_quad(self, offset):
(8)                 r""" Class-aware wrapper for `::pdf_to_quad()`."""
(8)                 return _mupdf.PdfObj_pdf_to_quad(self, offset)
(4)             def pdf_to_real(self):
```

```
28617: (8)                  r""" Class-aware wrapper for `::pdf_to_real()`."""
28618: (8)                  return _mupdf.PdfObj_pdf_to_real(self)
28619: (4)              def pdf_to_real_default(self, _def):
28620: (8)                  r""" Class-aware wrapper for `::pdf_to_real_default()`."""
28621: (8)                  return _mupdf.PdfObj_pdf_to_real_default(self, _def)
28622: (4)              def pdf_to_rect(self):
28623: (8)                  r""" Class-aware wrapper for `::pdf_to_rect()`."""
28624: (8)                  return _mupdf.PdfObj_pdf_to_rect(self)
28625: (4)              def pdf_to_str_buf(self):
28626: (8)                  r""" Class-aware wrapper for `::pdf_to_str_buf()`."""
28627: (8)                  return _mupdf.PdfObj_pdf_to_str_buf(self)
28628: (4)              def pdf_to_str_len(self):
28629: (8)                  r""" Class-aware wrapper for `::pdf_to_str_len()`."""
28630: (8)                  return _mupdf.PdfObj_pdf_to_str_len(self)
28631: (4)              def pdf_to_string(self, sizep):
28632: (8)                  r"""
28633: (8)                  Class-aware wrapper for `::pdf_to_string()`.
28634: (8)                  This method has out-params. Python/C# wrappers look like:
28635: (12)                     `pdf_to_string()` => `(const char *, size_t sizep)`
28636: (8)                  """
28637: (8)                  return _mupdf.PdfObj_pdf_to_string(self, sizep)
28638: (4)              def pdf_to_text_string(self):
28639: (8)                  r""" Class-aware wrapper for `::pdf_to_text_string()`."""
28640: (8)                  return _mupdf.PdfObj_pdf_to_text_string(self)
28641: (4)              def pdf_unmark_obj(self):
28642: (8)                  r""" Class-aware wrapper for `::pdf_unmark_obj()`."""
28643: (8)                  return _mupdf.PdfObj_pdf_unmark_obj(self)
28644: (4)              def pdf_verify_embedded_file_checksum(self):
28645: (8)                  r""" Class-aware wrapper for
`::pdf_verify_embedded_file_checksum()`."""
28646: (8)                  return _mupdf.PdfObj_pdf_verify_embedded_file_checksum(self)
28647: (4)              def pdf_walk_tree(self, kid_name, arrive, leave, arg, names, values):
28648: (8)                  r"""
28649: (8)                  Class-aware wrapper for `::pdf_walk_tree()`.
28650: (8)                  This method has out-params. Python/C# wrappers look like:
28651: (12)                     `pdf_walk_tree(::pdf_obj *kid_name, void (*arrive)(::fz_context *,
::pdf_obj *, void *, ::pdf_obj **), void (*leave)(::fz_context *, ::pdf_obj *, void *), void *arg,
::pdf_obj **names, ::pdf_obj **values)` => `()`
28652: (8)                  """
28653: (8)                  return _mupdf.PdfObj_pdf_walk_tree(self, kid_name, arrive, leave, arg,
names, values)
28654: (4)              def pdf_xobject_bbox(self):
28655: (8)                  r""" Class-aware wrapper for `::pdf_xobject_bbox()`."""
28656: (8)                  return _mupdf.PdfObj_pdf_xobject_bbox(self)
28657: (4)              def pdf_xobject_colorspace(self):
28658: (8)                  r""" Class-aware wrapper for `::pdf_xobject_colorspace()`."""
28659: (8)                  return _mupdf.PdfObj_pdf_xobject_colorspace(self)
28660: (4)              def pdf_xobject_isolated(self):
28661: (8)                  r""" Class-aware wrapper for `::pdf_xobject_isolated()`."""
28662: (8)                  return _mupdf.PdfObj_pdf_xobject_isolated(self)
28663: (4)              def pdf_xobject_knockout(self):
28664: (8)                  r""" Class-aware wrapper for `::pdf_xobject_knockout()`."""
28665: (8)                  return _mupdf.PdfObj_pdf_xobject_knockout(self)
28666: (4)              def pdf_xobject_matrix(self):
28667: (8)                  r""" Class-aware wrapper for `::pdf_xobject_matrix()`."""
28668: (8)                  return _mupdf.PdfObj_pdf_xobject_matrix(self)
28669: (4)              def pdf_xobject_resources(self):
28670: (8)                  r""" Class-aware wrapper for `::pdf_xobject_resources()`."""
28671: (8)                  return _mupdf.PdfObj_pdf_xobject_resources(self)
28672: (4)              def pdf_xobject_transparency(self):
28673: (8)                  r""" Class-aware wrapper for `::pdf_xobject_transparency()`."""
28674: (8)                  return _mupdf.PdfObj_pdf_xobject_transparency(self)
28675: (4)              def pdf_dict_get(self, *args):
28676: (8)                  r"""
28677: (8)                  *Overload 1:*
28678: (8)                  Class-aware wrapper for `::pdf_dict_get()`.
28679: (8)                  |
28680: (8)                  *Overload 2:*
28681: (8)                  Typesafe wrapper for looking up things such as PDF_ENUM_NAME_Annots.
```

```
28682: (8)                    """
28683: (8)                    return _mupdf.PdfObj_pdf_dict_get(self, *args)
28684: (4)                def pdf_load_field_name2(self):
28685: (8)                    r""" Alternative to `pdf_load_field_name()` that returns a
std::string."""
28686: (8)                    return _mupdf.PdfObj_pdf_load_field_name2(self)
28687: (4)                def __init__(self, *args):
28688: (8)                    r"""
28689: (8)                    *Overload 1:*
28690: (8)                    == Constructors.  Constructor using `pdf_new_action_from_link()`.
28691: (8)                    |
28692: (8)                    *Overload 2:*
28693: (8)                    Constructor using `pdf_new_array()`.
28694: (8)                    |
28695: (8)                    *Overload 3:*
28696: (8)                    Constructor using `pdf_new_date()`.
28697: (8)                    |
28698: (8)                    *Overload 4:*
28699: (8)                    Constructor using `pdf_new_dest_from_link()`.
28700: (8)                    |
28701: (8)                    *Overload 5:*
28702: (8)                    Constructor using `pdf_new_indirect()`.
28703: (8)                    |
28704: (8)                    *Overload 6:*
28705: (8)                    Constructor using `pdf_new_int()`.
28706: (8)                    |
28707: (8)                    *Overload 7:*
28708: (8)                    Constructor using `pdf_new_matrix()`.
28709: (8)                    |
28710: (8)                    *Overload 8:*
28711: (8)                    Constructor using `pdf_new_name()`.
28712: (8)                    |
28713: (8)                    *Overload 9:*
28714: (8)                    Constructor using `pdf_new_real()`.
28715: (8)                    |
28716: (8)                    *Overload 10:*
28717: (8)                    Constructor using `pdf_new_rect()`.
28718: (8)                    |
28719: (8)                    *Overload 11:*
28720: (8)                    Constructor using `pdf_new_string()`.
28721: (8)                    |
28722: (8)                    *Overload 12:*
28723: (8)                    Constructor using `pdf_new_xobject()`.
28724: (8)                    |
28725: (8)                    *Overload 13:*
28726: (8)                    Copy constructor using `pdf_keep_obj()`.
28727: (8)                    |
28728: (8)                    *Overload 14:*
28729: (8)                    Constructor using raw copy of pre-existing `::pdf_obj`.
28730: (8)                    |
28731: (8)                    *Overload 15:*
28732: (8)                    Constructor using raw copy of pre-existing `::pdf_obj`.
28733: (8)                    """
28734: (8)                    _mupdf.PdfObj_swiginit(self, _mupdf.new_PdfObj(*args))
28735: (4)                __swig_destroy__ = _mupdf.delete_PdfObj
28736: (4)                def m_internal_value(self):
28737: (8)                    r""" Return numerical value of .m_internal; helps with Python
debugging."""
28738: (8)                    return _mupdf.PdfObj_m_internal_value(self)
28739: (4)                m_internal = property(_mupdf.PdfObj_m_internal_get,
_mupdf.PdfObj_m_internal_set, doc=r""" Pointer to wrapped data.""")
28740: (4)                s_num_instances = property(_mupdf.PdfObj_s_num_instances_get,
_mupdf.PdfObj_s_num_instances_set)
28741: (0)            _mupdf.PdfObj_swigregister(PdfObj)
28742: (0)            class PdfOcgDescriptor(object):
28743: (4)                r""" Wrapper class for struct `pdf_ocg_descriptor`. Not copyable or
assignable."""
28744: (4)                thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
28745: (4)              __repr__ = _swig_repr
28746: (4)              def __init__(self, *args):
28747: (8)                  r"""
28748: (8)                  *Overload 1:*
28749: (8)                  Default constructor, sets `m_internal` to null.
28750: (8)                  |
28751: (8)                  *Overload 2:*
28752: (8)                  Constructor using raw copy of pre-existing `::pdf_ocg_descriptor`.
28753: (8)                  """
28754: (8)                  _mupdf.PdfOcgDescriptor_swiginit(self,
_mupdf.new_PdfOcgDescriptor(*args))
28755: (4)              __swig_destroy__ = _mupdf.delete_PdfOcgDescriptor
28756: (4)              def m_internal_value(self):
28757: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28758: (8)                  return _mupdf.PdfOcgDescriptor_m_internal_value(self)
28759: (4)              m_internal = property(_mupdf.PdfOcgDescriptor_m_internal_get,
_mupdf.PdfOcgDescriptor_m_internal_set, doc=r""" Pointer to wrapped data.""")
28760: (4)              s_num_instances = property(_mupdf.PdfOcgDescriptor_s_num_instances_get,
_mupdf.PdfOcgDescriptor_s_num_instances_set)
28761: (0)          _mupdf.PdfOcgDescriptor_swigregister(PdfOcgDescriptor)
28762: (0)          class PdfPage(object):
28763: (4)              r""" Wrapper class for struct `pdf_page`."""
28764: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28765: (4)              __repr__ = _swig_repr
28766: (4)              def pdf_bound_page(self, box):
28767: (8)                  r""" Class-aware wrapper for `::pdf_bound_page()`."""
28768: (8)                  return _mupdf.PdfPage_pdf_bound_page(self, box)
28769: (4)              def pdf_create_annot(self, type):
28770: (8)                  r""" Class-aware wrapper for `::pdf_create_annot()`."""
28771: (8)                  return _mupdf.PdfPage_pdf_create_annot(self, type)
28772: (4)              def pdf_create_annot_raw(self, type):
28773: (8)                  r""" Class-aware wrapper for `::pdf_create_annot_raw()`."""
28774: (8)                  return _mupdf.PdfPage_pdf_create_annot_raw(self, type)
28775: (4)              def pdf_create_link(self, bbox, uri):
28776: (8)                  r""" Class-aware wrapper for `::pdf_create_link()`."""
28777: (8)                  return _mupdf.PdfPage_pdf_create_link(self, bbox, uri)
28778: (4)              def pdf_create_signature_widget(self, name):
28779: (8)                  r""" Class-aware wrapper for `::pdf_create_signature_widget()`."""
28780: (8)                  return _mupdf.PdfPage_pdf_create_signature_widget(self, name)
28781: (4)              def pdf_delete_annot(self, annot):
28782: (8)                  r""" Class-aware wrapper for `::pdf_delete_annot()`."""
28783: (8)                  return _mupdf.PdfPage_pdf_delete_annot(self, annot)
28784: (4)              def pdf_delete_link(self, link):
28785: (8)                  r""" Class-aware wrapper for `::pdf_delete_link()`."""
28786: (8)                  return _mupdf.PdfPage_pdf_delete_link(self, link)
28787: (4)              def pdf_first_annot(self):
28788: (8)                  r""" Class-aware wrapper for `::pdf_first_annot()`."""
28789: (8)                  return _mupdf.PdfPage_pdf_first_annot(self)
28790: (4)              def pdf_first_widget(self):
28791: (8)                  r""" Class-aware wrapper for `::pdf_first_widget()`."""
28792: (8)                  return _mupdf.PdfPage_pdf_first_widget(self)
28793: (4)              def pdf_load_links(self):
28794: (8)                  r""" Class-aware wrapper for `::pdf_load_links()`."""
28795: (8)                  return _mupdf.PdfPage_pdf_load_links(self)
28796: (4)              def pdf_new_link(self, rect, uri, obj):
28797: (8)                  r""" Class-aware wrapper for `::pdf_new_link()`."""
28798: (8)                  return _mupdf.PdfPage_pdf_new_link(self, rect, uri, obj)
28799: (4)              def pdf_new_pixmap_from_page_contents_with_separations_and_usage(self,
ctm, cs, seps, alpha, usage, box):
28800: (8)                  r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_contents_with_separations_and_usage()`."""
28801: (8)                  return
_mupdf.PdfPage_pdf_new_pixmap_from_page_contents_with_separations_and_usage(self, ctm, cs, seps,
alpha, usage, box)
28802: (4)              def pdf_new_pixmap_from_page_contents_with_usage(self, ctm, cs, alpha,
usage, box):
28803: (8)                  r""" Class-aware wrapper for
```

```
`::pdf_new_pixmap_from_page_contents_with_usage()`."""
28804: (8)                  return
_mupdf.PdfPage_pdf_new_pixmap_from_page_contents_with_usage(self, ctm, cs, alpha, usage, box)
28805: (4)              def pdf_new_pixmap_from_page_with_separations_and_usage(self, ctm, cs,
seps, alpha, usage, box):
28806: (8)                  r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_with_separations_and_usage()`."""
28807: (8)                  return
_mupdf.PdfPage_pdf_new_pixmap_from_page_with_separations_and_usage(self, ctm, cs, seps, alpha,
usage, box)
28808: (4)              def pdf_new_pixmap_from_page_with_usage(self, ctm, cs, alpha, usage, box):
28809: (8)                  r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_with_usage()`."""
28810: (8)                  return _mupdf.PdfPage_pdf_new_pixmap_from_page_with_usage(self, ctm,
cs, alpha, usage, box)
28811: (4)              def pdf_page_contents(self):
28812: (8)                  r""" Class-aware wrapper for `::pdf_page_contents()`."""
28813: (8)                  return _mupdf.PdfPage_pdf_page_contents(self)
28814: (4)              def pdf_page_event_close(self):
28815: (8)                  r""" Class-aware wrapper for `::pdf_page_event_close()`."""
28816: (8)                  return _mupdf.PdfPage_pdf_page_event_close(self)
28817: (4)              def pdf_page_event_open(self):
28818: (8)                  r""" Class-aware wrapper for `::pdf_page_event_open()`."""
28819: (8)                  return _mupdf.PdfPage_pdf_page_event_open(self)
28820: (4)              def pdf_page_group(self):
28821: (8)                  r""" Class-aware wrapper for `::pdf_page_group()`."""
28822: (8)                  return _mupdf.PdfPage_pdf_page_group(self)
28823: (4)              def pdf_page_has_transparency(self):
28824: (8)                  r""" Class-aware wrapper for `::pdf_page_has_transparency()`."""
28825: (8)                  return _mupdf.PdfPage_pdf_page_has_transparency(self)
28826: (4)              def pdf_page_presentation(self, transition, duration):
28827: (8)                  r"""
28828: (8)                  Class-aware wrapper for `::pdf_page_presentation()`.
28829: (8)                  This method has out-params. Python/C# wrappers look like:
28830: (12)                     `pdf_page_presentation(::fz_transition *transition)` =>
`(fz_transition *, float duration)`
28831: (8)                  """
28832: (8)                  return _mupdf.PdfPage_pdf_page_presentation(self, transition,
duration)
28833: (4)              def pdf_page_resources(self):
28834: (8)                  r""" Class-aware wrapper for `::pdf_page_resources()`."""
28835: (8)                  return _mupdf.PdfPage_pdf_page_resources(self)
28836: (4)              def pdf_page_separations(self):
28837: (8)                  r""" Class-aware wrapper for `::pdf_page_separations()`."""
28838: (8)                  return _mupdf.PdfPage_pdf_page_separations(self)
28839: (4)              def pdf_page_transform(self, mediabox, ctm):
28840: (8)                  r""" Class-aware wrapper for `::pdf_page_transform()`."""
28841: (8)                  return _mupdf.PdfPage_pdf_page_transform(self, mediabox, ctm)
28842: (4)              def pdf_page_transform_box(self, mediabox, ctm, box):
28843: (8)                  r""" Class-aware wrapper for `::pdf_page_transform_box()`."""
28844: (8)                  return _mupdf.PdfPage_pdf_page_transform_box(self, mediabox, ctm, box)
28845: (4)              def pdf_run_page(self, dev, ctm, cookie):
28846: (8)                  r""" Class-aware wrapper for `::pdf_run_page()`."""
28847: (8)                  return _mupdf.PdfPage_pdf_run_page(self, dev, ctm, cookie)
28848: (4)              def pdf_run_page_annots(self, dev, ctm, cookie):
28849: (8)                  r""" Class-aware wrapper for `::pdf_run_page_annots()`."""
28850: (8)                  return _mupdf.PdfPage_pdf_run_page_annots(self, dev, ctm, cookie)
28851: (4)              def pdf_run_page_annots_with_usage(self, dev, ctm, usage, cookie):
28852: (8)                  r""" Class-aware wrapper for `::pdf_run_page_annots_with_usage()`."""
28853: (8)                  return _mupdf.PdfPage_pdf_run_page_annots_with_usage(self, dev, ctm,
usage, cookie)
28854: (4)              def pdf_run_page_contents(self, dev, ctm, cookie):
28855: (8)                  r""" Class-aware wrapper for `::pdf_run_page_contents()`."""
28856: (8)                  return _mupdf.PdfPage_pdf_run_page_contents(self, dev, ctm, cookie)
28857: (4)              def pdf_run_page_contents_with_usage(self, dev, ctm, usage, cookie):
28858: (8)                  r""" Class-aware wrapper for
`::pdf_run_page_contents_with_usage()`."""
28859: (8)                  return _mupdf.PdfPage_pdf_run_page_contents_with_usage(self, dev, ctm,
usage, cookie)
```

```
28860: (4)              def pdf_run_page_widgets(self, dev, ctm, cookie):
28861: (8)                  r""" Class-aware wrapper for `::pdf_run_page_widgets()`."""
28862: (8)                  return _mupdf.PdfPage_pdf_run_page_widgets(self, dev, ctm, cookie)
28863: (4)              def pdf_run_page_widgets_with_usage(self, dev, ctm, usage, cookie):
28864: (8)                  r""" Class-aware wrapper for `::pdf_run_page_widgets_with_usage()`."""
28865: (8)                  return _mupdf.PdfPage_pdf_run_page_widgets_with_usage(self, dev, ctm,
usage, cookie)
28866: (4)              def pdf_run_page_with_usage(self, dev, ctm, usage, cookie):
28867: (8)                  r""" Class-aware wrapper for `::pdf_run_page_with_usage()`."""
28868: (8)                  return _mupdf.PdfPage_pdf_run_page_with_usage(self, dev, ctm, usage,
cookie)
28869: (4)              def pdf_set_page_box(self, box, rect):
28870: (8)                  r""" Class-aware wrapper for `::pdf_set_page_box()`."""
28871: (8)                  return _mupdf.PdfPage_pdf_set_page_box(self, box, rect)
28872: (4)              def pdf_update_page(self):
28873: (8)                  r""" Class-aware wrapper for `::pdf_update_page()`."""
28874: (8)                  return _mupdf.PdfPage_pdf_update_page(self)
28875: (4)              def super(self):
28876: (8)                  r""" Returns wrapper for .super member."""
28877: (8)                  return _mupdf.PdfPage_super(self)
28878: (4)              def doc(self):
28879: (8)                  r""" Returns wrapper for .doc member."""
28880: (8)                  return _mupdf.PdfPage_doc(self)
28881: (4)              def obj(self):
28882: (8)                  r""" Returns wrapper for .obj member."""
28883: (8)                  return _mupdf.PdfPage_obj(self)
28884: (4)              def __init__(self, *args):
28885: (8)                  r"""
28886: (8)                  *Overload 1:*
28887: (8)                  Copy constructor using `pdf_keep_page()`.
28888: (8)                  |
28889: (8)                  *Overload 2:*
28890: (8)                  Default constructor, sets `m_internal` to null.
28891: (8)                  |
28892: (8)                  *Overload 3:*
28893: (8)                  Constructor using raw copy of pre-existing `::pdf_page`.
28894: (8)                  """
28895: (8)                  _mupdf.PdfPage_swiginit(self, _mupdf.new_PdfPage(*args))
28896: (4)              __swig_destroy__ = _mupdf.delete_PdfPage
28897: (4)              def m_internal_value(self):
28898: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28899: (8)                  return _mupdf.PdfPage_m_internal_value(self)
28900: (4)              m_internal = property(_mupdf.PdfPage_m_internal_get,
_mupdf.PdfPage_m_internal_set, doc=r""" Pointer to wrapped data.""")
28901: (4)              s_num_instances = property(_mupdf.PdfPage_s_num_instances_get,
_mupdf.PdfPage_s_num_instances_set)
28902: (0)          _mupdf.PdfPage_swigregister(PdfPage)
28903: (0)          class PdfPattern(object):
28904: (4)              r""" Wrapper class for struct `pdf_pattern`."""
28905: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28906: (4)              __repr__ = _swig_repr
28907: (4)              def __init__(self, *args):
28908: (8)                  r"""
28909: (8)                  *Overload 1:*
28910: (8)                  Copy constructor using `pdf_keep_pattern()`.
28911: (8)                  |
28912: (8)                  *Overload 2:*
28913: (8)                  Default constructor, sets `m_internal` to null.
28914: (8)                  |
28915: (8)                  *Overload 3:*
28916: (8)                  Constructor using raw copy of pre-existing `::pdf_pattern`.
28917: (8)                  """
28918: (8)                  _mupdf.PdfPattern_swiginit(self, _mupdf.new_PdfPattern(*args))
28919: (4)              __swig_destroy__ = _mupdf.delete_PdfPattern
28920: (4)              def m_internal_value(self):
28921: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
```

```
28922: (8)                  return _mupdf.PdfPattern_m_internal_value(self)
28923: (4)              m_internal = property(_mupdf.PdfPattern_m_internal_get,
_mupdf.PdfPattern_m_internal_set, doc=r""" Pointer to wrapped data.""")
28924: (4)              s_num_instances = property(_mupdf.PdfPattern_s_num_instances_get,
_mupdf.PdfPattern_s_num_instances_set)
28925: (0)          _mupdf.PdfPattern_swigregister(PdfPattern)
28926: (0)          class PdfPkcs7DistinguishedName(object):
28927: (4)              r""" Wrapper class for struct `pdf_pkcs7_distinguished_name`. Not copyable
or assignable."""
28928: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28929: (4)              __repr__ = _swig_repr
28930: (4)              def pdf_signature_drop_distinguished_name(self):
28931: (8)                  r""" Class-aware wrapper for
`::pdf_signature_drop_distinguished_name()`."""
28932: (8)                  return
_mupdf.PdfPkcs7DistinguishedName_pdf_signature_drop_distinguished_name(self)
28933: (4)              def pdf_signature_format_distinguished_name(self):
28934: (8)                  r""" Class-aware wrapper for
`::pdf_signature_format_distinguished_name()`."""
28935: (8)                  return
_mupdf.PdfPkcs7DistinguishedName_pdf_signature_format_distinguished_name(self)
28936: (4)              def __init__(self, *args):
28937: (8)                  r"""
28938: (8)                  *Overload 1:*
28939: (8)                  Default constructor, sets `m_internal` to null.
28940: (8)                  |
28941: (8)                  *Overload 2:*
28942: (8)                  Constructor using raw copy of pre-existing
`::pdf_pkcs7_distinguished_name`.
28943: (8)                  """
28944: (8)                  _mupdf.PdfPkcs7DistinguishedName_swiginit(self,
_mupdf.new_PdfPkcs7DistinguishedName(*args))
28945: (4)              __swig_destroy__ = _mupdf.delete_PdfPkcs7DistinguishedName
28946: (4)              def m_internal_value(self):
28947: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28948: (8)                  return _mupdf.PdfPkcs7DistinguishedName_m_internal_value(self)
28949: (4)              m_internal = property(_mupdf.PdfPkcs7DistinguishedName_m_internal_get,
_mupdf.PdfPkcs7DistinguishedName_m_internal_set, doc=r""" Pointer to wrapped data.""")
28950: (4)              s_num_instances =
property(_mupdf.PdfPkcs7DistinguishedName_s_num_instances_get,
_mupdf.PdfPkcs7DistinguishedName_s_num_instances_set)
28951: (0)          _mupdf.PdfPkcs7DistinguishedName_swigregister(PdfPkcs7DistinguishedName)
28952: (0)          class PdfPkcs7Signer(object):
28953: (4)              r""" Wrapper class for struct `pdf_pkcs7_signer`. Not copyable or
assignable."""
28954: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28955: (4)              __repr__ = _swig_repr
28956: (4)              def __init__(self, *args):
28957: (8)                  r"""
28958: (8)                  *Overload 1:*
28959: (8)                  Default constructor, sets `m_internal` to null.
28960: (8)                  |
28961: (8)                  *Overload 2:*
28962: (8)                  Constructor using raw copy of pre-existing `::pdf_pkcs7_signer`.
28963: (8)                  """
28964: (8)                  _mupdf.PdfPkcs7Signer_swiginit(self, _mupdf.new_PdfPkcs7Signer(*args))
28965: (4)              __swig_destroy__ = _mupdf.delete_PdfPkcs7Signer
28966: (4)              def m_internal_value(self):
28967: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28968: (8)                  return _mupdf.PdfPkcs7Signer_m_internal_value(self)
28969: (4)              m_internal = property(_mupdf.PdfPkcs7Signer_m_internal_get,
_mupdf.PdfPkcs7Signer_m_internal_set, doc=r""" Pointer to wrapped data.""")
28970: (4)              s_num_instances = property(_mupdf.PdfPkcs7Signer_s_num_instances_get,
_mupdf.PdfPkcs7Signer_s_num_instances_set)
28971: (0)          _mupdf.PdfPkcs7Signer_swigregister(PdfPkcs7Signer)
```

```
28972: (0)          class PdfPkcs7Verifier(object):
28973: (4)              r""" Wrapper class for struct `pdf_pkcs7_verifier`. Not copyable or
assignable."""
28974: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28975: (4)              __repr__ = _swig_repr
28976: (4)              def __init__(self, *args):
28977: (8)                  r"""
28978: (8)                  *Overload 1:*
28979: (8)                  Default constructor, sets `m_internal` to null.
28980: (8)                  |
28981: (8)                  *Overload 2:*
28982: (8)                  Constructor using raw copy of pre-existing `::pdf_pkcs7_verifier`.
28983: (8)                  """
28984: (8)                  _mupdf.PdfPkcs7Verifier_swiginit(self,
_mupdf.new_PdfPkcs7Verifier(*args))
28985: (4)              __swig_destroy__ = _mupdf.delete_PdfPkcs7Verifier
28986: (4)              def m_internal_value(self):
28987: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
28988: (8)                  return _mupdf.PdfPkcs7Verifier_m_internal_value(self)
28989: (4)              m_internal = property(_mupdf.PdfPkcs7Verifier_m_internal_get,
_mupdf.PdfPkcs7Verifier_m_internal_set, doc=r""" Pointer to wrapped data.""")
28990: (4)              s_num_instances = property(_mupdf.PdfPkcs7Verifier_s_num_instances_get,
_mupdf.PdfPkcs7Verifier_s_num_instances_set)
28991: (0)          _mupdf.PdfPkcs7Verifier_swigregister(PdfPkcs7Verifier)
28992: (0)          class PdfProcessor(object):
28993: (4)              r""" Wrapper class for struct `pdf_processor`."""
28994: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
28995: (4)              __repr__ = _swig_repr
28996: (4)              @staticmethod
28997: (4)              def pdf_new_color_filter(doc, chain, struct_parents, transform, options,
copts):
28998: (8)                  r""" Class-aware wrapper for `::pdf_new_color_filter()`."""
28999: (8)                  return _mupdf.PdfProcessor_pdf_new_color_filter(doc, chain,
struct_parents, transform, options, copts)
29000: (4)              def pdf_close_processor(self):
29001: (8)                  r""" Class-aware wrapper for `::pdf_close_processor()`."""
29002: (8)                  return _mupdf.PdfProcessor_pdf_close_processor(self)
29003: (4)              def pdf_process_annot(self, annot, cookie):
29004: (8)                  r""" Class-aware wrapper for `::pdf_process_annot()`."""
29005: (8)                  return _mupdf.PdfProcessor_pdf_process_annot(self, annot, cookie)
29006: (4)              def pdf_process_contents(self, doc, res, stm, cookie, out_res):
29007: (8)                  r"""
29008: (8)                  Class-aware wrapper for `::pdf_process_contents()`.
29009: (8)                  This method has out-params. Python/C# wrappers look like:
29010: (12)                     `pdf_process_contents(::pdf_document *doc, ::pdf_obj *res,
::pdf_obj *stm, ::fz_cookie *cookie, ::pdf_obj **out_res)` =>
29011: (8)                  """
29012: (8)                  return _mupdf.PdfProcessor_pdf_process_contents(self, doc, res, stm,
cookie, out_res)
29013: (4)              def pdf_process_glyph(self, doc, resources, contents):
29014: (8)                  r""" Class-aware wrapper for `::pdf_process_glyph()`."""
29015: (8)                  return _mupdf.PdfProcessor_pdf_process_glyph(self, doc, resources,
contents)
29016: (4)              def pdf_process_raw_contents(self, doc, rdb, stmobj, cookie):
29017: (8)                  r""" Class-aware wrapper for `::pdf_process_raw_contents()`."""
29018: (8)                  return _mupdf.PdfProcessor_pdf_process_raw_contents(self, doc, rdb,
stmobj, cookie)
29019: (4)              def pdf_processor_pop_resources(self):
29020: (8)                  r""" Class-aware wrapper for `::pdf_processor_pop_resources()`."""
29021: (8)                  return _mupdf.PdfProcessor_pdf_processor_pop_resources(self)
29022: (4)              def pdf_processor_push_resources(self, res):
29023: (8)                  r""" Class-aware wrapper for `::pdf_processor_push_resources()`."""
29024: (8)                  return _mupdf.PdfProcessor_pdf_processor_push_resources(self, res)
29025: (4)              def pdf_reset_processor(self):
29026: (8)                  r""" Class-aware wrapper for `::pdf_reset_processor()`."""
29027: (8)                  return _mupdf.PdfProcessor_pdf_reset_processor(self)
```

```
29028: (4)              def __init__(self, *args):
29029: (8)                  r"""
29030: (8)                  *Overload 1:*
29031: (8)                  == Constructors.  Constructor using `pdf_new_buffer_processor()`.
29032: (8)                  |
29033: (8)                  *Overload 2:*
29034: (8)                  Constructor using `pdf_new_output_processor()`.
29035: (8)                  |
29036: (8)                  *Overload 3:*
29037: (8)                  Constructor using `pdf_new_run_processor()`.
29038: (8)                  |
29039: (8)                  *Overload 4:*
29040: (8)                  Constructor using `pdf_new_sanitize_filter()`.
29041: (8)                  |
29042: (8)                  *Overload 5:*
29043: (8)                  Copy constructor using `pdf_keep_processor()`.
29044: (8)                  |
29045: (8)                  *Overload 6:*
29046: (8)                  Default constructor, sets `m_internal` to null.
29047: (8)                  |
29048: (8)                  *Overload 7:*
29049: (8)                  Constructor using raw copy of pre-existing `::pdf_processor`.
29050: (8)                  """
29051: (8)                  _mupdf.PdfProcessor_swiginit(self, _mupdf.new_PdfProcessor(*args))
29052: (4)              __swig_destroy__ = _mupdf.delete_PdfProcessor
29053: (4)              def m_internal_value(self):
29054: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29055: (8)                  return _mupdf.PdfProcessor_m_internal_value(self)
29056: (4)              m_internal = property(_mupdf.PdfProcessor_m_internal_get,
_mupdf.PdfProcessor_m_internal_set, doc=r""" Pointer to wrapped data.""")
29057: (4)              s_num_instances = property(_mupdf.PdfProcessor_s_num_instances_get,
_mupdf.PdfProcessor_s_num_instances_set)
29058: (0)          _mupdf.PdfProcessor_swigregister(PdfProcessor)
29059: (0)          class PdfProcessor2(PdfProcessor):
29060: (4)              r""" Wrapper class for struct pdf_processor with virtual fns for each
fnptr; this is for use as a SWIG Director class."""
29061: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29062: (4)              __repr__ = _swig_repr
29063: (4)              def __init__(self):
29064: (8)                  r""" == Constructor."""
29065: (8)                  if self.__class__ == PdfProcessor2:
29066: (12)                     _self = None
29067: (8)                  else:
29068: (12)                     _self = self
29069: (8)                  _mupdf.PdfProcessor2_swiginit(self, _mupdf.new_PdfProcessor2(_self, ))
29070: (4)              def use_virtual_close_processor(self, use=True):
29071: (8)                  r"""
29072: (8)                  These methods set the function pointers in *m_internal
29073: (8)                  to point to internal callbacks that call our virtual methods.
29074: (8)                  """
29075: (8)                  return _mupdf.PdfProcessor2_use_virtual_close_processor(self, use)
29076: (4)              def use_virtual_drop_processor(self, use=True):
29077: (8)                  return _mupdf.PdfProcessor2_use_virtual_drop_processor(self, use)
29078: (4)              def use_virtual_reset_processor(self, use=True):
29079: (8)                  return _mupdf.PdfProcessor2_use_virtual_reset_processor(self, use)
29080: (4)              def use_virtual_push_resources(self, use=True):
29081: (8)                  return _mupdf.PdfProcessor2_use_virtual_push_resources(self, use)
29082: (4)              def use_virtual_pop_resources(self, use=True):
29083: (8)                  return _mupdf.PdfProcessor2_use_virtual_pop_resources(self, use)
29084: (4)              def use_virtual_op_w(self, use=True):
29085: (8)                  return _mupdf.PdfProcessor2_use_virtual_op_w(self, use)
29086: (4)              def use_virtual_op_j(self, use=True):
29087: (8)                  return _mupdf.PdfProcessor2_use_virtual_op_j(self, use)
29088: (4)              def use_virtual_op_J(self, use=True):
29089: (8)                  return _mupdf.PdfProcessor2_use_virtual_op_J(self, use)
29090: (4)              def use_virtual_op_M(self, use=True):
29091: (8)                  return _mupdf.PdfProcessor2_use_virtual_op_M(self, use)
```

```
(4)     def use_virtual_op_d(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_d(self, use)
(4)     def use_virtual_op_ri(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_ri(self, use)
(4)     def use_virtual_op_i(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_i(self, use)
(4)     def use_virtual_op_gs_begin(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_begin(self, use)
(4)     def use_virtual_op_gs_BM(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_BM(self, use)
(4)     def use_virtual_op_gs_ca(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_ca(self, use)
(4)     def use_virtual_op_gs_CA(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_CA(self, use)
(4)     def use_virtual_op_gs_SMask(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_SMask(self, use)
(4)     def use_virtual_op_gs_end(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_gs_end(self, use)
(4)     def use_virtual_op_q(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_q(self, use)
(4)     def use_virtual_op_Q(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_Q(self, use)
(4)     def use_virtual_op_cm(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_cm(self, use)
(4)     def use_virtual_op_m(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_m(self, use)
(4)     def use_virtual_op_l(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_l(self, use)
(4)     def use_virtual_op_c(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_c(self, use)
(4)     def use_virtual_op_v(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_v(self, use)
(4)     def use_virtual_op_y(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_y(self, use)
(4)     def use_virtual_op_h(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_h(self, use)
(4)     def use_virtual_op_re(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_re(self, use)
(4)     def use_virtual_op_S(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_S(self, use)
(4)     def use_virtual_op_s(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_s(self, use)
(4)     def use_virtual_op_F(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_F(self, use)
(4)     def use_virtual_op_f(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_f(self, use)
(4)     def use_virtual_op_fstar(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_fstar(self, use)
(4)     def use_virtual_op_B(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_B(self, use)
(4)     def use_virtual_op_Bstar(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_Bstar(self, use)
(4)     def use_virtual_op_b(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_b(self, use)
(4)     def use_virtual_op_bstar(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_bstar(self, use)
(4)     def use_virtual_op_n(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_n(self, use)
(4)     def use_virtual_op_W(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_W(self, use)
(4)     def use_virtual_op_Wstar(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_Wstar(self, use)
(4)     def use_virtual_op_BT(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_BT(self, use)
(4)     def use_virtual_op_ET(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_ET(self, use)
(4)     def use_virtual_op_Tc(self, use=True):
(8)         return _mupdf.PdfProcessor2_use_virtual_op_Tc(self, use)
(4)     def use_virtual_op_Tw(self, use=True):
```

```
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tw(self, use)
(4)             def use_virtual_op_Tz(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tz(self, use)
(4)             def use_virtual_op_TL(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_TL(self, use)
(4)             def use_virtual_op_Tf(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tf(self, use)
(4)             def use_virtual_op_Tr(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tr(self, use)
(4)             def use_virtual_op_Ts(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Ts(self, use)
(4)             def use_virtual_op_Td(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Td(self, use)
(4)             def use_virtual_op_TD(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_TD(self, use)
(4)             def use_virtual_op_Tm(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tm(self, use)
(4)             def use_virtual_op_Tstar(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tstar(self, use)
(4)             def use_virtual_op_TJ(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_TJ(self, use)
(4)             def use_virtual_op_Tj(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Tj(self, use)
(4)             def use_virtual_op_squote(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_squote(self, use)
(4)             def use_virtual_op_dquote(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_dquote(self, use)
(4)             def use_virtual_op_d0(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_d0(self, use)
(4)             def use_virtual_op_d1(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_d1(self, use)
(4)             def use_virtual_op_CS(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_CS(self, use)
(4)             def use_virtual_op_cs(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_cs(self, use)
(4)             def use_virtual_op_SC_pattern(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_SC_pattern(self, use)
(4)             def use_virtual_op_sc_pattern(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_sc_pattern(self, use)
(4)             def use_virtual_op_SC_shade(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_SC_shade(self, use)
(4)             def use_virtual_op_sc_shade(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_sc_shade(self, use)
(4)             def use_virtual_op_SC_color(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_SC_color(self, use)
(4)             def use_virtual_op_sc_color(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_sc_color(self, use)
(4)             def use_virtual_op_G(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_G(self, use)
(4)             def use_virtual_op_g(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_g(self, use)
(4)             def use_virtual_op_RG(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_RG(self, use)
(4)             def use_virtual_op_rg(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_rg(self, use)
(4)             def use_virtual_op_K(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_K(self, use)
(4)             def use_virtual_op_k(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_k(self, use)
(4)             def use_virtual_op_BI(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_BI(self, use)
(4)             def use_virtual_op_sh(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_sh(self, use)
(4)             def use_virtual_op_Do_image(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Do_image(self, use)
(4)             def use_virtual_op_Do_form(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_Do_form(self, use)
(4)             def use_virtual_op_MP(self, use=True):
(8)                 return _mupdf.PdfProcessor2_use_virtual_op_MP(self, use)
```

```
29230: (4)                def use_virtual_op_DP(self, use=True):
29231: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_DP(self, use)
29232: (4)                def use_virtual_op_BMC(self, use=True):
29233: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_BMC(self, use)
29234: (4)                def use_virtual_op_BDC(self, use=True):
29235: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_BDC(self, use)
29236: (4)                def use_virtual_op_EMC(self, use=True):
29237: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_EMC(self, use)
29238: (4)                def use_virtual_op_BX(self, use=True):
29239: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_BX(self, use)
29240: (4)                def use_virtual_op_EX(self, use=True):
29241: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_EX(self, use)
29242: (4)                def use_virtual_op_gs_OP(self, use=True):
29243: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_gs_OP(self, use)
29244: (4)                def use_virtual_op_gs_op(self, use=True):
29245: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_gs_op(self, use)
29246: (4)                def use_virtual_op_gs_OPM(self, use=True):
29247: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_gs_OPM(self, use)
29248: (4)                def use_virtual_op_gs_UseBlackPtComp(self, use=True):
29249: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_gs_UseBlackPtComp(self,
use)
29250: (4)                def use_virtual_op_END(self, use=True):
29251: (8)                    return _mupdf.PdfProcessor2_use_virtual_op_END(self, use)
29252: (4)                def close_processor(self, arg_0):
29253: (8)                    r""" Default virtual method implementations; these all throw an
exception."""
29254: (8)                    return _mupdf.PdfProcessor2_close_processor(self, arg_0)
29255: (4)                def drop_processor(self, arg_0):
29256: (8)                    return _mupdf.PdfProcessor2_drop_processor(self, arg_0)
29257: (4)                def reset_processor(self, arg_0):
29258: (8)                    return _mupdf.PdfProcessor2_reset_processor(self, arg_0)
29259: (4)                def push_resources(self, arg_0, arg_2):
29260: (8)                    return _mupdf.PdfProcessor2_push_resources(self, arg_0, arg_2)
29261: (4)                def pop_resources(self, arg_0):
29262: (8)                    return _mupdf.PdfProcessor2_pop_resources(self, arg_0)
29263: (4)                def op_w(self, arg_0, arg_2):
29264: (8)                    return _mupdf.PdfProcessor2_op_w(self, arg_0, arg_2)
29265: (4)                def op_j(self, arg_0, arg_2):
29266: (8)                    return _mupdf.PdfProcessor2_op_j(self, arg_0, arg_2)
29267: (4)                def op_J(self, arg_0, arg_2):
29268: (8)                    return _mupdf.PdfProcessor2_op_J(self, arg_0, arg_2)
29269: (4)                def op_M(self, arg_0, arg_2):
29270: (8)                    return _mupdf.PdfProcessor2_op_M(self, arg_0, arg_2)
29271: (4)                def op_d(self, arg_0, arg_2, arg_3):
29272: (8)                    return _mupdf.PdfProcessor2_op_d(self, arg_0, arg_2, arg_3)
29273: (4)                def op_ri(self, arg_0, arg_2):
29274: (8)                    return _mupdf.PdfProcessor2_op_ri(self, arg_0, arg_2)
29275: (4)                def op_i(self, arg_0, arg_2):
29276: (8)                    return _mupdf.PdfProcessor2_op_i(self, arg_0, arg_2)
29277: (4)                def op_gs_begin(self, arg_0, arg_2, arg_3):
29278: (8)                    return _mupdf.PdfProcessor2_op_gs_begin(self, arg_0, arg_2, arg_3)
29279: (4)                def op_gs_BM(self, arg_0, arg_2):
29280: (8)                    return _mupdf.PdfProcessor2_op_gs_BM(self, arg_0, arg_2)
29281: (4)                def op_gs_ca(self, arg_0, arg_2):
29282: (8)                    return _mupdf.PdfProcessor2_op_gs_ca(self, arg_0, arg_2)
29283: (4)                def op_gs_CA(self, arg_0, arg_2):
29284: (8)                    return _mupdf.PdfProcessor2_op_gs_CA(self, arg_0, arg_2)
29285: (4)                def op_gs_SMask(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29286: (8)                    return _mupdf.PdfProcessor2_op_gs_SMask(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
29287: (4)                def op_gs_end(self, arg_0):
29288: (8)                    return _mupdf.PdfProcessor2_op_gs_end(self, arg_0)
29289: (4)                def op_q(self, arg_0):
29290: (8)                    return _mupdf.PdfProcessor2_op_q(self, arg_0)
29291: (4)                def op_Q(self, arg_0):
29292: (8)                    return _mupdf.PdfProcessor2_op_Q(self, arg_0)
29293: (4)                def op_cm(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
29294: (8)                    return _mupdf.PdfProcessor2_op_cm(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
```

```
(4)             def op_m(self, arg_0, arg_2, arg_3):
(8)                 return _mupdf.PdfProcessor2_op_m(self, arg_0, arg_2, arg_3)
(4)             def op_l(self, arg_0, arg_2, arg_3):
(8)                 return _mupdf.PdfProcessor2_op_l(self, arg_0, arg_2, arg_3)
(4)             def op_c(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
(8)                 return _mupdf.PdfProcessor2_op_c(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7)
(4)             def op_v(self, arg_0, arg_2, arg_3, arg_4, arg_5):
(8)                 return _mupdf.PdfProcessor2_op_v(self, arg_0, arg_2, arg_3, arg_4, arg_5)
(4)             def op_y(self, arg_0, arg_2, arg_3, arg_4, arg_5):
(8)                 return _mupdf.PdfProcessor2_op_y(self, arg_0, arg_2, arg_3, arg_4, arg_5)
(4)             def op_h(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_h(self, arg_0)
(4)             def op_re(self, arg_0, arg_2, arg_3, arg_4, arg_5):
(8)                 return _mupdf.PdfProcessor2_op_re(self, arg_0, arg_2, arg_3, arg_4, arg_5)
(4)             def op_S(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_S(self, arg_0)
(4)             def op_s(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_s(self, arg_0)
(4)             def op_F(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_F(self, arg_0)
(4)             def op_f(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_f(self, arg_0)
(4)             def op_fstar(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_fstar(self, arg_0)
(4)             def op_B(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_B(self, arg_0)
(4)             def op_Bstar(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_Bstar(self, arg_0)
(4)             def op_b(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_b(self, arg_0)
(4)             def op_bstar(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_bstar(self, arg_0)
(4)             def op_n(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_n(self, arg_0)
(4)             def op_W(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_W(self, arg_0)
(4)             def op_Wstar(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_Wstar(self, arg_0)
(4)             def op_BT(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_BT(self, arg_0)
(4)             def op_ET(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_ET(self, arg_0)
(4)             def op_Tc(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_Tc(self, arg_0, arg_2)
(4)             def op_Tw(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_Tw(self, arg_0, arg_2)
(4)             def op_Tz(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_Tz(self, arg_0, arg_2)
(4)             def op_TL(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_TL(self, arg_0, arg_2)
(4)             def op_Tf(self, arg_0, arg_2, arg_3, arg_4):
(8)                 return _mupdf.PdfProcessor2_op_Tf(self, arg_0, arg_2, arg_3, arg_4)
(4)             def op_Tr(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_Tr(self, arg_0, arg_2)
(4)             def op_Ts(self, arg_0, arg_2):
(8)                 return _mupdf.PdfProcessor2_op_Ts(self, arg_0, arg_2)
(4)             def op_Td(self, arg_0, arg_2, arg_3):
(8)                 return _mupdf.PdfProcessor2_op_Td(self, arg_0, arg_2, arg_3)
(4)             def op_TD(self, arg_0, arg_2, arg_3):
(8)                 return _mupdf.PdfProcessor2_op_TD(self, arg_0, arg_2, arg_3)
(4)             def op_Tm(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
(8)                 return _mupdf.PdfProcessor2_op_Tm(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7)
(4)             def op_Tstar(self, arg_0):
(8)                 return _mupdf.PdfProcessor2_op_Tstar(self, arg_0)
```

```
29359: (4)              def op_TJ(self, arg_0, arg_2):
29360: (8)                  return _mupdf.PdfProcessor2_op_TJ(self, arg_0, arg_2)
29361: (4)              def op_Tj(self, arg_0, arg_2, arg_3):
29362: (8)                  return _mupdf.PdfProcessor2_op_Tj(self, arg_0, arg_2, arg_3)
29363: (4)              def op_squote(self, arg_0, arg_2, arg_3):
29364: (8)                  return _mupdf.PdfProcessor2_op_squote(self, arg_0, arg_2, arg_3)
29365: (4)              def op_dquote(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29366: (8)                  return _mupdf.PdfProcessor2_op_dquote(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
29367: (4)              def op_d0(self, arg_0, arg_2, arg_3):
29368: (8)                  return _mupdf.PdfProcessor2_op_d0(self, arg_0, arg_2, arg_3)
29369: (4)              def op_d1(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6, arg_7):
29370: (8)                  return _mupdf.PdfProcessor2_op_d1(self, arg_0, arg_2, arg_3, arg_4,
arg_5, arg_6, arg_7)
29371: (4)              def op_CS(self, arg_0, arg_2, arg_3):
29372: (8)                  return _mupdf.PdfProcessor2_op_CS(self, arg_0, arg_2, arg_3)
29373: (4)              def op_cs(self, arg_0, arg_2, arg_3):
29374: (8)                  return _mupdf.PdfProcessor2_op_cs(self, arg_0, arg_2, arg_3)
29375: (4)              def op_SC_pattern(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29376: (8)                  return _mupdf.PdfProcessor2_op_SC_pattern(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
29377: (4)              def op_sc_pattern(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29378: (8)                  return _mupdf.PdfProcessor2_op_sc_pattern(self, arg_0, arg_2, arg_3,
arg_4, arg_5)
29379: (4)              def op_SC_shade(self, arg_0, arg_2, arg_3):
29380: (8)                  return _mupdf.PdfProcessor2_op_SC_shade(self, arg_0, arg_2, arg_3)
29381: (4)              def op_sc_shade(self, arg_0, arg_2, arg_3):
29382: (8)                  return _mupdf.PdfProcessor2_op_sc_shade(self, arg_0, arg_2, arg_3)
29383: (4)              def op_SC_color(self, arg_0, arg_2, arg_3):
29384: (8)                  return _mupdf.PdfProcessor2_op_SC_color(self, arg_0, arg_2, arg_3)
29385: (4)              def op_sc_color(self, arg_0, arg_2, arg_3):
29386: (8)                  return _mupdf.PdfProcessor2_op_sc_color(self, arg_0, arg_2, arg_3)
29387: (4)              def op_G(self, arg_0, arg_2):
29388: (8)                  return _mupdf.PdfProcessor2_op_G(self, arg_0, arg_2)
29389: (4)              def op_g(self, arg_0, arg_2):
29390: (8)                  return _mupdf.PdfProcessor2_op_g(self, arg_0, arg_2)
29391: (4)              def op_RG(self, arg_0, arg_2, arg_3, arg_4):
29392: (8)                  return _mupdf.PdfProcessor2_op_RG(self, arg_0, arg_2, arg_3, arg_4)
29393: (4)              def op_rg(self, arg_0, arg_2, arg_3, arg_4):
29394: (8)                  return _mupdf.PdfProcessor2_op_rg(self, arg_0, arg_2, arg_3, arg_4)
29395: (4)              def op_K(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29396: (8)                  return _mupdf.PdfProcessor2_op_K(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
29397: (4)              def op_k(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29398: (8)                  return _mupdf.PdfProcessor2_op_k(self, arg_0, arg_2, arg_3, arg_4,
arg_5)
29399: (4)              def op_BI(self, arg_0, arg_2, arg_3):
29400: (8)                  return _mupdf.PdfProcessor2_op_BI(self, arg_0, arg_2, arg_3)
29401: (4)              def op_sh(self, arg_0, arg_2, arg_3):
29402: (8)                  return _mupdf.PdfProcessor2_op_sh(self, arg_0, arg_2, arg_3)
29403: (4)              def op_Do_image(self, arg_0, arg_2, arg_3):
29404: (8)                  return _mupdf.PdfProcessor2_op_Do_image(self, arg_0, arg_2, arg_3)
29405: (4)              def op_Do_form(self, arg_0, arg_2, arg_3):
29406: (8)                  return _mupdf.PdfProcessor2_op_Do_form(self, arg_0, arg_2, arg_3)
29407: (4)              def op_MP(self, arg_0, arg_2):
29408: (8)                  return _mupdf.PdfProcessor2_op_MP(self, arg_0, arg_2)
29409: (4)              def op_DP(self, arg_0, arg_2, arg_3, arg_4):
29410: (8)                  return _mupdf.PdfProcessor2_op_DP(self, arg_0, arg_2, arg_3, arg_4)
29411: (4)              def op_BMC(self, arg_0, arg_2):
29412: (8)                  return _mupdf.PdfProcessor2_op_BMC(self, arg_0, arg_2)
29413: (4)              def op_BDC(self, arg_0, arg_2, arg_3, arg_4):
29414: (8)                  return _mupdf.PdfProcessor2_op_BDC(self, arg_0, arg_2, arg_3, arg_4)
29415: (4)              def op_EMC(self, arg_0):
29416: (8)                  return _mupdf.PdfProcessor2_op_EMC(self, arg_0)
29417: (4)              def op_BX(self, arg_0):
29418: (8)                  return _mupdf.PdfProcessor2_op_BX(self, arg_0)
29419: (4)              def op_EX(self, arg_0):
29420: (8)                  return _mupdf.PdfProcessor2_op_EX(self, arg_0)
29421: (4)              def op_gs_OP(self, arg_0, arg_2):
```

```
29422: (8)                  return _mupdf.PdfProcessor2_op_gs_OP(self, arg_0, arg_2)
29423: (4)              def op_gs_op(self, arg_0, arg_2):
29424: (8)                  return _mupdf.PdfProcessor2_op_gs_op(self, arg_0, arg_2)
29425: (4)              def op_gs_OPM(self, arg_0, arg_2):
29426: (8)                  return _mupdf.PdfProcessor2_op_gs_OPM(self, arg_0, arg_2)
29427: (4)              def op_gs_UseBlackPtComp(self, arg_0, arg_2):
29428: (8)                  return _mupdf.PdfProcessor2_op_gs_UseBlackPtComp(self, arg_0, arg_2)
29429: (4)              def op_END(self, arg_0):
29430: (8)                  return _mupdf.PdfProcessor2_op_END(self, arg_0)
29431: (4)              __swig_destroy__ = _mupdf.delete_PdfProcessor2
29432: (4)              def __disown__(self):
29433: (8)                  self.this.disown()
29434: (8)                  _mupdf.disown_PdfProcessor2(self)
29435: (8)                  return weakref.proxy(self)
29436: (0)          _mupdf.PdfProcessor2_swigregister(PdfProcessor2)
29437: (0)          class PdfRange(object):
29438: (4)              r""" Wrapper class for struct `pdf_range`. Not copyable or assignable."""
29439: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29440: (4)              __repr__ = _swig_repr
29441: (4)              def __init__(self, *args):
29442: (8)                  r"""
29443: (8)                  *Overload 1:*
29444: (8)                  Default constructor, sets `m_internal` to null.
29445: (8)                  |
29446: (8)                  *Overload 2:*
29447: (8)                  Constructor using raw copy of pre-existing `::pdf_range`.
29448: (8)                  """
29449: (8)                  _mupdf.PdfRange_swiginit(self, _mupdf.new_PdfRange(*args))
29450: (4)              __swig_destroy__ = _mupdf.delete_PdfRange
29451: (4)              def m_internal_value(self):
29452: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29453: (8)                  return _mupdf.PdfRange_m_internal_value(self)
29454: (4)              m_internal = property(_mupdf.PdfRange_m_internal_get,
_mupdf.PdfRange_m_internal_set, doc=r""" Pointer to wrapped data.""")
29455: (4)              s_num_instances = property(_mupdf.PdfRange_s_num_instances_get,
_mupdf.PdfRange_s_num_instances_set)
29456: (0)          _mupdf.PdfRange_swigregister(PdfRange)
29457: (0)          class PdfRedactOptions(object):
29458: (4)              r""" Wrapper class for struct `pdf_redact_options`."""
29459: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29460: (4)              __repr__ = _swig_repr
29461: (4)              def __init__(self, *args):
29462: (8)                  r"""
29463: (8)                  *Overload 1:*
29464: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
29465: (8)                  |
29466: (8)                  *Overload 2:*
29467: (8)                  Constructor using raw copy of pre-existing `::pdf_redact_options`.
29468: (8)                  |
29469: (8)                  *Overload 3:*
29470: (8)                  Constructor using raw copy of pre-existing `::pdf_redact_options`.
29471: (8)                  """
29472: (8)                  _mupdf.PdfRedactOptions_swiginit(self,
_mupdf.new_PdfRedactOptions(*args))
29473: (4)              def internal(self, *args):
29474: (8)                  r"""
29475: (8)                  *Overload 1:*
29476: (8)                  Access as underlying struct.
29477: (8)                  |
29478: (8)                  *Overload 2:*
29479: (8)                  Access as underlying struct.
29480: (8)                  """
29481: (8)                  return _mupdf.PdfRedactOptions_internal(self, *args)
29482: (4)              __swig_destroy__ = _mupdf.delete_PdfRedactOptions
29483: (4)              black_boxes = property(_mupdf.PdfRedactOptions_black_boxes_get,
```

```
_mupdf.PdfRedactOptions_black_boxes_set)
29484: (4)              image_method = property(_mupdf.PdfRedactOptions_image_method_get,
_mupdf.PdfRedactOptions_image_method_set)
29485: (4)              line_art = property(_mupdf.PdfRedactOptions_line_art_get,
_mupdf.PdfRedactOptions_line_art_set)
29486: (4)              text = property(_mupdf.PdfRedactOptions_text_get,
_mupdf.PdfRedactOptions_text_set)
29487: (4)              s_num_instances = property(_mupdf.PdfRedactOptions_s_num_instances_get,
_mupdf.PdfRedactOptions_s_num_instances_set)
29488: (4)              def to_string(self):
29489: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
29490: (8)                  return _mupdf.PdfRedactOptions_to_string(self)
29491: (4)              def __eq__(self, rhs):
29492: (8)                  r""" Comparison method."""
29493: (8)                  return _mupdf.PdfRedactOptions___eq__(self, rhs)
29494: (4)              def __ne__(self, rhs):
29495: (8)                  r""" Comparison method."""
29496: (8)                  return _mupdf.PdfRedactOptions___ne__(self, rhs)
29497: (0)          _mupdf.PdfRedactOptions_swigregister(PdfRedactOptions)
29498: (0)          class PdfRevPageMap(object):
29499: (4)              r""" Wrapper class for struct `pdf_rev_page_map`. Not copyable or
assignable."""
29500: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29501: (4)              __repr__ = _swig_repr
29502: (4)              def __init__(self, *args):
29503: (8)                  r"""
29504: (8)                  *Overload 1:*
29505: (8)                  Default constructor, sets `m_internal` to null.
29506: (8)                  |
29507: (8)                  *Overload 2:*
29508: (8)                  Constructor using raw copy of pre-existing `::pdf_rev_page_map`.
29509: (8)                  """
29510: (8)                  _mupdf.PdfRevPageMap_swiginit(self, _mupdf.new_PdfRevPageMap(*args))
29511: (4)              __swig_destroy__ = _mupdf.delete_PdfRevPageMap
29512: (4)              def m_internal_value(self):
29513: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29514: (8)                  return _mupdf.PdfRevPageMap_m_internal_value(self)
29515: (4)              m_internal = property(_mupdf.PdfRevPageMap_m_internal_get,
_mupdf.PdfRevPageMap_m_internal_set, doc=r""" Pointer to wrapped data.""")
29516: (4)              s_num_instances = property(_mupdf.PdfRevPageMap_s_num_instances_get,
_mupdf.PdfRevPageMap_s_num_instances_set)
29517: (0)          _mupdf.PdfRevPageMap_swigregister(PdfRevPageMap)
29518: (0)          class PdfSanitizeFilterOptions(object):
29519: (4)              r""" Wrapper class for struct `pdf_sanitize_filter_options`."""
29520: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29521: (4)              __repr__ = _swig_repr
29522: (4)              def __init__(self, *args):
29523: (8)                  r"""
29524: (8)                  *Overload 1:*
29525: (8)                  We use default copy constructor and operator=.  Default constructor,
sets each member to default value.
29526: (8)                  |
29527: (8)                  *Overload 2:*
29528: (8)                  Constructor using raw copy of pre-existing
`::pdf_sanitize_filter_options`.
29529: (8)                  |
29530: (8)                  *Overload 3:*
29531: (8)                  Constructor using raw copy of pre-existing
`::pdf_sanitize_filter_options`.
29532: (8)                  """
29533: (8)                  _mupdf.PdfSanitizeFilterOptions_swiginit(self,
_mupdf.new_PdfSanitizeFilterOptions(*args))
29534: (4)              def internal(self, *args):
29535: (8)                  r"""
29536: (8)                  *Overload 1:*
```

```
29537: (8)                  Access as underlying struct.
29538: (8)                  |
29539: (8)                  *Overload 2:*
29540: (8)                  Access as underlying struct.
29541: (8)                  """
29542: (8)                  return _mupdf.PdfSanitizeFilterOptions_internal(self, *args)
29543: (4)              __swig_destroy__ = _mupdf.delete_PdfSanitizeFilterOptions
29544: (4)              opaque = property(_mupdf.PdfSanitizeFilterOptions_opaque_get,
_mupdf.PdfSanitizeFilterOptions_opaque_set)
29545: (4)              image_filter = property(_mupdf.PdfSanitizeFilterOptions_image_filter_get,
_mupdf.PdfSanitizeFilterOptions_image_filter_set)
29546: (4)              text_filter = property(_mupdf.PdfSanitizeFilterOptions_text_filter_get,
_mupdf.PdfSanitizeFilterOptions_text_filter_set)
29547: (4)              after_text_object =
property(_mupdf.PdfSanitizeFilterOptions_after_text_object_get,
_mupdf.PdfSanitizeFilterOptions_after_text_object_set)
29548: (4)              culler = property(_mupdf.PdfSanitizeFilterOptions_culler_get,
_mupdf.PdfSanitizeFilterOptions_culler_set)
29549: (4)              s_num_instances =
property(_mupdf.PdfSanitizeFilterOptions_s_num_instances_get,
_mupdf.PdfSanitizeFilterOptions_s_num_instances_set)
29550: (4)              def to_string(self):
29551: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
29552: (8)                  return _mupdf.PdfSanitizeFilterOptions_to_string(self)
29553: (4)              def __eq__(self, rhs):
29554: (8)                  r""" Comparison method."""
29555: (8)                  return _mupdf.PdfSanitizeFilterOptions___eq__(self, rhs)
29556: (4)              def __ne__(self, rhs):
29557: (8)                  r""" Comparison method."""
29558: (8)                  return _mupdf.PdfSanitizeFilterOptions___ne__(self, rhs)
29559: (0)          _mupdf.PdfSanitizeFilterOptions_swigregister(PdfSanitizeFilterOptions)
29560: (0)          class PdfSanitizeFilterOptions2(PdfSanitizeFilterOptions):
29561: (4)              r""" Wrapper class for struct pdf_sanitize_filter_options with virtual fns
for each fnptr; this is for use as a SWIG Director class."""
29562: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29563: (4)              __repr__ = _swig_repr
29564: (4)              def __init__(self):
29565: (8)                  r""" == Constructor."""
29566: (8)                  if self.__class__ == PdfSanitizeFilterOptions2:
29567: (12)                     _self = None
29568: (8)                  else:
29569: (12)                     _self = self
29570: (8)                  _mupdf.PdfSanitizeFilterOptions2_swiginit(self,
_mupdf.new_PdfSanitizeFilterOptions2(_self, ))
29571: (4)              def use_virtual_image_filter(self, use=True):
29572: (8)                  r"""
29573: (8)                  These methods set the function pointers in *m_internal
29574: (8)                  to point to internal callbacks that call our virtual methods.
29575: (8)                  """
29576: (8)                  return _mupdf.PdfSanitizeFilterOptions2_use_virtual_image_filter(self,
use)
29577: (4)              def use_virtual_text_filter(self, use=True):
29578: (8)                  return _mupdf.PdfSanitizeFilterOptions2_use_virtual_text_filter(self,
use)
29579: (4)              def use_virtual_after_text_object(self, use=True):
29580: (8)                  return
_mupdf.PdfSanitizeFilterOptions2_use_virtual_after_text_object(self, use)
29581: (4)              def use_virtual_culler(self, use=True):
29582: (8)                  return _mupdf.PdfSanitizeFilterOptions2_use_virtual_culler(self, use)
29583: (4)              def image_filter(self, arg_0, arg_2, arg_3, arg_4, arg_5):
29584: (8)                  r""" Default virtual method implementations; these all throw an
exception."""
29585: (8)                  return _mupdf.PdfSanitizeFilterOptions2_image_filter(self, arg_0,
arg_2, arg_3, arg_4, arg_5)
29586: (4)              def text_filter(self, arg_0, arg_2, arg_3, arg_4, arg_5, arg_6):
29587: (8)                  return _mupdf.PdfSanitizeFilterOptions2_text_filter(self, arg_0,
arg_2, arg_3, arg_4, arg_5, arg_6)
```

```
29588: (4)              def after_text_object(self, arg_0, arg_2, arg_3, arg_4):
29589: (8)                  return _mupdf.PdfSanitizeFilterOptions2_after_text_object(self, arg_0,
arg_2, arg_3, arg_4)
29590: (4)              def culler(self, arg_0, arg_2, arg_3):
29591: (8)                  return _mupdf.PdfSanitizeFilterOptions2_culler(self, arg_0, arg_2,
arg_3)
29592: (4)              __swig_destroy__ = _mupdf.delete_PdfSanitizeFilterOptions2
29593: (4)              def __disown__(self):
29594: (8)                  self.this.disown()
29595: (8)                  _mupdf.disown_PdfSanitizeFilterOptions2(self)
29596: (8)                  return weakref.proxy(self)
29597: (0)          _mupdf.PdfSanitizeFilterOptions2_swigregister(PdfSanitizeFilterOptions2)
29598: (0)          class PdfTextObjectState(object):
29599: (4)              r""" Wrapper class for struct `pdf_text_object_state`. Not copyable or
assignable."""
29600: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29601: (4)              __repr__ = _swig_repr
29602: (4)              def pdf_tos_get_text(self):
29603: (8)                  r""" Class-aware wrapper for `::pdf_tos_get_text()`."""
29604: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_get_text(self)
29605: (4)              def pdf_tos_make_trm(self, text, fontdesc, cid, trm):
29606: (8)                  r""" Class-aware wrapper for `::pdf_tos_make_trm()`."""
29607: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_make_trm(self, text,
fontdesc, cid, trm)
29608: (4)              def pdf_tos_move_after_char(self):
29609: (8)                  r""" Class-aware wrapper for `::pdf_tos_move_after_char()`."""
29610: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_move_after_char(self)
29611: (4)              def pdf_tos_newline(self, leading):
29612: (8)                  r""" Class-aware wrapper for `::pdf_tos_newline()`."""
29613: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_newline(self, leading)
29614: (4)              def pdf_tos_reset(self, render):
29615: (8)                  r""" Class-aware wrapper for `::pdf_tos_reset()`."""
29616: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_reset(self, render)
29617: (4)              def pdf_tos_set_matrix(self, a, b, c, d, e, f):
29618: (8)                  r""" Class-aware wrapper for `::pdf_tos_set_matrix()`."""
29619: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_set_matrix(self, a, b, c, d,
e, f)
29620: (4)              def pdf_tos_translate(self, tx, ty):
29621: (8)                  r""" Class-aware wrapper for `::pdf_tos_translate()`."""
29622: (8)                  return _mupdf.PdfTextObjectState_pdf_tos_translate(self, tx, ty)
29623: (4)              def __init__(self, *args):
29624: (8)                  r"""
29625: (8)                  *Overload 1:*
29626: (8)                  Default constructor, sets `m_internal` to null.
29627: (8)                  |
29628: (8)                  *Overload 2:*
29629: (8)                  Constructor using raw copy of pre-existing `::pdf_text_object_state`.
29630: (8)                  """
29631: (8)                  _mupdf.PdfTextObjectState_swiginit(self,
_mupdf.new_PdfTextObjectState(*args))
29632: (4)              __swig_destroy__ = _mupdf.delete_PdfTextObjectState
29633: (4)              def m_internal_value(self):
29634: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29635: (8)                  return _mupdf.PdfTextObjectState_m_internal_value(self)
29636: (4)              m_internal = property(_mupdf.PdfTextObjectState_m_internal_get,
_mupdf.PdfTextObjectState_m_internal_set, doc=r""" Pointer to wrapped data.""")
29637: (4)              s_num_instances = property(_mupdf.PdfTextObjectState_s_num_instances_get,
_mupdf.PdfTextObjectState_s_num_instances_set)
29638: (0)          _mupdf.PdfTextObjectState_swigregister(PdfTextObjectState)
29639: (0)          class PdfTextState(object):
29640: (4)              r""" Wrapper class for struct `pdf_text_state`. Not copyable or
assignable."""
29641: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29642: (4)              __repr__ = _swig_repr
29643: (4)              def __init__(self, *args):
29644: (8)                  r"""
```

```
29645: (8)                  *Overload 1:*
29646: (8)                  Default constructor, sets `m_internal` to null.
29647: (8)                  |
29648: (8)                  *Overload 2:*
29649: (8)                  Constructor using raw copy of pre-existing `::pdf_text_state`.
29650: (8)                  """
29651: (8)                  _mupdf.PdfTextState_swiginit(self, _mupdf.new_PdfTextState(*args))
29652: (4)              __swig_destroy__ = _mupdf.delete_PdfTextState
29653: (4)              def m_internal_value(self):
29654: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29655: (8)                  return _mupdf.PdfTextState_m_internal_value(self)
29656: (4)              m_internal = property(_mupdf.PdfTextState_m_internal_get,
_mupdf.PdfTextState_m_internal_set, doc=r""" Pointer to wrapped data.""")
29657: (4)              s_num_instances = property(_mupdf.PdfTextState_s_num_instances_get,
_mupdf.PdfTextState_s_num_instances_set)
29658: (0)          _mupdf.PdfTextState_swigregister(PdfTextState)
29659: (0)          class PdfUnsavedSig(object):
29660: (4)              r""" Wrapper class for struct `pdf_unsaved_sig`. Not copyable or
assignable."""
29661: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29662: (4)              __repr__ = _swig_repr
29663: (4)              def __init__(self, *args):
29664: (8)                  r"""
29665: (8)                  *Overload 1:*
29666: (8)                  Default constructor, sets `m_internal` to null.
29667: (8)                  |
29668: (8)                  *Overload 2:*
29669: (8)                  Constructor using raw copy of pre-existing `::pdf_unsaved_sig`.
29670: (8)                  """
29671: (8)                  _mupdf.PdfUnsavedSig_swiginit(self, _mupdf.new_PdfUnsavedSig(*args))
29672: (4)              __swig_destroy__ = _mupdf.delete_PdfUnsavedSig
29673: (4)              def m_internal_value(self):
29674: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29675: (8)                  return _mupdf.PdfUnsavedSig_m_internal_value(self)
29676: (4)              m_internal = property(_mupdf.PdfUnsavedSig_m_internal_get,
_mupdf.PdfUnsavedSig_m_internal_set, doc=r""" Pointer to wrapped data.""")
29677: (4)              s_num_instances = property(_mupdf.PdfUnsavedSig_s_num_instances_get,
_mupdf.PdfUnsavedSig_s_num_instances_set)
29678: (0)          _mupdf.PdfUnsavedSig_swigregister(PdfUnsavedSig)
29679: (0)          class PdfVmtx(object):
29680: (4)              r""" Wrapper class for struct `pdf_vmtx`. Not copyable or assignable."""
29681: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29682: (4)              __repr__ = _swig_repr
29683: (4)              def __init__(self, *args):
29684: (8)                  r"""
29685: (8)                  *Overload 1:*
29686: (8)                  Default constructor, sets `m_internal` to null.
29687: (8)                  |
29688: (8)                  *Overload 2:*
29689: (8)                  Constructor using raw copy of pre-existing `::pdf_vmtx`.
29690: (8)                  """
29691: (8)                  _mupdf.PdfVmtx_swiginit(self, _mupdf.new_PdfVmtx(*args))
29692: (4)              __swig_destroy__ = _mupdf.delete_PdfVmtx
29693: (4)              def m_internal_value(self):
29694: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29695: (8)                  return _mupdf.PdfVmtx_m_internal_value(self)
29696: (4)              m_internal = property(_mupdf.PdfVmtx_m_internal_get,
_mupdf.PdfVmtx_m_internal_set, doc=r""" Pointer to wrapped data.""")
29697: (4)              s_num_instances = property(_mupdf.PdfVmtx_s_num_instances_get,
_mupdf.PdfVmtx_s_num_instances_set)
29698: (0)          _mupdf.PdfVmtx_swigregister(PdfVmtx)
29699: (0)          class PdfWriteOptions(object):
29700: (4)              r""" Wrapper class for struct `pdf_write_options`."""
29701: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
29702: (4)              __repr__ = _swig_repr
29703: (4)              def pdf_parse_write_options(self, args):
29704: (8)                  r""" We use default copy constructor and operator=.  Class-aware
wrapper for `::pdf_parse_write_options()`."""
29705: (8)                  return _mupdf.PdfWriteOptions_pdf_parse_write_options(self, args)
29706: (4)              def opwd_utf8_set_value(self, text):
29707: (8)                  r""" Copies <text> into opwd_utf8[]."""
29708: (8)                  return _mupdf.PdfWriteOptions_opwd_utf8_set_value(self, text)
29709: (4)              def upwd_utf8_set_value(self, text):
29710: (8)                  r""" Copies <text> into upwd_utf8[]."""
29711: (8)                  return _mupdf.PdfWriteOptions_upwd_utf8_set_value(self, text)
29712: (4)              def __init__(self, *args):
29713: (8)                  r"""
29714: (8)                  *Overload 1:*
29715: (8)                  Default constructor, makes copy of pdf_default_write_options.
29716: (8)                  |
29717: (8)                  *Overload 2:*
29718: (8)                  Copy constructor using raw memcopy().
29719: (8)                  |
29720: (8)                  *Overload 3:*
29721: (8)                  Constructor using raw copy of pre-existing `::pdf_write_options`.
29722: (8)                  |
29723: (8)                  *Overload 4:*
29724: (8)                  Constructor using raw copy of pre-existing `::pdf_write_options`.
29725: (8)                  """
29726: (8)                  _mupdf.PdfWriteOptions_swiginit(self,
_mupdf.new_PdfWriteOptions(*args))
29727: (4)              def internal(self, *args):
29728: (8)                  r"""
29729: (8)                  *Overload 1:*
29730: (8)                  Access as underlying struct.
29731: (8)                  |
29732: (8)                  *Overload 2:*
29733: (8)                  Access as underlying struct.
29734: (8)                  """
29735: (8)                  return _mupdf.PdfWriteOptions_internal(self, *args)
29736: (4)              __swig_destroy__ = _mupdf.delete_PdfWriteOptions
29737: (4)              do_incremental = property(_mupdf.PdfWriteOptions_do_incremental_get,
_mupdf.PdfWriteOptions_do_incremental_set)
29738: (4)              do_pretty = property(_mupdf.PdfWriteOptions_do_pretty_get,
_mupdf.PdfWriteOptions_do_pretty_set)
29739: (4)              do_ascii = property(_mupdf.PdfWriteOptions_do_ascii_get,
_mupdf.PdfWriteOptions_do_ascii_set)
29740: (4)              do_compress = property(_mupdf.PdfWriteOptions_do_compress_get,
_mupdf.PdfWriteOptions_do_compress_set)
29741: (4)              do_compress_images =
property(_mupdf.PdfWriteOptions_do_compress_images_get,
_mupdf.PdfWriteOptions_do_compress_images_set)
29742: (4)              do_compress_fonts = property(_mupdf.PdfWriteOptions_do_compress_fonts_get,
_mupdf.PdfWriteOptions_do_compress_fonts_set)
29743: (4)              do_decompress = property(_mupdf.PdfWriteOptions_do_decompress_get,
_mupdf.PdfWriteOptions_do_decompress_set)
29744: (4)              do_garbage = property(_mupdf.PdfWriteOptions_do_garbage_get,
_mupdf.PdfWriteOptions_do_garbage_set)
29745: (4)              do_linear = property(_mupdf.PdfWriteOptions_do_linear_get,
_mupdf.PdfWriteOptions_do_linear_set)
29746: (4)              do_clean = property(_mupdf.PdfWriteOptions_do_clean_get,
_mupdf.PdfWriteOptions_do_clean_set)
29747: (4)              do_sanitize = property(_mupdf.PdfWriteOptions_do_sanitize_get,
_mupdf.PdfWriteOptions_do_sanitize_set)
29748: (4)              do_appearance = property(_mupdf.PdfWriteOptions_do_appearance_get,
_mupdf.PdfWriteOptions_do_appearance_set)
29749: (4)              do_encrypt = property(_mupdf.PdfWriteOptions_do_encrypt_get,
_mupdf.PdfWriteOptions_do_encrypt_set)
29750: (4)              dont_regenerate_id =
property(_mupdf.PdfWriteOptions_dont_regenerate_id_get,
_mupdf.PdfWriteOptions_dont_regenerate_id_set)
29751: (4)              permissions = property(_mupdf.PdfWriteOptions_permissions_get,
```

```
_mupdf.PdfWriteOptions_permissions_set)
29752: (4)              opwd_utf8 = property(_mupdf.PdfWriteOptions_opwd_utf8_get,
_mupdf.PdfWriteOptions_opwd_utf8_set)
29753: (4)              upwd_utf8 = property(_mupdf.PdfWriteOptions_upwd_utf8_get,
_mupdf.PdfWriteOptions_upwd_utf8_set)
29754: (4)              do_snapshot = property(_mupdf.PdfWriteOptions_do_snapshot_get,
_mupdf.PdfWriteOptions_do_snapshot_set)
29755: (4)              do_preserve_metadata =
property(_mupdf.PdfWriteOptions_do_preserve_metadata_get,
_mupdf.PdfWriteOptions_do_preserve_metadata_set)
29756: (4)              do_use_objstms = property(_mupdf.PdfWriteOptions_do_use_objstms_get,
_mupdf.PdfWriteOptions_do_use_objstms_set)
29757: (4)              compression_effort =
property(_mupdf.PdfWriteOptions_compression_effort_get,
_mupdf.PdfWriteOptions_compression_effort_set)
29758: (4)              s_num_instances = property(_mupdf.PdfWriteOptions_s_num_instances_get,
_mupdf.PdfWriteOptions_s_num_instances_set)
29759: (4)              def to_string(self):
29760: (8)                  r""" Returns string containing our members, labelled and inside (...),
using operator<<."""
29761: (8)                  return _mupdf.PdfWriteOptions_to_string(self)
29762: (4)              def __eq__(self, rhs):
29763: (8)                  r""" Comparison method."""
29764: (8)                  return _mupdf.PdfWriteOptions___eq__(self, rhs)
29765: (4)              def __ne__(self, rhs):
29766: (8)                  r""" Comparison method."""
29767: (8)                  return _mupdf.PdfWriteOptions___ne__(self, rhs)
29768: (0)          _mupdf.PdfWriteOptions_swigregister(PdfWriteOptions)
29769: (0)          class PdfXrange(object):
29770: (4)              r""" Wrapper class for struct `pdf_xrange`. Not copyable or assignable."""
29771: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29772: (4)              __repr__ = _swig_repr
29773: (4)              def __init__(self, *args):
29774: (8)                  r"""
29775: (8)                  *Overload 1:*
29776: (8)                  Default constructor, sets `m_internal` to null.
29777: (8)                  |
29778: (8)                  *Overload 2:*
29779: (8)                  Constructor using raw copy of pre-existing `::pdf_xrange`.
29780: (8)                  """
29781: (8)                  _mupdf.PdfXrange_swiginit(self, _mupdf.new_PdfXrange(*args))
29782: (4)              __swig_destroy__ = _mupdf.delete_PdfXrange
29783: (4)              def m_internal_value(self):
29784: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29785: (8)                  return _mupdf.PdfXrange_m_internal_value(self)
29786: (4)              m_internal = property(_mupdf.PdfXrange_m_internal_get,
_mupdf.PdfXrange_m_internal_set, doc=r""" Pointer to wrapped data.""")
29787: (4)              s_num_instances = property(_mupdf.PdfXrange_s_num_instances_get,
_mupdf.PdfXrange_s_num_instances_set)
29788: (0)          _mupdf.PdfXrange_swigregister(PdfXrange)
29789: (0)          class PdfXref(object):
29790: (4)              r""" Wrapper class for struct `pdf_xref`. Not copyable or assignable."""
29791: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29792: (4)              __repr__ = _swig_repr
29793: (4)              def __init__(self, *args):
29794: (8)                  r"""
29795: (8)                  *Overload 1:*
29796: (8)                  == Constructors.  Constructor using `pdf_new_local_xref()`.
29797: (8)                  |
29798: (8)                  *Overload 2:*
29799: (8)                  Default constructor, sets `m_internal` to null.
29800: (8)                  |
29801: (8)                  *Overload 3:*
29802: (8)                  Constructor using raw copy of pre-existing `::pdf_xref`.
29803: (8)                  """
29804: (8)                  _mupdf.PdfXref_swiginit(self, _mupdf.new_PdfXref(*args))
```

```
29805: (4)              __swig_destroy__ = _mupdf.delete_PdfXref
29806: (4)              def m_internal_value(self):
29807: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29808: (8)                  return _mupdf.PdfXref_m_internal_value(self)
29809: (4)              m_internal = property(_mupdf.PdfXref_m_internal_get,
_mupdf.PdfXref_m_internal_set, doc=r""" Pointer to wrapped data.""")
29810: (4)              s_num_instances = property(_mupdf.PdfXref_s_num_instances_get,
_mupdf.PdfXref_s_num_instances_set)
29811: (0)          _mupdf.PdfXref_swigregister(PdfXref)
29812: (0)          class PdfXrefEntry(object):
29813: (4)              r""" Wrapper class for struct `pdf_xref_entry`. Not copyable or
assignable."""
29814: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29815: (4)              __repr__ = _swig_repr
29816: (4)              def __init__(self, *args):
29817: (8)                  r"""
29818: (8)                  *Overload 1:*
29819: (8)                  Default constructor, sets `m_internal` to null.
29820: (8)                  |
29821: (8)                  *Overload 2:*
29822: (8)                  Constructor using raw copy of pre-existing `::pdf_xref_entry`.
29823: (8)                  """
29824: (8)                  _mupdf.PdfXrefEntry_swiginit(self, _mupdf.new_PdfXrefEntry(*args))
29825: (4)              __swig_destroy__ = _mupdf.delete_PdfXrefEntry
29826: (4)              def m_internal_value(self):
29827: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29828: (8)                  return _mupdf.PdfXrefEntry_m_internal_value(self)
29829: (4)              m_internal = property(_mupdf.PdfXrefEntry_m_internal_get,
_mupdf.PdfXrefEntry_m_internal_set, doc=r""" Pointer to wrapped data.""")
29830: (4)              s_num_instances = property(_mupdf.PdfXrefEntry_s_num_instances_get,
_mupdf.PdfXrefEntry_s_num_instances_set)
29831: (0)          _mupdf.PdfXrefEntry_swigregister(PdfXrefEntry)
29832: (0)          class PdfXrefSubsec(object):
29833: (4)              r""" Wrapper class for struct `pdf_xref_subsec`. Not copyable or
assignable."""
29834: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29835: (4)              __repr__ = _swig_repr
29836: (4)              def __init__(self, *args):
29837: (8)                  r"""
29838: (8)                  *Overload 1:*
29839: (8)                  Default constructor, sets `m_internal` to null.
29840: (8)                  |
29841: (8)                  *Overload 2:*
29842: (8)                  Constructor using raw copy of pre-existing `::pdf_xref_subsec`.
29843: (8)                  """
29844: (8)                  _mupdf.PdfXrefSubsec_swiginit(self, _mupdf.new_PdfXrefSubsec(*args))
29845: (4)              __swig_destroy__ = _mupdf.delete_PdfXrefSubsec
29846: (4)              def m_internal_value(self):
29847: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29848: (8)                  return _mupdf.PdfXrefSubsec_m_internal_value(self)
29849: (4)              m_internal = property(_mupdf.PdfXrefSubsec_m_internal_get,
_mupdf.PdfXrefSubsec_m_internal_set, doc=r""" Pointer to wrapped data.""")
29850: (4)              s_num_instances = property(_mupdf.PdfXrefSubsec_s_num_instances_get,
_mupdf.PdfXrefSubsec_s_num_instances_set)
29851: (0)          _mupdf.PdfXrefSubsec_swigregister(PdfXrefSubsec)
29852: (0)          def num_instances():
29853: (4)              r""" Returns map from class name (for example FzDocument) to
s_num_instances."""
29854: (4)              return _mupdf.num_instances()
29855: (0)          class FzLinkIterator(object):
29856: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29857: (4)              __repr__ = _swig_repr
29858: (4)              def __init__(self, item):
```

```
29859: (8)                  _mupdf.FzLinkIterator_swiginit(self, _mupdf.new_FzLinkIterator(item))
29860: (4)              def __increment__(self):
29861: (8)                  return _mupdf.FzLinkIterator___increment__(self)
29862: (4)              def __eq__(self, rhs):
29863: (8)                  return _mupdf.FzLinkIterator___eq__(self, rhs)
29864: (4)              def __ne__(self, rhs):
29865: (8)                  return _mupdf.FzLinkIterator___ne__(self, rhs)
29866: (4)              def __ref__(self):
29867: (8)                  return _mupdf.FzLinkIterator___ref__(self)
29868: (4)              def __deref__(self):
29869: (8)                  return _mupdf.FzLinkIterator___deref__(self)
29870: (4)              __swig_destroy__ = _mupdf.delete_FzLinkIterator
29871: (4)              def fz_set_link_rect(self, rect):
29872: (8)                  r""" Class-aware wrapper for `::fz_set_link_rect()`."""
29873: (8)                  return _mupdf.FzLinkIterator_fz_set_link_rect(self, rect)
29874: (4)              def fz_set_link_uri(self, uri):
29875: (8)                  r""" Class-aware wrapper for `::fz_set_link_uri()`."""
29876: (8)                  return _mupdf.FzLinkIterator_fz_set_link_uri(self, uri)
29877: (4)              def begin(self):
29878: (8)                  r""" Used for iteration over linked list of FzLink items starting at
fz_link::."""
29879: (8)                  return _mupdf.FzLinkIterator_begin(self)
29880: (4)              def end(self):
29881: (8)                  r""" Used for iteration over linked list of FzLink items starting at
fz_link::."""
29882: (8)                  return _mupdf.FzLinkIterator_end(self)
29883: (4)              def refs(self):
29884: (8)                  return _mupdf.FzLinkIterator_refs(self)
29885: (4)              def next(self):
29886: (8)                  return _mupdf.FzLinkIterator_next(self)
29887: (4)              def rect(self):
29888: (8)                  return _mupdf.FzLinkIterator_rect(self)
29889: (4)              def uri(self):
29890: (8)                  return _mupdf.FzLinkIterator_uri(self)
29891: (4)              def m_internal_value(self):
29892: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29893: (8)                  return _mupdf.FzLinkIterator_m_internal_value(self)
29894: (4)              m_internal = property(_mupdf.FzLinkIterator_m_internal_get,
_mupdf.FzLinkIterator_m_internal_set, doc=r""" Pointer to wrapped data.""")
29895: (4)              s_num_instances = property(_mupdf.FzLinkIterator_s_num_instances_get,
_mupdf.FzLinkIterator_s_num_instances_set)
29896: (0)          _mupdf.FzLinkIterator_swigregister(FzLinkIterator)
29897: (0)          class FzStextBlockIterator(object):
29898: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29899: (4)              __repr__ = _swig_repr
29900: (4)              def __init__(self, item):
29901: (8)                  _mupdf.FzStextBlockIterator_swiginit(self,
_mupdf.new_FzStextBlockIterator(item))
29902: (4)              def __increment__(self):
29903: (8)                  return _mupdf.FzStextBlockIterator___increment__(self)
29904: (4)              def __eq__(self, rhs):
29905: (8)                  return _mupdf.FzStextBlockIterator___eq__(self, rhs)
29906: (4)              def __ne__(self, rhs):
29907: (8)                  return _mupdf.FzStextBlockIterator___ne__(self, rhs)
29908: (4)              def __ref__(self):
29909: (8)                  return _mupdf.FzStextBlockIterator___ref__(self)
29910: (4)              def __deref__(self):
29911: (8)                  return _mupdf.FzStextBlockIterator___deref__(self)
29912: (4)              __swig_destroy__ = _mupdf.delete_FzStextBlockIterator
29913: (4)              def begin(self):
29914: (8)                  r""" Used for iteration over linked list of FzStextChar items starting
at fz_stext_char::first_char."""
29915: (8)                  return _mupdf.FzStextBlockIterator_begin(self)
29916: (4)              def end(self):
29917: (8)                  r""" Used for iteration over linked list of FzStextChar items starting
at fz_stext_char::first_char."""
29918: (8)                  return _mupdf.FzStextBlockIterator_end(self)
```

```
29919: (4)              def m_internal_value(self):
29920: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29921: (8)                  return _mupdf.FzStextBlockIterator_m_internal_value(self)
29922: (4)              m_internal = property(_mupdf.FzStextBlockIterator_m_internal_get,
_mupdf.FzStextBlockIterator_m_internal_set, doc=r""" Pointer to wrapped data.""")
29923: (4)              s_num_instances =
property(_mupdf.FzStextBlockIterator_s_num_instances_get,
_mupdf.FzStextBlockIterator_s_num_instances_set)
29924: (0)          _mupdf.FzStextBlockIterator_swigregister(FzStextBlockIterator)
29925: (0)          class FzStextLineIterator(object):
29926: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29927: (4)              __repr__ = _swig_repr
29928: (4)              def __init__(self, item):
29929: (8)                  _mupdf.FzStextLineIterator_swiginit(self,
_mupdf.new_FzStextLineIterator(item))
29930: (4)              def __increment__(self):
29931: (8)                  return _mupdf.FzStextLineIterator___increment__(self)
29932: (4)              def __eq__(self, rhs):
29933: (8)                  return _mupdf.FzStextLineIterator___eq__(self, rhs)
29934: (4)              def __ne__(self, rhs):
29935: (8)                  return _mupdf.FzStextLineIterator___ne__(self, rhs)
29936: (4)              def __ref__(self):
29937: (8)                  return _mupdf.FzStextLineIterator___ref__(self)
29938: (4)              def __deref__(self):
29939: (8)                  return _mupdf.FzStextLineIterator___deref__(self)
29940: (4)              __swig_destroy__ = _mupdf.delete_FzStextLineIterator
29941: (4)              def m_internal_value(self):
29942: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29943: (8)                  return _mupdf.FzStextLineIterator_m_internal_value(self)
29944: (4)              m_internal = property(_mupdf.FzStextLineIterator_m_internal_get,
_mupdf.FzStextLineIterator_m_internal_set, doc=r""" Pointer to wrapped data.""")
29945: (4)              s_num_instances = property(_mupdf.FzStextLineIterator_s_num_instances_get,
_mupdf.FzStextLineIterator_s_num_instances_set)
29946: (0)          _mupdf.FzStextLineIterator_swigregister(FzStextLineIterator)
29947: (0)          class FzStextPageIterator(object):
29948: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
29949: (4)              __repr__ = _swig_repr
29950: (4)              def __init__(self, item):
29951: (8)                  _mupdf.FzStextPageIterator_swiginit(self,
_mupdf.new_FzStextPageIterator(item))
29952: (4)              def __increment__(self):
29953: (8)                  return _mupdf.FzStextPageIterator___increment__(self)
29954: (4)              def __eq__(self, rhs):
29955: (8)                  return _mupdf.FzStextPageIterator___eq__(self, rhs)
29956: (4)              def __ne__(self, rhs):
29957: (8)                  return _mupdf.FzStextPageIterator___ne__(self, rhs)
29958: (4)              def __ref__(self):
29959: (8)                  return _mupdf.FzStextPageIterator___ref__(self)
29960: (4)              def __deref__(self):
29961: (8)                  return _mupdf.FzStextPageIterator___deref__(self)
29962: (4)              __swig_destroy__ = _mupdf.delete_FzStextPageIterator
29963: (4)              def i_transform(self):
29964: (8)                  r""" Returns m_internal.u.i.transform if m_internal->type is
FZ_STEXT_BLOCK_IMAGE, else throws."""
29965: (8)                  return _mupdf.FzStextPageIterator_i_transform(self)
29966: (4)              def i_image(self):
29967: (8)                  r""" Returns m_internal.u.i.image if m_internal->type is
FZ_STEXT_BLOCK_IMAGE, else throws."""
29968: (8)                  return _mupdf.FzStextPageIterator_i_image(self)
29969: (4)              def begin(self):
29970: (8)                  r""" Used for iteration over linked list of FzStextLine items starting
at fz_stext_line::u.t.first_line."""
29971: (8)                  return _mupdf.FzStextPageIterator_begin(self)
29972: (4)              def end(self):
29973: (8)                  r""" Used for iteration over linked list of FzStextLine items starting
```

```
at fz_stext_line::u.t.first_line."""
29974: (8)                  return _mupdf.FzStextPageIterator_end(self)
29975: (4)              def m_internal_value(self):
29976: (8)                  r""" Return numerical value of .m_internal; helps with Python
debugging."""
29977: (8)                  return _mupdf.FzStextPageIterator_m_internal_value(self)
29978: (4)              m_internal = property(_mupdf.FzStextPageIterator_m_internal_get,
_mupdf.FzStextPageIterator_m_internal_set, doc=r""" Pointer to wrapped data.""")
29979: (4)              s_num_instances = property(_mupdf.FzStextPageIterator_s_num_instances_get,
_mupdf.FzStextPageIterator_s_num_instances_set)
29980: (0)          _mupdf.FzStextPageIterator_swigregister(FzStextPageIterator)
29981: (0)          def fz_aa_level():
29982: (4)              r"""
29983: (4)              Class-aware wrapper for `::fz_aa_level()`.
29984: (8)                  Get the number of bits of antialiasing we are
29985: (8)                  using (for graphics). Between 0 and 8.
29986: (4)              """
29987: (4)              return _mupdf.fz_aa_level()
29988: (0)          def fz_abs(f):
29989: (4)              r"""
29990: (4)              Class-aware wrapper for `::fz_abs()`.
29991: (8)                  Some standard math functions, done as static inlines for speed.
29992: (8)                  People with compilers that do not adequately implement inline
29993: (8)                  may like to reimplement these using macros.
29994: (4)              """
29995: (4)              return _mupdf.fz_abs(f)
29996: (0)          def fz_absi(i):
29997: (4)              r""" Class-aware wrapper for `::fz_absi()`."""
29998: (4)              return _mupdf.fz_absi(i)
29999: (0)          def fz_add_layout_char(block, x, w, p):
30000: (4)              r"""
30001: (4)              Class-aware wrapper for `::fz_add_layout_char()`.
30002: (8)                  Add a new char to the line at the end of the layout block.
30003: (4)              """
30004: (4)              return _mupdf.fz_add_layout_char(block, x, w, p)
30005: (0)          def fz_add_layout_line(block, x, y, h, p):
30006: (4)              r"""
30007: (4)              Class-aware wrapper for `::fz_add_layout_line()`.
30008: (8)                  Add a new line to the end of the layout block.
30009: (4)              """
30010: (4)              return _mupdf.fz_add_layout_line(block, x, y, h, p)
30011: (0)          def fz_add_separation(sep, name, cs, cs_channel):
30012: (4)              r"""
30013: (4)              Class-aware wrapper for `::fz_add_separation()`.
30014: (8)                  Add a separation (null terminated name, colorspace)
30015: (4)              """
30016: (4)              return _mupdf.fz_add_separation(sep, name, cs, cs_channel)
30017: (0)          def fz_add_separation_equivalents(sep, rgba, cmyk, name):
30018: (4)              r"""
30019: (4)              Class-aware wrapper for `::fz_add_separation_equivalents()`.
30020: (8)                  Add a separation with equivalents (null terminated name,
30021: (8)                  colorspace)
30022: (8)                  (old, deprecated)
30023: (4)              """
30024: (4)              return _mupdf.fz_add_separation_equivalents(sep, rgba, cmyk, name)
30025: (0)          def fz_adjust_rect_for_stroke(rect, stroke, ctm):
30026: (4)              r"""
30027: (4)              Class-aware wrapper for `::fz_adjust_rect_for_stroke()`.
30028: (8)                  Given a rectangle (assumed to be the bounding box for a path),
30029: (8)                  expand it to allow for the expansion of the bbox that would be
30030: (8)                  seen by stroking the path with the given stroke state and
30031: (8)                  transform.
30032: (4)              """
30033: (4)              return _mupdf.fz_adjust_rect_for_stroke(rect, stroke, ctm)
30034: (0)          def fz_advance_glyph(font, glyph, wmode):
30035: (4)              r"""
30036: (4)              Class-aware wrapper for `::fz_advance_glyph()`.
30037: (8)                  Return the advance for a given glyph.
30038: (8)                  font: The font to look for the glyph in.
```

```
30039: (8)                 glyph: The glyph to find the advance for.
30040: (8)                 wmode: 1 for vertical mode, 0 for horizontal.
30041: (8)                 Returns the advance for the glyph.
30042: (4)             """
30043: (4)             return _mupdf.fz_advance_glyph(font, glyph, wmode)
30044: (0)         def fz_aes_crypt_cbc(ctx, mode, length, iv, input, output):
30045: (4)             r"""
30046: (4)             Class-aware wrapper for `::fz_aes_crypt_cbc()`.
30047: (8)                 AES block processing. Encrypts or Decrypts (according to mode,
30048: (8)                 which must match what was initially set up) length bytes (which
30049: (8)                 must be a multiple of 16), using (and modifying) the insertion
30050: (8)                 vector iv, reading from input, and writing to output.
30051: (8)                 Never throws an exception.
30052: (4)             """
30053: (4)             return _mupdf.fz_aes_crypt_cbc(ctx, mode, length, iv, input, output)
30054: (0)         def fz_aes_setkey_dec(ctx, key, keysize):
30055: (4)             r"""
30056: (4)             Class-aware wrapper for `::fz_aes_setkey_dec()`.
30057: (8)                 AES decryption intialisation. Fills in the supplied context
30058: (8)                 and prepares for decryption using the given key.
30059: (8)                 Returns non-zero for error (key size other than 128/192/256).
30060: (8)                 Never throws an exception.
30061: (4)             """
30062: (4)             return _mupdf.fz_aes_setkey_dec(ctx, key, keysize)
30063: (0)         def fz_aes_setkey_enc(ctx, key, keysize):
30064: (4)             r"""
30065: (4)             Class-aware wrapper for `::fz_aes_setkey_enc()`.
30066: (8)                 AES encryption intialisation. Fills in the supplied context
30067: (8)                 and prepares for encryption using the given key.
30068: (8)                 Returns non-zero for error (key size other than 128/192/256).
30069: (8)                 Never throws an exception.
30070: (4)             """
30071: (4)             return _mupdf.fz_aes_setkey_enc(ctx, key, keysize)
30072: (0)         def fz_alpha_from_gray(gray):
30073: (4)             r""" Class-aware wrapper for `::fz_alpha_from_gray()`."""
30074: (4)             return _mupdf.fz_alpha_from_gray(gray)
30075: (0)         def fz_append_base64(out, data, size, newline):
30076: (4)             r"""
30077: (4)             Class-aware wrapper for `::fz_append_base64()`.
30078: (8)                 Write a base64 encoded data block, optionally with periodic newlines.
30079: (4)             """
30080: (4)             return _mupdf.fz_append_base64(out, data, size, newline)
30081: (0)         def fz_append_base64_buffer(out, data, newline):
30082: (4)             r"""
30083: (4)             Class-aware wrapper for `::fz_append_base64_buffer()`.
30084: (8)                 Append a base64 encoded fz_buffer, optionally with periodic newlines.
30085: (4)             """
30086: (4)             return _mupdf.fz_append_base64_buffer(out, data, newline)
30087: (0)         def fz_append_bits(buf, value, count):
30088: (4)             r""" Class-aware wrapper for `::fz_append_bits()`."""
30089: (4)             return _mupdf.fz_append_bits(buf, value, count)
30090: (0)         def fz_append_bits_pad(buf):
30091: (4)             r""" Class-aware wrapper for `::fz_append_bits_pad()`."""
30092: (4)             return _mupdf.fz_append_bits_pad(buf)
30093: (0)         def fz_append_buffer(destination, source):
30094: (4)             r"""
30095: (4)             Class-aware wrapper for `::fz_append_buffer()`.
30096: (8)                 Append the contents of the source buffer onto the end of the
30097: (8)                 destination buffer, extending automatically as required.
30098: (8)                 Ownership of buffers does not change.
30099: (4)             """
30100: (4)             return _mupdf.fz_append_buffer(destination, source)
30101: (0)         def fz_append_byte(buf, c):
30102: (4)             r""" Class-aware wrapper for `::fz_append_byte()`."""
30103: (4)             return _mupdf.fz_append_byte(buf, c)
30104: (0)         def fz_append_data(buf, data, len):
30105: (4)             r"""
30106: (4)             Class-aware wrapper for `::fz_append_data()`.
30107: (8)                 fz_append_*: Append data to a buffer.
```

```
30108: (8)                 The buffer will automatically grow as required.
30109: (4)             """
30110: (4)             return _mupdf.fz_append_data(buf, data, len)
30111: (0)         def fz_append_image_as_data_uri(out, image):
30112: (4)             r""" Class-aware wrapper for `::fz_append_image_as_data_uri()`."""
30113: (4)             return _mupdf.fz_append_image_as_data_uri(out, image)
30114: (0)         def fz_append_int16_be(buf, x):
30115: (4)             r""" Class-aware wrapper for `::fz_append_int16_be()`."""
30116: (4)             return _mupdf.fz_append_int16_be(buf, x)
30117: (0)         def fz_append_int16_le(buf, x):
30118: (4)             r""" Class-aware wrapper for `::fz_append_int16_le()`."""
30119: (4)             return _mupdf.fz_append_int16_le(buf, x)
30120: (0)         def fz_append_int32_be(buf, x):
30121: (4)             r""" Class-aware wrapper for `::fz_append_int32_be()`."""
30122: (4)             return _mupdf.fz_append_int32_be(buf, x)
30123: (0)         def fz_append_int32_le(buf, x):
30124: (4)             r""" Class-aware wrapper for `::fz_append_int32_le()`."""
30125: (4)             return _mupdf.fz_append_int32_le(buf, x)
30126: (0)         def fz_append_pdf_string(buffer, text):
30127: (4)             r"""
30128: (4)             Class-aware wrapper for `::fz_append_pdf_string()`.
30129: (8)                 fz_append_pdf_string: Append a string with PDF syntax quotes and
30130: (8)                 escapes.
30131: (8)                 The buffer will automatically grow as required.
30132: (4)             """
30133: (4)             return _mupdf.fz_append_pdf_string(buffer, text)
30134: (0)         def fz_append_pixmap_as_data_uri(out, pixmap):
30135: (4)             r""" Class-aware wrapper for `::fz_append_pixmap_as_data_uri()`."""
30136: (4)             return _mupdf.fz_append_pixmap_as_data_uri(out, pixmap)
30137: (0)         def fz_append_rune(buf, c):
30138: (4)             r""" Class-aware wrapper for `::fz_append_rune()`."""
30139: (4)             return _mupdf.fz_append_rune(buf, c)
30140: (0)         def fz_append_string(buf, data):
30141: (4)             r""" Class-aware wrapper for `::fz_append_string()`."""
30142: (4)             return _mupdf.fz_append_string(buf, data)
30143: (0)         def fz_arc4_encrypt(state, dest, src, len):
30144: (4)             r"""
30145: (4)             Class-aware wrapper for `::fz_arc4_encrypt()`.
30146: (8)                 RC4 block encrypt operation; encrypt src into dst (both of
30147: (8)                 length len) updating the RC4 state as we go.
30148: (8)                 Never throws an exception.
30149: (4)             """
30150: (4)             return _mupdf.fz_arc4_encrypt(state, dest, src, len)
30151: (0)         def fz_arc4_final(state):
30152: (4)             r"""
30153: (4)             Class-aware wrapper for `::fz_arc4_final()`.
30154: (8)                 RC4 finalization. Zero the context.
30155: (8)                 Never throws an exception.
30156: (4)             """
30157: (4)             return _mupdf.fz_arc4_final(state)
30158: (0)         def fz_arc4_init(state, key, len):
30159: (4)             r"""
30160: (4)             Class-aware wrapper for `::fz_arc4_init()`.
30161: (8)                 RC4 initialization. Begins an RC4 operation, writing a new
30162: (8)                 context.
30163: (8)                 Never throws an exception.
30164: (4)             """
30165: (4)             return _mupdf.fz_arc4_init(state, key, len)
30166: (0)         def fz_archive_format(arch):
30167: (4)             r"""
30168: (4)             Class-aware wrapper for `::fz_archive_format()`.
30169: (8)                 Return a pointer to a string describing the format of the
30170: (8)                 archive.
30171: (8)                 The lifetime of the string is unspecified (in current
30172: (8)                 implementations the string will persist until the archive
30173: (8)                 is closed, but this is not guaranteed).
30174: (4)             """
30175: (4)             return _mupdf.fz_archive_format(arch)
30176: (0)         def fz_atof(s):
```

```
30177: (4)              r"""
30178: (4)              Class-aware wrapper for `::fz_atof()`.
30179: (8)                  Range checking atof
30180: (4)              """
30181: (4)              return _mupdf.fz_atof(s)
30182: (0)          def fz_atoi(s):
30183: (4)              r"""
30184: (4)              Class-aware wrapper for `::fz_atoi()`.
30185: (8)                  atoi that copes with NULL
30186: (4)              """
30187: (4)              return _mupdf.fz_atoi(s)
30188: (0)          def fz_atoi64(s):
30189: (4)              r"""
30190: (4)              Class-aware wrapper for `::fz_atoi64()`.
30191: (8)                  64bit atoi that copes with NULL
30192: (4)              """
30193: (4)              return _mupdf.fz_atoi64(s)
30194: (0)          def fz_authenticate_password(doc, password):
30195: (4)              r"""
30196: (4)              Class-aware wrapper for `::fz_authenticate_password()`.
30197: (8)                  Test if the given password can decrypt the document.
30198: (8)                  password: The password string to be checked. Some document
30199: (8)                  specifications do not specify any particular text encoding, so
30200: (8)                  neither do we.
30201: (8)                  Returns 0 for failure to authenticate, non-zero for success.
30202: (8)                  For PDF documents, further information can be given by examining
30203: (8)                  the bits in the return code.
30204: (12)                     Bit 0 => No password required
30205: (12)                     Bit 1 => User password authenticated
30206: (12)                     Bit 2 => Owner password authenticated
30207: (4)              """
30208: (4)              return _mupdf.fz_authenticate_password(doc, password)
30209: (0)          def fz_available(stm, max):
30210: (4)              r"""
30211: (4)              Class-aware wrapper for `::fz_available()`.
30212: (8)                  Ask how many bytes are available immediately from
30213: (8)                  a given stream.
30214: (8)                  stm: The stream to read from.
30215: (8)                  max: A hint for the underlying stream; the maximum number of
30216: (8)                  bytes that we are sure we will want to read. If you do not know
30217: (8)                  this number, give 1.
30218: (8)                  Returns the number of bytes immediately available between the
30219: (8)                  read and write pointers. This number is guaranteed only to be 0
30220: (8)                  if we have hit EOF. The number of bytes returned here need have
30221: (8)                  no relation to max (could be larger, could be smaller).
30222: (4)              """
30223: (4)              return _mupdf.fz_available(stm, max)
30224: (0)          def fz_base_colorspace(cs):
30225: (4)              r"""
30226: (4)              Class-aware wrapper for `::fz_base_colorspace()`.
30227: (8)                  Get the 'base' colorspace for a colorspace.
30228: (8)                  For indexed colorspaces, this is the colorspace the index
30229: (8)                  decodes into. For all other colorspaces, it is the colorspace
30230: (8)                  itself.
30231: (8)                  The returned colorspace is 'borrowed' (i.e. no additional
30232: (8)                  references are taken or dropped).
30233: (4)              """
30234: (4)              return _mupdf.fz_base_colorspace(cs)
30235: (0)          def fz_basename(path):
30236: (4)              r"""
30237: (4)              Class-aware wrapper for `::fz_basename()`.
30238: (8)                  Find the filename component in a path.
30239: (4)              """
30240: (4)              return _mupdf.fz_basename(path)
30241: (0)          def fz_begin_group(dev, area, cs, isolated, knockout, blendmode, alpha):
30242: (4)              r""" Class-aware wrapper for `::fz_begin_group()`."""
30243: (4)              return _mupdf.fz_begin_group(dev, area, cs, isolated, knockout, blendmode,
alpha)
30244: (0)          def fz_begin_layer(dev, layer_name):
```

```
30245: (4)              r""" Class-aware wrapper for `::fz_begin_layer()`."""
30246: (4)              return _mupdf.fz_begin_layer(dev, layer_name)
30247: (0)          def fz_begin_mask(dev, area, luminosity, colorspace, bc, color_params):
30248: (4)              r""" Class-aware wrapper for `::fz_begin_mask()`."""
30249: (4)              return _mupdf.fz_begin_mask(dev, area, luminosity, colorspace, bc,
color_params)
30250: (0)          def fz_begin_metatext(dev, meta, text):
30251: (4)              r""" Class-aware wrapper for `::fz_begin_metatext()`."""
30252: (4)              return _mupdf.fz_begin_metatext(dev, meta, text)
30253: (0)          def fz_begin_page(wri, mediabox):
30254: (4)              r"""
30255: (4)              Class-aware wrapper for `::fz_begin_page()`.
30256: (8)                  Called to start the process of writing a page to
30257: (8)                  a document.
30258: (8)                  mediabox: page size rectangle in points.
30259: (8)                  Returns a borrowed fz_device to write page contents to. This
30260: (8)                  should be kept if required, and only dropped if it was kept.
30261: (4)              """
30262: (4)              return _mupdf.fz_begin_page(wri, mediabox)
30263: (0)          def fz_begin_structure(dev, standard, raw, idx):
30264: (4)              r""" Class-aware wrapper for `::fz_begin_structure()`."""
30265: (4)              return _mupdf.fz_begin_structure(dev, standard, raw, idx)
30266: (0)          def fz_begin_tile(dev, area, view, xstep, ystep, ctm):
30267: (4)              r""" Class-aware wrapper for `::fz_begin_tile()`."""
30268: (4)              return _mupdf.fz_begin_tile(dev, area, view, xstep, ystep, ctm)
30269: (0)          def fz_begin_tile_id(dev, area, view, xstep, ystep, ctm, id):
30270: (4)              r""" Class-aware wrapper for `::fz_begin_tile_id()`."""
30271: (4)              return _mupdf.fz_begin_tile_id(dev, area, view, xstep, ystep, ctm, id)
30272: (0)          def fz_bidi_fragment_text(text, textlen, baseDir, callback, arg, flags):
30273: (4)              r"""
30274: (5)               Class-aware wrapper for `::fz_bidi_fragment_text()`.
30275: (8)                  This function has out-params. Python/C# wrappers look like:
30276: (12)                     `fz_bidi_fragment_text(const uint32_t *text, size_t textlen,
::fz_bidi_fragment_fn *callback, void *arg, int flags)` => ::fz_bidi_direction baseDir
30277: (12)                     Partitions the given Unicode sequence into one or more
30278: (12)                     unidirectional fragments and invokes the given callback
30279: (12)                     function for each fragment.
30280: (12)                     For example, if directionality of text is:
30281: (20)                             0123456789
30282: (20)                             rrlllrrrrr,
30283: (12)                     we'll invoke callback with:
30284: (20)                             &text[0], length == 2
30285: (20)                             &text[2], length == 3
30286: (20)                             &text[5], length == 5
30287: (12)                     :type text: int, in
30288: (4)              :param text:    start of Unicode sequence
30289: (16)                         :type textlen: int, in
30290: (4)              :param textlen:   number of Unicodes to analyse
30291: (16)                         :type baseDir: int, in
30292: (4)              :param baseDir:   direction of paragraph (specify FZ_BIDI_NEUTRAL to force
auto-detection)
30293: (16)                         :type callback: ::fz_bidi_fragment_fn, in
30294: (4)              :param callback:  function to be called for each fragment
30295: (16)                         :type arg: void, in
30296: (4)              :param arg:    data to be passed to the callback function
30297: (16)                         :type flags: int, in
30298: (4)              :param flags:     flags to control operation (see fz_bidi_flags above)
30299: (4)              """
30300: (4)              return _mupdf.fz_bidi_fragment_text(text, textlen, baseDir, callback, arg,
flags)
30301: (0)          def fz_bitmap_details(bitmap, w, h, n, stride):
30302: (4)              r"""
30303: (4)              Class-aware wrapper for `::fz_bitmap_details()`.
30304: (4)              This function has out-params. Python/C# wrappers look like:
30305: (8)                  `fz_bitmap_details(::fz_bitmap *bitmap)` => `(int w, int h, int n, int
stride)`
30306: (8)                  Retrieve details of a given bitmap.
30307: (8)                  bitmap: The bitmap to query.
30308: (8)                  w: Pointer to storage to retrieve width (or NULL).
```

```
(8)                 h: Pointer to storage to retrieve height (or NULL).
(8)                 n: Pointer to storage to retrieve number of color components (or
(8)                 NULL).
(8)                 stride: Pointer to storage to retrieve bitmap stride (or NULL).
(4)             """
(4)             return _mupdf.fz_bitmap_details(bitmap, w, h, n, stride)
(0)         def fz_blendmode_name(blendmode):
(4)             r"""
(4)             Class-aware wrapper for `::fz_blendmode_name()`.
(8)                 Map from enumeration to blend mode string.
(8)                 The string is static, with arbitrary lifespan.
(4)             """
(4)             return _mupdf.fz_blendmode_name(blendmode)
(0)         def fz_bound_display_list(list):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_display_list()`.
(8)                 Return the bounding box of the page recorded in a display list.
(4)             """
(4)             return _mupdf.fz_bound_display_list(list)
(0)         def fz_bound_glyph(font, gid, trm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_glyph()`.
(8)                 Return a bbox for a given glyph in a font.
(8)                 font: The font to look for the glyph in.
(8)                 gid: The glyph to bound.
(8)                 trm: The matrix to apply to the glyph before bounding.
(8)                 Returns rectangle by value containing the bounds of the given
(8)                 glyph.
(4)             """
(4)             return _mupdf.fz_bound_glyph(font, gid, trm)
(0)         def fz_bound_page(page):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_page()`.
(8)                 Determine the size of a page at 72 dpi.
(4)             """
(4)             return _mupdf.fz_bound_page(page)
(0)         def fz_bound_page_box(page, box):
(4)             r""" Class-aware wrapper for `::fz_bound_page_box()`."""
(4)             return _mupdf.fz_bound_page_box(page, box)
(0)         def fz_bound_path(path, stroke, ctm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_path()`.
(8)                 Return a bounding rectangle for a path.
(8)                 path: The path to bound.
(8)                 stroke: If NULL, the bounding rectangle given is for
(8)                 the filled path. If non-NULL the bounding rectangle
(8)                 given is for the path stroked with the given attributes.
(8)                 ctm: The matrix to apply to the path during stroking.
(8)                 r: Pointer to a fz_rect which will be used to hold
(8)                 the result.
(8)                 Returns r, updated to contain the bounding rectangle.
(4)             """
(4)             return _mupdf.fz_bound_path(path, stroke, ctm)
(0)         def fz_bound_shade(shade, ctm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_shade()`.
(8)                 Bound a given shading.
(8)                 shade: The shade to bound.
(8)                 ctm: The transform to apply to the shade before bounding.
(8)                 r: Pointer to storage to put the bounds in.
(8)                 Returns r, updated to contain the bounds for the shading.
(4)             """
(4)             return _mupdf.fz_bound_shade(shade, ctm)
(0)         def fz_bound_text(text, stroke, ctm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_bound_text()`.
(8)                 Find the bounds of a given text object.
(8)                 text: The text object to find the bounds of.
(8)                 stroke: Pointer to the stroke attributes (for stroked
```

```
30378: (8)                  text), or NULL (for filled text).
30379: (8)                  ctm: The matrix in use.
30380: (8)                  r: pointer to storage for the bounds.
30381: (8)                  Returns a pointer to r, which is updated to contain the
30382: (8)                  bounding box for the text object.
30383: (4)              """
30384: (4)              return _mupdf.fz_bound_text(text, stroke, ctm)
30385: (0)          def fz_box_type_from_string(name):
30386: (4)              r""" Class-aware wrapper for `::fz_box_type_from_string()`."""
30387: (4)              return _mupdf.fz_box_type_from_string(name)
30388: (0)          def fz_buffer_extract(buf, data):
30389: (4)              r"""
30390: (4)              Class-aware wrapper for `::fz_buffer_extract()`.
30391: (4)              This function has out-params. Python/C# wrappers look like:
30392: (8)                  `fz_buffer_extract(::fz_buffer *buf)` => `(size_t, unsigned char
*data)`
30393: (8)                  Take ownership of buffer contents.
30394: (8)                  Performs the same task as fz_buffer_storage, but ownership of
30395: (8)                  the data buffer returns with this call. The buffer is left
30396: (8)                  empty.
30397: (8)                  Note: Bad things may happen if this is called on a buffer with
30398: (8)                  multiple references that is being used from multiple threads.
30399: (8)                  data: Pointer to place to retrieve data pointer.
30400: (8)                  Returns length of stream.
30401: (4)              """
30402: (4)              return _mupdf.fz_buffer_extract(buf, data)
30403: (0)          def fz_buffer_storage(buf, datap):
30404: (4)              r"""
30405: (4)              Class-aware wrapper for `::fz_buffer_storage()`.
30406: (4)              This function has out-params. Python/C# wrappers look like:
30407: (8)                  `fz_buffer_storage(::fz_buffer *buf)` => `(size_t, unsigned char
*datap)`
30408: (8)                  Retrieve internal memory of buffer.
30409: (8)                  datap: Output parameter that will be pointed to the data.
30410: (8)                  Returns the current size of the data in bytes.
30411: (4)              """
30412: (4)              return _mupdf.fz_buffer_storage(buf, datap)
30413: (0)          def fz_calloc(count, size):
30414: (4)              r"""
30415: (4)              Class-aware wrapper for `::fz_calloc()`.
30416: (8)                  Allocate array of memory of count entries of size bytes.
30417: (8)                  Clears the memory to zero.
30418: (8)                  Throws exception in the event of failure to allocate.
30419: (4)              """
30420: (4)              return _mupdf.fz_calloc(count, size)
30421: (0)          def fz_calloc_no_throw(count, size):
30422: (4)              r"""
30423: (4)              Class-aware wrapper for `::fz_calloc_no_throw()`.
30424: (8)                  fz_calloc equivalent that returns NULL rather than throwing
30425: (8)                  exceptions.
30426: (4)              """
30427: (4)              return _mupdf.fz_calloc_no_throw(count, size)
30428: (0)          def fz_caught():
30429: (4)              r"""
30430: (4)              Class-aware wrapper for `::fz_caught()`.
30431: (8)                  Within an fz_catch() block, retrieve the error code for
30432: (8)                  the current exception.
30433: (8)                  This assumes no intervening use of fz_try/fz_catch.
30434: (4)              """
30435: (4)              return _mupdf.fz_caught()
30436: (0)          def fz_caught_errno():
30437: (4)              r""" Class-aware wrapper for `::fz_caught_errno()`."""
30438: (4)              return _mupdf.fz_caught_errno()
30439: (0)          def fz_caught_message():
30440: (4)              r"""
30441: (4)              Class-aware wrapper for `::fz_caught_message()`.
30442: (8)                  Within an fz_catch() block, retrieve the formatted message
30443: (8)                  string for the current exception.
30444: (8)                  This assumes no intervening use of fz_try/fz_catch.
```

```
30445: (4)             """
30446: (4)             return _mupdf.fz_caught_message()
30447: (0)         def fz_chartorune(rune, str):
30448: (4)             r"""
30449: (4)             Class-aware wrapper for `::fz_chartorune()`.
30450: (4)             This function has out-params. Python/C# wrappers look like:
30451: (8)                 `fz_chartorune(const char *str)` => `(int, int rune)`
30452: (8)                 UTF8 decode a single rune from a sequence of chars.
30453: (8)                 rune: Pointer to an int to assign the decoded 'rune' to.
30454: (8)                 str: Pointer to a UTF8 encoded string.
30455: (8)                 Returns the number of bytes consumed.
30456: (4)             """
30457: (4)             return _mupdf.fz_chartorune(rune, str)
30458: (0)         def fz_clamp(x, min, max):
30459: (4)             r""" Class-aware wrapper for `::fz_clamp()`."""
30460: (4)             return _mupdf.fz_clamp(x, min, max)
30461: (0)         def fz_clamp64(x, min, max):
30462: (4)             r""" Class-aware wrapper for `::fz_clamp64()`."""
30463: (4)             return _mupdf.fz_clamp64(x, min, max)
30464: (0)         def fz_clamp_color(cs, _in, out):
30465: (4)             r"""
30466: (4)             Class-aware wrapper for `::fz_clamp_color()`.
30467: (4)             This function has out-params. Python/C# wrappers look like:
30468: (8)                 `fz_clamp_color(::fz_colorspace *cs, const float *in)` => float out
30469: (8)                 Clamp the samples in a color to the correct ranges for a
30470: (8)                 given colorspace.
30471: (4)             """
30472: (4)             return _mupdf.fz_clamp_color(cs, _in, out)
30473: (0)         def fz_clamp_location(doc, loc):
30474: (4)             r"""
30475: (4)             Class-aware wrapper for `::fz_clamp_location()`.
30476: (8)                 Clamps a location into valid chapter/page range. (First clamps
30477: (8)                 the chapter into range, then the page into range).
30478: (4)             """
30479: (4)             return _mupdf.fz_clamp_location(doc, loc)
30480: (0)         def fz_clampd(x, min, max):
30481: (4)             r""" Class-aware wrapper for `::fz_clampd()`."""
30482: (4)             return _mupdf.fz_clampd(x, min, max)
30483: (0)         def fz_clampi(x, min, max):
30484: (4)             r""" Class-aware wrapper for `::fz_clampi()`."""
30485: (4)             return _mupdf.fz_clampi(x, min, max)
30486: (0)         def fz_clampp(x, min, max):
30487: (4)             r""" Class-aware wrapper for `::fz_clampp()`."""
30488: (4)             return _mupdf.fz_clampp(x, min, max)
30489: (0)         def fz_cleanname(name):
30490: (4)             r"""
30491: (4)             Class-aware wrapper for `::fz_cleanname()`.
30492: (8)                 rewrite path to the shortest string that names the same path.
30493: (8)                 Eliminates multiple and trailing slashes, interprets "." and
30494: (8)                 "..". Overwrites the string in place.
30495: (4)             """
30496: (4)             return _mupdf.fz_cleanname(name)
30497: (0)         def fz_cleanname_strdup(name):
30498: (4)             r"""
30499: (4)             Class-aware wrapper for `::fz_cleanname_strdup()`.
30500: (8)                 rewrite path to the shortest string that names the same path.
30501: (8)                 Eliminates multiple and trailing slashes, interprets "." and
30502: (8)                 "..". Allocates a new string that the caller must free.
30503: (4)             """
30504: (4)             return _mupdf.fz_cleanname_strdup(name)
30505: (0)         def fz_clear_bitmap(bit):
30506: (4)             r"""
30507: (4)             Class-aware wrapper for `::fz_clear_bitmap()`.
30508: (8)                 Set the entire bitmap to 0.
30509: (8)                 Never throws exceptions.
30510: (4)             """
30511: (4)             return _mupdf.fz_clear_bitmap(bit)
30512: (0)         def fz_clear_buffer(buf):
30513: (4)             r"""
```

```
(4)             Class-aware wrapper for `::fz_clear_buffer()`.
(8)                 Empties the buffer. Storage is not freed, but is held ready
(8)                 to be reused as the buffer is refilled.
(8)                 Never throws exceptions.
(4)             """
(4)             return _mupdf.fz_clear_buffer(buf)
(0)         def fz_clear_pixmap(pix):
(4)             r"""
(4)             Class-aware wrapper for `::fz_clear_pixmap()`.
(8)                 Sets all components (including alpha) of
(8)                 all pixels in a pixmap to 0.
(8)                 pix: The pixmap to clear.
(4)             """
(4)             return _mupdf.fz_clear_pixmap(pix)
(0)         def fz_clear_pixmap_rect_with_value(pix, value, r):
(4)             r"""
(4)             Class-aware wrapper for `::fz_clear_pixmap_rect_with_value()`.
(8)                 Clears a subrect of a pixmap with the given value.
(8)                 pix: The pixmap to clear.
(8)                 value: Values in the range 0 to 255 are valid. Each component
(8)                 sample for each pixel in the pixmap will be set to this value,
(8)                 while alpha will always be set to 255 (non-transparent).
(8)                 r: the rectangle.
(4)             """
(4)             return _mupdf.fz_clear_pixmap_rect_with_value(pix, value, r)
(0)         def fz_clear_pixmap_with_value(pix, value):
(4)             r"""
(4)             Class-aware wrapper for `::fz_clear_pixmap_with_value()`.
(8)                 Clears a pixmap with the given value.
(8)                 pix: The pixmap to clear.
(8)                 value: Values in the range 0 to 255 are valid. Each component
(8)                 sample for each pixel in the pixmap will be set to this value,
(8)                 while alpha will always be set to 255 (non-transparent).
(8)                 This function is horrible, and should be removed from the
(8)                 API and replaced with a less magic one.
(4)             """
(4)             return _mupdf.fz_clear_pixmap_with_value(pix, value)
(0)         def fz_clip_image_mask(dev, image, ctm, scissor):
(4)             r""" Class-aware wrapper for `::fz_clip_image_mask()`."""
(4)             return _mupdf.fz_clip_image_mask(dev, image, ctm, scissor)
(0)         def fz_clip_path(dev, path, even_odd, ctm, scissor):
(4)             r""" Class-aware wrapper for `::fz_clip_path()`."""
(4)             return _mupdf.fz_clip_path(dev, path, even_odd, ctm, scissor)
(0)         def fz_clip_stroke_path(dev, path, stroke, ctm, scissor):
(4)             r""" Class-aware wrapper for `::fz_clip_stroke_path()`."""
(4)             return _mupdf.fz_clip_stroke_path(dev, path, stroke, ctm, scissor)
(0)         def fz_clip_stroke_text(dev, text, stroke, ctm, scissor):
(4)             r""" Class-aware wrapper for `::fz_clip_stroke_text()`."""
(4)             return _mupdf.fz_clip_stroke_text(dev, text, stroke, ctm, scissor)
(0)         def fz_clip_text(dev, text, ctm, scissor):
(4)             r""" Class-aware wrapper for `::fz_clip_text()`."""
(4)             return _mupdf.fz_clip_text(dev, text, ctm, scissor)
(0)         def fz_clone_buffer(buf):
(4)             r"""
(4)             Class-aware wrapper for `::fz_clone_buffer()`.
(8)                 Make a new buffer, containing a copy of the data used in
(8)                 the original.
(4)             """
(4)             return _mupdf.fz_clone_buffer(buf)
(0)         def fz_clone_default_colorspaces(base):
(4)             r"""
(4)             Class-aware wrapper for `::fz_clone_default_colorspaces()`.
(8)                 Returns a reference to a newly cloned default colorspaces
(8)                 structure.
(8)                 The new clone may safely be altered without fear of race
(8)                 conditions as the caller is the only reference holder.
(4)             """
(4)             return _mupdf.fz_clone_default_colorspaces(base)
(0)         def fz_clone_path(path):
```

```
30583: (4)              r"""
30584: (4)              Class-aware wrapper for `::fz_clone_path()`.
30585: (8)                  Clone the data for a path.
30586: (8)                  This is used in preference to fz_keep_path when a whole
30587: (8)                  new copy of a path is required, rather than just a shared
30588: (8)                  pointer. This probably indicates that the path is about to
30589: (8)                  be modified.
30590: (8)                  path: path to clone.
30591: (8)                  Throws exceptions on failure to allocate.
30592: (4)              """
30593: (4)              return _mupdf.fz_clone_path(path)
30594: (0)          def fz_clone_pixmap(old):
30595: (4)              r"""
30596: (4)              Class-aware wrapper for `::fz_clone_pixmap()`.
30597: (8)                  Clone a pixmap, copying the pixels and associated data to new
30598: (8)                  storage.
30599: (8)                  The reference count of 'old' is unchanged.
30600: (4)              """
30601: (4)              return _mupdf.fz_clone_pixmap(old)
30602: (0)          def fz_clone_pixmap_area_with_different_seps(src, bbox, dcs, seps,
color_params, default_cs):
30603: (4)              r""" Class-aware wrapper for
`::fz_clone_pixmap_area_with_different_seps()`."""
30604: (4)              return _mupdf.fz_clone_pixmap_area_with_different_seps(src, bbox, dcs,
seps, color_params, default_cs)
30605: (0)          def fz_clone_separations_for_overprint(seps):
30606: (4)              r"""
30607: (4)              Class-aware wrapper for `::fz_clone_separations_for_overprint()`.
30608: (8)                  Return a separations object with all the spots in the input
30609: (8)                  separations object that are set to composite, reset to be
30610: (8)                  enabled. If there ARE no spots in the object, this returns
30611: (8)                  NULL. If the object already has all its spots enabled, then
30612: (8)                  just returns another handle on the same object.
30613: (4)              """
30614: (4)              return _mupdf.fz_clone_separations_for_overprint(seps)
30615: (0)          def fz_clone_stroke_state(stroke):
30616: (4)              r"""
30617: (4)              Class-aware wrapper for `::fz_clone_stroke_state()`.
30618: (8)                  Create an identical stroke_state structure and return a
30619: (8)                  reference to it.
30620: (8)                  stroke: The stroke state reference to clone.
30621: (8)                  Exceptions may be thrown in the event of a failure to
30622: (8)                  allocate.
30623: (4)              """
30624: (4)              return _mupdf.fz_clone_stroke_state(stroke)
30625: (0)          def fz_close_band_writer(writer):
30626: (4)              r"""
30627: (4)              Class-aware wrapper for `::fz_close_band_writer()`.
30628: (8)                  Finishes up the output and closes the band writer. After this
30629: (8)                  call no more headers or bands may be written.
30630: (4)              """
30631: (4)              return _mupdf.fz_close_band_writer(writer)
30632: (0)          def fz_close_device(dev):
30633: (4)              r"""
30634: (4)              Class-aware wrapper for `::fz_close_device()`.
30635: (8)                  Signal the end of input, and flush any buffered output.
30636: (8)                  This is NOT called implicitly on fz_drop_device. This
30637: (8)                  may throw exceptions.
30638: (4)              """
30639: (4)              return _mupdf.fz_close_device(dev)
30640: (0)          def fz_close_document_writer(wri):
30641: (4)              r"""
30642: (4)              Class-aware wrapper for `::fz_close_document_writer()`.
30643: (8)                  Called to end the process of writing
30644: (8)                  pages to a document.
30645: (8)                  This writes any file level trailers required. After this
30646: (8)                  completes successfully the file is up to date and complete.
30647: (4)              """
30648: (4)              return _mupdf.fz_close_document_writer(wri)
```

```
30649: (0)          def fz_close_output(arg_0):
30650: (4)              r"""
30651: (4)              Class-aware wrapper for `::fz_close_output()`.
30652: (8)                  Flush pending output and close an output stream.
30653: (4)              """
30654: (4)              return _mupdf.fz_close_output(arg_0)
30655: (0)          def fz_close_zip_writer(zip):
30656: (4)              r"""
30657: (4)              Class-aware wrapper for `::fz_close_zip_writer()`.
30658: (8)                  Close the zip file for writing.
30659: (8)                  This flushes any pending data to the file. This can throw
30660: (8)                  exceptions.
30661: (4)              """
30662: (4)              return _mupdf.fz_close_zip_writer(zip)
30663: (0)          def fz_closepath(path):
30664: (4)              r"""
30665: (4)              Class-aware wrapper for `::fz_closepath()`.
30666: (8)                  Close the current subpath.
30667: (8)                  path: The path to modify.
30668: (8)                  Throws exceptions on failure to allocate, attempting to modify
30669: (8)                  a packed path, and illegal path closes (i.e. closing a non open
30670: (8)                  path).
30671: (4)              """
30672: (4)              return _mupdf.fz_closepath(path)
30673: (0)          def fz_colorspace_colorant(cs, n):
30674: (4)              r"""
30675: (4)              Class-aware wrapper for `::fz_colorspace_colorant()`.
30676: (8)                  Retrieve a the name for a colorant.
30677: (8)                  Returns a pointer with the same lifespan as the colorspace.
30678: (4)              """
30679: (4)              return _mupdf.fz_colorspace_colorant(cs, n)
30680: (0)          def fz_colorspace_device_n_has_cmyk(cs):
30681: (4)              r"""
30682: (4)              Class-aware wrapper for `::fz_colorspace_device_n_has_cmyk()`.
30683: (8)                  True if DeviceN color space has cyan magenta yellow or black as
30684: (8)                  one of its colorants.
30685: (4)              """
30686: (4)              return _mupdf.fz_colorspace_device_n_has_cmyk(cs)
30687: (0)          def fz_colorspace_device_n_has_only_cmyk(cs):
30688: (4)              r"""
30689: (4)              Class-aware wrapper for `::fz_colorspace_device_n_has_only_cmyk()`.
30690: (8)                  True if DeviceN color space has only colorants from the CMYK set.
30691: (4)              """
30692: (4)              return _mupdf.fz_colorspace_device_n_has_only_cmyk(cs)
30693: (0)          def fz_colorspace_is_cmyk(cs):
30694: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_cmyk()`."""
30695: (4)              return _mupdf.fz_colorspace_is_cmyk(cs)
30696: (0)          def fz_colorspace_is_device(cs):
30697: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_device()`."""
30698: (4)              return _mupdf.fz_colorspace_is_device(cs)
30699: (0)          def fz_colorspace_is_device_cmyk(cs):
30700: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_device_cmyk()`."""
30701: (4)              return _mupdf.fz_colorspace_is_device_cmyk(cs)
30702: (0)          def fz_colorspace_is_device_gray(cs):
30703: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_device_gray()`."""
30704: (4)              return _mupdf.fz_colorspace_is_device_gray(cs)
30705: (0)          def fz_colorspace_is_device_n(cs):
30706: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_device_n()`."""
30707: (4)              return _mupdf.fz_colorspace_is_device_n(cs)
30708: (0)          def fz_colorspace_is_gray(cs):
30709: (4)              r"""
30710: (4)              Class-aware wrapper for `::fz_colorspace_is_gray()`.
30711: (8)                  Tests for particular types of colorspaces
30712: (4)              """
30713: (4)              return _mupdf.fz_colorspace_is_gray(cs)
30714: (0)          def fz_colorspace_is_indexed(cs):
30715: (4)              r""" Class-aware wrapper for `::fz_colorspace_is_indexed()`."""
30716: (4)              return _mupdf.fz_colorspace_is_indexed(cs)
30717: (0)          def fz_colorspace_is_lab(cs):
```

```
(4)             r""" Class-aware wrapper for `::fz_colorspace_is_lab()`."""
(4)             return _mupdf.fz_colorspace_is_lab(cs)
(0)         def fz_colorspace_is_lab_icc(cs):
(4)             r""" Class-aware wrapper for `::fz_colorspace_is_lab_icc()`."""
(4)             return _mupdf.fz_colorspace_is_lab_icc(cs)
(0)         def fz_colorspace_is_rgb(cs):
(4)             r""" Class-aware wrapper for `::fz_colorspace_is_rgb()`."""
(4)             return _mupdf.fz_colorspace_is_rgb(cs)
(0)         def fz_colorspace_is_subtractive(cs):
(4)             r"""
(4)             Class-aware wrapper for `::fz_colorspace_is_subtractive()`.
(8)                 True for CMYK, Separation and DeviceN colorspaces.
(4)             """
(4)             return _mupdf.fz_colorspace_is_subtractive(cs)
(0)         def fz_colorspace_n(cs):
(4)             r"""
(4)             Class-aware wrapper for `::fz_colorspace_n()`.
(8)                 Query the number of colorants in a colorspace.
(4)             """
(4)             return _mupdf.fz_colorspace_n(cs)
(0)         def fz_colorspace_name(cs):
(4)             r"""
(4)             Class-aware wrapper for `::fz_colorspace_name()`.
(8)                 Query the name of a colorspace.
(8)                 The returned string has the same lifespan as the colorspace
(8)                 does. Caller should not free it.
(4)             """
(4)             return _mupdf.fz_colorspace_name(cs)
(0)         def fz_colorspace_name_colorant(cs, n, name):
(4)             r"""
(4)             Class-aware wrapper for `::fz_colorspace_name_colorant()`.
(8)                 Assign a name for a given colorant in a colorspace.
(8)                 Used while initially setting up a colorspace. The string is
(8)                 copied into local storage, so need not be retained by the
(8)                 caller.
(4)             """
(4)             return _mupdf.fz_colorspace_name_colorant(cs, n, name)
(0)         def fz_colorspace_type(cs):
(4)             r"""
(4)             Class-aware wrapper for `::fz_colorspace_type()`.
(8)                 Query the type of colorspace.
(4)             """
(4)             return _mupdf.fz_colorspace_type(cs)
(0)         def fz_compare_separations(sep1, sep2):
(4)             r"""
(4)             Class-aware wrapper for `::fz_compare_separations()`.
(8)                 Compare 2 separations structures (or NULLs).
(8)                 Return 0 if identical, non-zero if not identical.
(4)             """
(4)             return _mupdf.fz_compare_separations(sep1, sep2)
(0)         def fz_compress_ccitt_fax_g3(data, columns, rows, stride):
(4)             r"""
(4)             Class-aware wrapper for `::fz_compress_ccitt_fax_g3()`.
(8)                 Compress bitmap data as CCITT Group 3 1D fax image.
(8)                 Creates a stream assuming the default PDF parameters,
(8)                 except the number of columns.
(4)             """
(4)             return _mupdf.fz_compress_ccitt_fax_g3(data, columns, rows, stride)
(0)         def fz_compress_ccitt_fax_g4(data, columns, rows, stride):
(4)             r"""
(4)             Class-aware wrapper for `::fz_compress_ccitt_fax_g4()`.
(8)                 Compress bitmap data as CCITT Group 4 2D fax image.
(8)                 Creates a stream assuming the default PDF parameters, except
(8)                 K=-1 and the number of columns.
(4)             """
(4)             return _mupdf.fz_compress_ccitt_fax_g4(data, columns, rows, stride)
(0)         def fz_compressed_buffer_size(buffer):
(4)             r"""
(4)             Class-aware wrapper for `::fz_compressed_buffer_size()`.
```

```
30787: (8)                  Return the storage size used for a buffer and its data.
30788: (8)                  Used in implementing store handling.
30789: (8)                  Never throws exceptions.
30790: (4)              """
30791: (4)              return _mupdf.fz_compressed_buffer_size(buffer)
30792: (0)          def fz_compressed_image_buffer(image):
30793: (4)              r"""
30794: (4)              Class-aware wrapper for `::fz_compressed_image_buffer()`.
30795: (8)                  Retrieve the underlying compressed data for an image.
30796: (8)                  Returns a pointer to the underlying data buffer for an image,
30797: (8)                  or NULL if this image is not based upon a compressed data
30798: (8)                  buffer.
30799: (8)                  This is not a reference counted structure, so no reference is
30800: (8)                  returned. Lifespan is limited to that of the image itself.
30801: (4)              """
30802: (4)              return _mupdf.fz_compressed_image_buffer(image)
30803: (0)          def fz_compressed_image_type(image):
30804: (4)              r"""
30805: (4)              Class-aware wrapper for `::fz_compressed_image_type()`.
30806: (8)                  Return the type of a compressed image.
30807: (8)                  Any non-compressed image will have the type returned as UNKNOWN.
30808: (4)              """
30809: (4)              return _mupdf.fz_compressed_image_type(image)
30810: (0)          def fz_concat(left, right):
30811: (4)              r"""
30812: (4)              Class-aware wrapper for `::fz_concat()`.
30813: (8)                  Multiply two matrices.
30814: (8)                  The order of the two matrices are important since matrix
30815: (8)                  multiplication is not commutative.
30816: (8)                  Returns result.
30817: (4)              """
30818: (4)              return _mupdf.fz_concat(left, right)
30819: (0)          def fz_contains_rect(a, b):
30820: (4)              r"""
30821: (4)              Class-aware wrapper for `::fz_contains_rect()`.
30822: (8)                  Test rectangle inclusion.
30823: (8)                  Return true if a entirely contains b.
30824: (4)              """
30825: (4)              return _mupdf.fz_contains_rect(a, b)
30826: (0)          def fz_convert_color(ss, sv, ds, dv, _is, params):
30827: (4)              r"""
30828: (4)              Class-aware wrapper for `::fz_convert_color()`.
30829: (4)              This function has out-params. Python/C# wrappers look like:
30830: (8)                  `fz_convert_color(::fz_colorspace *ss, const float *sv,
::fz_colorspace *ds, ::fz_colorspace *is, ::fz_color_params params)` => float dv
30831: (8)                  Convert color values sv from colorspace ss into colorvalues dv
30832: (8)                  for colorspace ds, via an optional intervening space is,
30833: (8)                  respecting the given color_params.
30834: (4)              """
30835: (4)              return _mupdf.fz_convert_color(ss, sv, ds, dv, _is, params)
30836: (0)          def fz_convert_error(code):
30837: (4)              r"""
30838: (4)              Class-aware wrapper for `::fz_convert_error()`.
30839: (4)              This function has out-params. Python/C# wrappers look like:
30840: (8)                  `fz_convert_error()` => `(const char *, int code)`
30841: (4)              """
30842: (4)              return _mupdf.fz_convert_error(code)
30843: (0)          def fz_convert_indexed_pixmap_to_base(src):
30844: (4)              r"""
30845: (4)              Class-aware wrapper for `::fz_convert_indexed_pixmap_to_base()`.
30846: (8)                  Convert pixmap from indexed to base colorspace.
30847: (8)                  This creates a new bitmap containing the converted pixmap data.
30848: (4)              """
30849: (4)              return _mupdf.fz_convert_indexed_pixmap_to_base(src)
30850: (0)          def fz_convert_pixmap(pix, cs_des, prf, default_cs, color_params, keep_alpha):
30851: (4)              r"""
30852: (4)              Class-aware wrapper for `::fz_convert_pixmap()`.
30853: (8)                  Convert an existing pixmap to a desired
30854: (8)                  colorspace. Other properties of the pixmap, such as resolution
```

```
30855: (8)                  and position are copied to the converted pixmap.
30856: (8)                  pix: The pixmap to convert.
30857: (8)                  default_cs: If NULL pix->colorspace is used. It is possible that
30858: (8)                  the data may need to be interpreted as one of the color spaces
30859: (8)                  in default_cs.
30860: (8)                  cs_des: Desired colorspace, may be NULL to denote alpha-only.
30861: (8)                  prf: Proofing color space through which we need to convert.
30862: (8)                  color_params: Parameters that may be used in conversion (e.g.
30863: (8)                  ri).
30864: (8)                  keep_alpha: If 0 any alpha component is removed, otherwise
30865: (8)                  alpha is kept if present in the pixmap.
30866: (4)              """
30867: (4)              return _mupdf.fz_convert_pixmap(pix, cs_des, prf, default_cs,
color_params, keep_alpha)
30868: (0)          def fz_convert_separation_colors(src_cs, src_color, dst_seps, dst_cs,
dst_color, color_params):
30869: (4)              r"""
30870: (4)              Class-aware wrapper for `::fz_convert_separation_colors()`.
30871: (4)              This function has out-params. Python/C# wrappers look like:
30872: (8)                  `fz_convert_separation_colors(::fz_colorspace *src_cs, const float
*src_color, ::fz_separations *dst_seps, ::fz_colorspace *dst_cs, ::fz_color_params color_params)`
=> float dst_color
30873: (8)                  Convert a color given in terms of one colorspace,
30874: (8)                  to a color in terms of another colorspace/separations.
30875: (4)              """
30876: (4)              return _mupdf.fz_convert_separation_colors(src_cs, src_color, dst_seps,
dst_cs, dst_color, color_params)
30877: (0)          def fz_convert_separation_pixmap_to_base(src):
30878: (4)              r"""
30879: (4)              Class-aware wrapper for `::fz_convert_separation_pixmap_to_base()`.
30880: (8)                  Convert pixmap from DeviceN/Separation to base colorspace.
30881: (8)                  This creates a new bitmap containing the converted pixmap data.
30882: (4)              """
30883: (4)              return _mupdf.fz_convert_separation_pixmap_to_base(src)
30884: (0)          def fz_copy_option(val, dest, maxlen):
30885: (4)              r"""
30886: (4)              Class-aware wrapper for `::fz_copy_option()`.
30887: (8)                  Copy an option (val) into a destination buffer (dest), of maxlen
30888: (8)                  bytes.
30889: (8)                  Returns the number of bytes (including terminator) that did not
30890: (8)                  fit. If val is maxlen or greater bytes in size, it will be left
30891: (8)                  unterminated.
30892: (4)              """
30893: (4)              return _mupdf.fz_copy_option(val, dest, maxlen)
30894: (0)          def fz_copy_pixmap_rect(dest, src, r, default_cs):
30895: (4)              r""" Class-aware wrapper for `::fz_copy_pixmap_rect()`."""
30896: (4)              return _mupdf.fz_copy_pixmap_rect(dest, src, r, default_cs)
30897: (0)          def fz_copy_rectangle(page, area, crlf):
30898: (4)              r"""
30899: (4)              Class-aware wrapper for `::fz_copy_rectangle()`.
30900: (8)                  Return a newly allocated UTF-8 string with the text for a given
30901: (8)                  selection rectangle.
30902: (8)                  crlf: If true, write "\r\n" style line endings (otherwise "\n"
30903: (8)                  only).
30904: (4)              """
30905: (4)              return _mupdf.fz_copy_rectangle(page, area, crlf)
30906: (0)          def fz_copy_selection(page, a, b, crlf):
30907: (4)              r"""
30908: (4)              Class-aware wrapper for `::fz_copy_selection()`.
30909: (8)                  Return a newly allocated UTF-8 string with the text for a given
30910: (8)                  selection.
30911: (8)                  crlf: If true, write "\r\n" style line endings (otherwise "\n"
30912: (8)                  only).
30913: (4)              """
30914: (4)              return _mupdf.fz_copy_selection(page, a, b, crlf)
30915: (0)          def fz_count_active_separations(seps):
30916: (4)              r"""
30917: (4)              Class-aware wrapper for `::fz_count_active_separations()`.
30918: (8)                  Return the number of active separations.
```

```
30919: (4)             """
30920: (4)             return _mupdf.fz_count_active_separations(seps)
30921: (0)         def fz_count_archive_entries(arch):
30922: (4)             r"""
30923: (4)             Class-aware wrapper for `::fz_count_archive_entries()`.
30924: (8)                 Number of entries in archive.
30925: (8)                 Will always return a value >= 0.
30926: (8)                 May throw an exception if this type of archive cannot count the
30927: (8)                 entries (such as a directory).
30928: (4)             """
30929: (4)             return _mupdf.fz_count_archive_entries(arch)
30930: (0)         def fz_count_chapter_pages(doc, chapter):
30931: (4)             r"""
30932: (4)             Class-aware wrapper for `::fz_count_chapter_pages()`.
30933: (8)                 Return the number of pages in a chapter.
30934: (8)                 May return 0.
30935: (4)             """
30936: (4)             return _mupdf.fz_count_chapter_pages(doc, chapter)
30937: (0)         def fz_count_chapters(doc):
30938: (4)             r"""
30939: (4)             Class-aware wrapper for `::fz_count_chapters()`.
30940: (8)                 Return the number of chapters in the document.
30941: (8)                 At least 1.
30942: (4)             """
30943: (4)             return _mupdf.fz_count_chapters(doc)
30944: (0)         def fz_count_pages(doc):
30945: (4)             r"""
30946: (4)             Class-aware wrapper for `::fz_count_pages()`.
30947: (8)                 Return the number of pages in document
30948: (8)                 May return 0 for documents with no pages.
30949: (4)             """
30950: (4)             return _mupdf.fz_count_pages(doc)
30951: (0)         def fz_count_separations(sep):
30952: (4)             r""" Class-aware wrapper for `::fz_count_separations()`."""
30953: (4)             return _mupdf.fz_count_separations(sep)
30954: (0)         def fz_create_link(page, bbox, uri):
30955: (4)             r"""
30956: (4)             Class-aware wrapper for `::fz_create_link()`.
30957: (8)                 Create a new link on a page.
30958: (4)             """
30959: (4)             return _mupdf.fz_create_link(page, bbox, uri)
30960: (0)         def fz_currentpoint(path):
30961: (4)             r"""
30962: (4)             Class-aware wrapper for `::fz_currentpoint()`.
30963: (8)                 Return the current point that a path has
30964: (8)                 reached or (0,0) if empty.
30965: (8)                 path: path to return the current point of.
30966: (4)             """
30967: (4)             return _mupdf.fz_currentpoint(path)
30968: (0)         def fz_curveto(path, x0, y0, x1, y1, x2, y2):
30969: (4)             r"""
30970: (4)             Class-aware wrapper for `::fz_curveto()`.
30971: (8)                 Append a 'curveto' command to an open path. (For a
30972: (8)                 cubic bezier).
30973: (8)                 path: The path to modify.
30974: (8)                 x0, y0: The coordinates of the first control point for the
30975: (8)                 curve.
30976: (8)                 x1, y1: The coordinates of the second control point for the
30977: (8)                 curve.
30978: (8)                 x2, y2: The end coordinates for the curve.
30979: (8)                 Throws exceptions on failure to allocate, or attempting to
30980: (8)                 modify a packed path.
30981: (4)             """
30982: (4)             return _mupdf.fz_curveto(path, x0, y0, x1, y1, x2, y2)
30983: (0)         def fz_curvetov(path, x1, y1, x2, y2):
30984: (4)             r"""
30985: (4)             Class-aware wrapper for `::fz_curvetov()`.
30986: (8)                 Append a 'curvetov' command to an open path. (For a
30987: (8)                 cubic bezier with the first control coordinate equal to
```

```
30988: (8)                 the start point).
30989: (8)                 path: The path to modify.
30990: (8)                 x1, y1: The coordinates of the second control point for the
30991: (8)                 curve.
30992: (8)                 x2, y2: The end coordinates for the curve.
30993: (8)                 Throws exceptions on failure to allocate, or attempting to
30994: (8)                 modify a packed path.
30995: (4)             """
30996: (4)             return _mupdf.fz_curvetov(path, x1, y1, x2, y2)
30997: (0)         def fz_curvetoy(path, x0, y0, x2, y2):
30998: (4)             r"""
30999: (4)             Class-aware wrapper for `::fz_curvetoy()`.
31000: (8)                 Append a 'curvetoy' command to an open path. (For a
31001: (8)                 cubic bezier with the second control coordinate equal to
31002: (8)                 the end point).
31003: (8)                 path: The path to modify.
31004: (8)                 x0, y0: The coordinates of the first control point for the
31005: (8)                 curve.
31006: (8)                 x2, y2: The end coordinates for the curve (and the second
31007: (8)                 control coordinate).
31008: (8)                 Throws exceptions on failure to allocate, or attempting to
31009: (8)                 modify a packed path.
31010: (4)             """
31011: (4)             return _mupdf.fz_curvetoy(path, x0, y0, x2, y2)
31012: (0)         def fz_debug_store(out):
31013: (4)             r"""
31014: (4)             Class-aware wrapper for `::fz_debug_store()`.
31015: (8)                 Output debugging information for the current state of the store
31016: (8)                 to the given output channel.
31017: (4)             """
31018: (4)             return _mupdf.fz_debug_store(out)
31019: (0)         def fz_debug_xml(item, level):
31020: (4)             r"""
31021: (4)             Class-aware wrapper for `::fz_debug_xml()`.
31022: (8)                 Pretty-print an XML tree to stdout. (Deprecated, use
31023: (8)                 fz_output_xml in preference).
31024: (4)             """
31025: (4)             return _mupdf.fz_debug_xml(item, level)
31026: (0)         def fz_decode_tile(pix, decode):
31027: (4)             r""" Class-aware wrapper for `::fz_decode_tile()`."""
31028: (4)             return _mupdf.fz_decode_tile(pix, decode)
31029: (0)         def fz_decode_uri(s):
31030: (4)             r"""
31031: (5)              Class-aware wrapper for `::fz_decode_uri()`.
31032: (4)             Return a new string representing the unencoded version of the given URI.
31033: (4)             This decodes all escape sequences except those that would result in a
reserved
31034: (4)             character that are part of the URI syntax (; / ? : @ & = + $ , #).
31035: (4)             """
31036: (4)             return _mupdf.fz_decode_uri(s)
31037: (0)         def fz_decode_uri_component(s):
31038: (4)             r"""
31039: (5)              Class-aware wrapper for `::fz_decode_uri_component()`.
31040: (4)             Return a new string representing the unencoded version of the given URI
component.
31041: (4)             This decodes all escape sequences!
31042: (4)             """
31043: (4)             return _mupdf.fz_decode_uri_component(s)
31044: (0)         def fz_decomp_image_from_stream(stm, image, subarea, indexed, l2factor,
l2extra):
31045: (4)             r"""
31046: (4)             Class-aware wrapper for `::fz_decomp_image_from_stream()`.
31047: (4)             This function has out-params. Python/C# wrappers look like:
31048: (8)                 `fz_decomp_image_from_stream(::fz_stream *stm, ::fz_compressed_image
*image, ::fz_irect *subarea, int indexed, int l2factor)` => `(fz_pixmap *, int l2extra)`
31049: (8)                 Decode a subarea of a compressed image. l2factor is the amount
31050: (8)                 of subsampling inbuilt to the stream (i.e. performed by the
31051: (8)                 decoder). If non NULL, l2extra is the extra amount of
31052: (8)                 subsampling that should be performed by this routine. This will
```

```
31053: (8)                  be updated on exit to the amount of subsampling that is still
31054: (8)                  required to be done.
31055: (8)                  Returns a kept reference.
31056: (4)              """
31057: (4)              return _mupdf.fz_decomp_image_from_stream(stm, image, subarea, indexed,
l2factor, l2extra)
31058: (0)          def fz_decouple_type3_font(font, t3doc):
31059: (4)              r""" Class-aware wrapper for `::fz_decouple_type3_font()`."""
31060: (4)              return _mupdf.fz_decouple_type3_font(font, t3doc)
31061: (0)          def fz_default_cmyk(default_cs):
31062: (4)              r""" Class-aware wrapper for `::fz_default_cmyk()`."""
31063: (4)              return _mupdf.fz_default_cmyk(default_cs)
31064: (0)          def fz_default_error_callback(user, message):
31065: (4)              r"""
31066: (4)              Class-aware wrapper for `::fz_default_error_callback()`.
31067: (8)                  FIXME: Better not to expose fz_default_error_callback, and
31068: (8)                  fz_default_warning callback and to allow 'NULL' to be used
31069: (8)                  int fz_set_xxxx_callback to mean "defaults".
31070: (8)                  FIXME: Do we need/want functions like
31071: (8)                  fz_error_callback(ctx, message) to allow callers to inject
31072: (8)                  stuff into the error/warning streams?
31073: (8)                  The default error callback. Declared publicly just so that the
31074: (8)                  error callback can be set back to this after it has been
31075: (8)                  overridden.
31076: (4)              """
31077: (4)              return _mupdf.fz_default_error_callback(user, message)
31078: (0)          def fz_default_gray(default_cs):
31079: (4)              r"""
31080: (4)              Class-aware wrapper for `::fz_default_gray()`.
31081: (8)                  Retrieve default colorspaces (typically page local).
31082: (8)                  If default_cs is non NULL, the default is retrieved from there,
31083: (8)                  otherwise the global default is retrieved.
31084: (8)                  These return borrowed references that should not be dropped,
31085: (8)                  unless they are kept first.
31086: (4)              """
31087: (4)              return _mupdf.fz_default_gray(default_cs)
31088: (0)          def fz_default_halftone(num_comps):
31089: (4)              r"""
31090: (4)              Class-aware wrapper for `::fz_default_halftone()`.
31091: (8)                  Create a 'default' halftone structure
31092: (8)                  for the given number of components.
31093: (8)                  num_comps: The number of components to use.
31094: (8)                  Returns a simple default halftone. The default halftone uses
31095: (8)                  the same halftone tile for each plane, which may not be ideal
31096: (8)                  for all purposes.
31097: (4)              """
31098: (4)              return _mupdf.fz_default_halftone(num_comps)
31099: (0)          def fz_default_output_intent(default_cs):
31100: (4)              r""" Class-aware wrapper for `::fz_default_output_intent()`."""
31101: (4)              return _mupdf.fz_default_output_intent(default_cs)
31102: (0)          def fz_default_rgb(default_cs):
31103: (4)              r""" Class-aware wrapper for `::fz_default_rgb()`."""
31104: (4)              return _mupdf.fz_default_rgb(default_cs)
31105: (0)          def fz_default_warning_callback(user, message):
31106: (4)              r"""
31107: (4)              Class-aware wrapper for `::fz_default_warning_callback()`.
31108: (8)                  The default warning callback. Declared publicly just so that
31109: (8)                  the warning callback can be set back to this after it has been
31110: (8)                  overridden.
31111: (4)              """
31112: (4)              return _mupdf.fz_default_warning_callback(user, message)
31113: (0)          def fz_defer_reap_end():
31114: (4)              r"""
31115: (4)              Class-aware wrapper for `::fz_defer_reap_end()`.
31116: (8)                  Decrement the defer reap count.
31117: (8)                  If the defer reap count returns to 0, and the store
31118: (8)                  has reapable objects in, a reap pass will begin.
31119: (8)                  Call this at the end of a process during which you
31120: (8)                  potentially might drop many reapable objects.
```

```
31121: (8)                  It is vital that every fz_defer_reap_start is matched
31122: (8)                  by a fz_defer_reap_end call.
31123: (4)              """
31124: (4)              return _mupdf.fz_defer_reap_end()
31125: (0)          def fz_defer_reap_start():
31126: (4)              r"""
31127: (4)              Class-aware wrapper for `::fz_defer_reap_start()`.
31128: (8)                  Increment the defer reap count.
31129: (8)                  No reap operations will take place (except for those
31130: (8)                  triggered by an immediate failed malloc) until the
31131: (8)                  defer reap count returns to 0.
31132: (8)                  Call this at the start of a process during which you
31133: (8)                  potentially might drop many reapable objects.
31134: (8)                  It is vital that every fz_defer_reap_start is matched
31135: (8)                  by a fz_defer_reap_end call.
31136: (4)              """
31137: (4)              return _mupdf.fz_defer_reap_start()
31138: (0)          def fz_deflate(dest, compressed_length, source, source_length, level):
31139: (4)              r"""
31140: (5)               Class-aware wrapper for `::fz_deflate()`.
31141: (8)                  This function has out-params. Python/C# wrappers look like:
31142: (12)                     `fz_deflate(unsigned char *dest, const unsigned char *source,
size_t source_length, ::fz_deflate_level level)` => size_t compressed_length
31143: (12)                     Compress source_length bytes of data starting
31144: (12)                     at source, into a buffer of length *destLen, starting at dest.
31145: (4)              compressed_length will be updated on exit to contain the size
31146: (12)                     actually used.
31147: (4)              """
31148: (4)              return _mupdf.fz_deflate(dest, compressed_length, source, source_length,
level)
31149: (0)          def fz_deflate_bound(size):
31150: (4)              r"""
31151: (4)              Class-aware wrapper for `::fz_deflate_bound()`.
31152: (8)                  Returns the upper bound on the
31153: (8)                  size of flated data of length size.
31154: (4)              """
31155: (4)              return _mupdf.fz_deflate_bound(size)
31156: (0)          def fz_delete_link(page, link):
31157: (4)              r"""
31158: (4)              Class-aware wrapper for `::fz_delete_link()`.
31159: (8)                  Delete an existing link on a page.
31160: (4)              """
31161: (4)              return _mupdf.fz_delete_link(page, link)
31162: (0)          def fz_detach_xml(node):
31163: (4)              r"""
31164: (4)              Class-aware wrapper for `::fz_detach_xml()`.
31165: (8)                  Detach a node from the tree, unlinking it from its parent,
31166: (8)                  and setting the document root to the node.
31167: (4)              """
31168: (4)              return _mupdf.fz_detach_xml(node)
31169: (0)          def fz_device_bgr():
31170: (4)              r""" Class-aware wrapper for `::fz_device_bgr()`."""
31171: (4)              return _mupdf.fz_device_bgr()
31172: (0)          def fz_device_cmyk():
31173: (4)              r""" Class-aware wrapper for `::fz_device_cmyk()`."""
31174: (4)              return _mupdf.fz_device_cmyk()
31175: (0)          def fz_device_current_scissor(dev):
31176: (4)              r"""
31177: (4)              Class-aware wrapper for `::fz_device_current_scissor()`.
31178: (8)                  Find current scissor region as tracked by the device.
31179: (4)              """
31180: (4)              return _mupdf.fz_device_current_scissor(dev)
31181: (0)          def fz_device_gray():
31182: (4)              r"""
31183: (4)              Class-aware wrapper for `::fz_device_gray()`.
31184: (8)                  Retrieve global default colorspaces.
31185: (8)                  These return borrowed references that should not be dropped,
31186: (8)                  unless they are kept first.
31187: (4)              """
```

```
31188: (4)             return _mupdf.fz_device_gray()
31189: (0)         def fz_device_lab():
31190: (4)             r""" Class-aware wrapper for `::fz_device_lab()`."""
31191: (4)             return _mupdf.fz_device_lab()
31192: (0)         def fz_device_rgb():
31193: (4)             r""" Class-aware wrapper for `::fz_device_rgb()`."""
31194: (4)             return _mupdf.fz_device_rgb()
31195: (0)         def fz_dirname(dir, path, dirsize):
31196: (4)             r"""
31197: (4)             Class-aware wrapper for `::fz_dirname()`.
31198: (8)                 extract the directory component from a path.
31199: (4)             """
31200: (4)             return _mupdf.fz_dirname(dir, path, dirsize)
31201: (0)         def fz_disable_device_hints(dev, hints):
31202: (4)             r"""
31203: (4)             Class-aware wrapper for `::fz_disable_device_hints()`.
31204: (8)                 Disable (clear) hint bits within the hint bitfield for a device.
31205: (4)             """
31206: (4)             return _mupdf.fz_disable_device_hints(dev, hints)
31207: (0)         def fz_disable_icc():
31208: (4)             r"""
31209: (4)             Class-aware wrapper for `::fz_disable_icc()`.
31210: (8)                 Disable icc profile based operation.
31211: (4)             """
31212: (4)             return _mupdf.fz_disable_icc()
31213: (0)         def fz_display_list_is_empty(list):
31214: (4)             r"""
31215: (4)             Class-aware wrapper for `::fz_display_list_is_empty()`.
31216: (8)                 Check for a display list being empty
31217: (8)                 list: The list to check.
31218: (8)                 Returns true if empty, false otherwise.
31219: (4)             """
31220: (4)             return _mupdf.fz_display_list_is_empty(list)
31221: (0)         def fz_div255(c, a):
31222: (4)             r"""
31223: (4)             Class-aware wrapper for `::fz_div255()`.
31224: (8)                 Undo alpha premultiplication.
31225: (4)             """
31226: (4)             return _mupdf.fz_div255(c, a)
31227: (0)         def fz_do_always():
31228: (4)             r""" Class-aware wrapper for `::fz_do_always()`."""
31229: (4)             return _mupdf.fz_do_always()
31230: (0)         def fz_do_catch():
31231: (4)             r""" Class-aware wrapper for `::fz_do_catch()`."""
31232: (4)             return _mupdf.fz_do_catch()
31233: (0)         def fz_do_try():
31234: (4)             r""" Class-aware wrapper for `::fz_do_try()`."""
31235: (4)             return _mupdf.fz_do_try()
31236: (0)         def fz_document_handler_open(handler, stream, accel, dir, recognize_state):
31237: (4)             r"""
31238: (4)             Class-aware wrapper for `::fz_document_handler_open()`.   Helper for
calling `fz_document_handler::open` function pointer via
31239: (4)             Swig from Python/C#.
31240: (4)             """
31241: (4)             return _mupdf.fz_document_handler_open(handler, stream, accel, dir,
recognize_state)
31242: (0)         def fz_document_handler_recognize(handler, magic):
31243: (4)             r"""
31244: (4)             Class-aware wrapper for `::fz_document_handler_recognize()`.   Helper for
calling a `fz_document_handler::recognize` function
31245: (4)             pointer via Swig from Python/C#.
31246: (4)             """
31247: (4)             return _mupdf.fz_document_handler_recognize(handler, magic)
31248: (0)         def fz_document_output_intent(doc):
31249: (4)             r"""
31250: (4)             Class-aware wrapper for `::fz_document_output_intent()`.
31251: (8)                 Find the output intent colorspace if the document has defined
31252: (8)                 one.
31253: (8)                 Returns a borrowed reference that should not be dropped, unless
```

```
31254: (8)                 it is kept first.
31255: (4)             """
31256: (4)             return _mupdf.fz_document_output_intent(doc)
31257: (0)         def fz_document_supports_accelerator(doc):
31258: (4)             r"""
31259: (4)             Class-aware wrapper for `::fz_document_supports_accelerator()`.
31260: (8)                 Query if the document supports the saving of accelerator data.
31261: (4)             """
31262: (4)             return _mupdf.fz_document_supports_accelerator(doc)
31263: (0)         def fz_dom_add_attribute(elt, att, value):
31264: (4)             r"""
31265: (4)             Class-aware wrapper for `::fz_dom_add_attribute()`.
31266: (8)                 Add an attribute to an element.
31267: (8)                 Ownership of att and value remain with the caller.
31268: (4)             """
31269: (4)             return _mupdf.fz_dom_add_attribute(elt, att, value)
31270: (0)         def fz_dom_append_child(parent, child):
31271: (4)             r"""
31272: (4)             Class-aware wrapper for `::fz_dom_append_child()`.
31273: (8)                 Insert an element as the last child of a parent, unlinking the
31274: (8)                 child from its current position if required.
31275: (4)             """
31276: (4)             return _mupdf.fz_dom_append_child(parent, child)
31277: (0)         def fz_dom_attribute(elt, att):
31278: (4)             r"""
31279: (4)             Class-aware wrapper for `::fz_dom_attribute()`.
31280: (8)                 Retrieve the value of a given attribute from a given element.
31281: (8)                 Returns a borrowed pointer to the value or NULL if not found.
31282: (4)             """
31283: (4)             return _mupdf.fz_dom_attribute(elt, att)
31284: (0)         def fz_dom_body(dom):
31285: (4)             r"""
31286: (4)             Class-aware wrapper for `::fz_dom_body()`.
31287: (8)                 Return a borrowed reference for the 'body' element of
31288: (8)                 the given DOM.
31289: (4)             """
31290: (4)             return _mupdf.fz_dom_body(dom)
31291: (0)         def fz_dom_clone(elt):
31292: (4)             r"""
31293: (4)             Class-aware wrapper for `::fz_dom_clone()`.
31294: (8)                 Clone an element (and its children).
31295: (8)                 A borrowed reference to the clone is returned. The clone is not
31296: (8)                 yet linked into the DOM.
31297: (4)             """
31298: (4)             return _mupdf.fz_dom_clone(elt)
31299: (0)         def fz_dom_create_element(dom, tag):
31300: (4)             r"""
31301: (4)             Class-aware wrapper for `::fz_dom_create_element()`.
31302: (8)                 Create an element of a given tag type for the given DOM.
31303: (8)                 The element is not linked into the DOM yet.
31304: (4)             """
31305: (4)             return _mupdf.fz_dom_create_element(dom, tag)
31306: (0)         def fz_dom_create_text_node(dom, text):
31307: (4)             r"""
31308: (4)             Class-aware wrapper for `::fz_dom_create_text_node()`.
31309: (8)                 Create a text node for the given DOM.
31310: (8)                 The element is not linked into the DOM yet.
31311: (4)             """
31312: (4)             return _mupdf.fz_dom_create_text_node(dom, text)
31313: (0)         def fz_dom_document_element(dom):
31314: (4)             r"""
31315: (4)             Class-aware wrapper for `::fz_dom_document_element()`.
31316: (8)                 Return a borrowed reference for the document (the top
31317: (8)                 level element) of the DOM.
31318: (4)             """
31319: (4)             return _mupdf.fz_dom_document_element(dom)
31320: (0)         def fz_dom_find(elt, tag, att, match):
31321: (4)             r"""
31322: (4)             Class-aware wrapper for `::fz_dom_find()`.
```

```
31323: (8)                  Find the first element matching the requirements in a depth first
traversal from elt.
31324: (8)                  The tagname must match tag, unless tag is NULL, when all tag names are
considered to match.
31325: (8)                  If att is NULL, then all tags match.
31326: (8)                  Otherwise:
31327: (12)                     If match is NULL, then only nodes that have an att attribute
match.
31328: (12)                     If match is non-NULL, then only nodes that have an att attribute
that matches match match.
31329: (8)                  Returns NULL (if no match found), or a borrowed reference to the first
matching element.
31330: (4)              """
31331: (4)              return _mupdf.fz_dom_find(elt, tag, att, match)
31332: (0)          def fz_dom_find_next(elt, tag, att, match):
31333: (4)              r"""
31334: (4)              Class-aware wrapper for `::fz_dom_find_next()`.
31335: (8)                  Find the next element matching the requirements.
31336: (4)              """
31337: (4)              return _mupdf.fz_dom_find_next(elt, tag, att, match)
31338: (0)          def fz_dom_first_child(elt):
31339: (4)              r"""
31340: (4)              Class-aware wrapper for `::fz_dom_first_child()`.
31341: (8)                  Return a borrowed reference to the first child of a node,
31342: (8)                  or NULL if there isn't one.
31343: (4)              """
31344: (4)              return _mupdf.fz_dom_first_child(elt)
31345: (0)          def fz_dom_get_attribute(elt, i, att):
31346: (4)              r"""
31347: (4)              Class-aware wrapper for `::fz_dom_get_attribute()`.
31348: (4)              This function has out-params. Python/C# wrappers look like:
31349: (8)                  `fz_dom_get_attribute(::fz_xml *elt, int i)` => `(const char *, const
char *att)`
31350: (8)                  Enumerate through the attributes of an element.
31351: (8)                  Call with i=0,1,2,3... to enumerate attributes.
31352: (8)                  On return *att and the return value will be NULL if there are not
31353: (8)                  that many attributes to read. Otherwise, *att will be filled in
31354: (8)                  with a borrowed pointer to the attribute name, and the return
31355: (8)                  value will be a borrowed pointer to the value.
31356: (4)              """
31357: (4)              return _mupdf.fz_dom_get_attribute(elt, i, att)
31358: (0)          def fz_dom_insert_after(node, new_elt):
31359: (4)              r"""
31360: (4)              Class-aware wrapper for `::fz_dom_insert_after()`.
31361: (8)                  Insert an element (new_elt), after another element (node),
31362: (8)                  unlinking the new_elt from its current position if required.
31363: (4)              """
31364: (4)              return _mupdf.fz_dom_insert_after(node, new_elt)
31365: (0)          def fz_dom_insert_before(node, new_elt):
31366: (4)              r"""
31367: (4)              Class-aware wrapper for `::fz_dom_insert_before()`.
31368: (8)                  Insert an element (new_elt), before another element (node),
31369: (8)                  unlinking the new_elt from its current position if required.
31370: (4)              """
31371: (4)              return _mupdf.fz_dom_insert_before(node, new_elt)
31372: (0)          def fz_dom_next(elt):
31373: (4)              r"""
31374: (4)              Class-aware wrapper for `::fz_dom_next()`.
31375: (8)                  Return a borrowed reference to the next sibling of a node,
31376: (8)                  or NULL if there isn't one.
31377: (4)              """
31378: (4)              return _mupdf.fz_dom_next(elt)
31379: (0)          def fz_dom_parent(elt):
31380: (4)              r"""
31381: (4)              Class-aware wrapper for `::fz_dom_parent()`.
31382: (8)                  Return a borrowed reference to the parent of a node,
31383: (8)                  or NULL if there isn't one.
31384: (4)              """
31385: (4)              return _mupdf.fz_dom_parent(elt)
```

```
31386: (0)          def fz_dom_previous(elt):
31387: (4)              r"""
31388: (4)              Class-aware wrapper for `::fz_dom_previous()`.
31389: (8)                  Return a borrowed reference to the previous sibling of a node,
31390: (8)                  or NULL if there isn't one.
31391: (4)              """
31392: (4)              return _mupdf.fz_dom_previous(elt)
31393: (0)          def fz_dom_remove(elt):
31394: (4)              r"""
31395: (4)              Class-aware wrapper for `::fz_dom_remove()`.
31396: (8)                  Remove an element from the DOM. The element can be added back
elsewhere
31397: (8)                  if required.
31398: (8)                  No reference counting changes for the element.
31399: (4)              """
31400: (4)              return _mupdf.fz_dom_remove(elt)
31401: (0)          def fz_dom_remove_attribute(elt, att):
31402: (4)              r"""
31403: (4)              Class-aware wrapper for `::fz_dom_remove_attribute()`.
31404: (8)                  Remove an attribute from an element.
31405: (4)              """
31406: (4)              return _mupdf.fz_dom_remove_attribute(elt, att)
31407: (0)          def fz_draw_story(story, dev, ctm):
31408: (4)              r""" Class-aware wrapper for `::fz_draw_story()`."""
31409: (4)              return _mupdf.fz_draw_story(story, dev, ctm)
31410: (0)          def fz_dump_glyph_cache_stats(out):
31411: (4)              r"""
31412: (4)              Class-aware wrapper for `::fz_dump_glyph_cache_stats()`.
31413: (8)                  Dump debug statistics for the glyph cache.
31414: (4)              """
31415: (4)              return _mupdf.fz_dump_glyph_cache_stats(out)
31416: (0)          def fz_duplicate_glyph_names_from_unicode(unicode):
31417: (4)              r""" Class-aware wrapper for
`::fz_duplicate_glyph_names_from_unicode()`."""
31418: (4)              return _mupdf.fz_duplicate_glyph_names_from_unicode(unicode)
31419: (0)          def fz_empty_store():
31420: (4)              r"""
31421: (4)              Class-aware wrapper for `::fz_empty_store()`.
31422: (8)                  Evict every item from the store.
31423: (4)              """
31424: (4)              return _mupdf.fz_empty_store()
31425: (0)          def fz_enable_device_hints(dev, hints):
31426: (4)              r"""
31427: (4)              Class-aware wrapper for `::fz_enable_device_hints()`.
31428: (8)                  Enable (set) hint bits within the hint bitfield for a device.
31429: (4)              """
31430: (4)              return _mupdf.fz_enable_device_hints(dev, hints)
31431: (0)          def fz_enable_icc():
31432: (4)              r"""
31433: (4)              Class-aware wrapper for `::fz_enable_icc()`.
31434: (8)                  Enable icc profile based operation.
31435: (4)              """
31436: (4)              return _mupdf.fz_enable_icc()
31437: (0)          def fz_encode_character(font, unicode):
31438: (4)              r"""
31439: (4)              Class-aware wrapper for `::fz_encode_character()`.
31440: (8)                  Find the glyph id for a given unicode
31441: (8)                  character within a font.
31442: (8)                  font: The font to look for the unicode character in.
31443: (8)                  unicode: The unicode character to encode.
31444: (8)                  Returns the glyph id for the given unicode value, or 0 if
31445: (8)                  unknown.
31446: (4)              """
31447: (4)              return _mupdf.fz_encode_character(font, unicode)
31448: (0)          def fz_encode_character_by_glyph_name(font, glyphname):
31449: (4)              r"""
31450: (4)              Class-aware wrapper for `::fz_encode_character_by_glyph_name()`.
31451: (8)                  Encode character.
31452: (8)                  Either by direct lookup of glyphname within a font, or, failing
```

```
31453: (8)                  that, by mapping glyphname to unicode and thence to the glyph
31454: (8)                  index within the given font.
31455: (8)                  Returns zero for type3 fonts.
31456: (4)              """
31457: (4)              return _mupdf.fz_encode_character_by_glyph_name(font, glyphname)
31458: (0)          def fz_encode_character_sc(font, unicode):
31459: (4)              r"""
31460: (4)              Class-aware wrapper for `::fz_encode_character_sc()`.
31461: (8)                  Encode character, preferring small-caps variant if available.
31462: (8)                  font: The font to look for the unicode character in.
31463: (8)                  unicode: The unicode character to encode.
31464: (8)                  Returns the glyph id for the given unicode value, or 0 if
31465: (8)                  unknown.
31466: (4)              """
31467: (4)              return _mupdf.fz_encode_character_sc(font, unicode)
31468: (0)          def fz_encode_character_with_fallback(font, unicode, script, language,
out_font):
31469: (4)              r"""
31470: (4)              Class-aware wrapper for `::fz_encode_character_with_fallback()`.
31471: (4)              This function has out-params. Python/C# wrappers look like:
31472: (8)                  `fz_encode_character_with_fallback(::fz_font *font, int unicode, int
script, int language, ::fz_font **out_font)` => `(int)`
31473: (8)                  Find the glyph id for
31474: (8)                  a given unicode character within a font, falling back to
31475: (8)                  an alternative if not found.
31476: (8)                  font: The font to look for the unicode character in.
31477: (8)                  unicode: The unicode character to encode.
31478: (8)                  script: The script in use.
31479: (8)                  language: The language in use.
31480: (8)                  out_font: The font handle in which the given glyph represents
31481: (8)                  the requested unicode character. The caller does not own the
31482: (8)                  reference it is passed, so should call fz_keep_font if it is
31483: (8)                  not simply to be used immediately.
31484: (8)                  Returns the glyph id for the given unicode value in the supplied
31485: (8)                  font (and sets *out_font to font) if it is present. Otherwise
31486: (8)                  an alternative fallback font (based on script/language) is
31487: (8)                  searched for. If the glyph is found therein, *out_font is set
31488: (8)                  to this reference, and the glyph reference is returned. If it
31489: (8)                  cannot be found anywhere, the function returns 0.
31490: (4)              """
31491: (4)              return _mupdf.fz_encode_character_with_fallback(font, unicode, script,
language, out_font)
31492: (0)          def fz_encode_uri(s):
31493: (4)              r"""
31494: (5)               Class-aware wrapper for `::fz_encode_uri()`.
31495: (4)              Return a new string representing the provided string encoded as a URI.
31496: (4)              """
31497: (4)              return _mupdf.fz_encode_uri(s)
31498: (0)          def fz_encode_uri_component(s):
31499: (4)              r"""
31500: (5)               Class-aware wrapper for `::fz_encode_uri_component()`.
31501: (4)              Return a new string representing the provided string encoded as an URI
component.
31502: (4)              This also encodes the special reserved characters (; / ? : @ & = + $ , #).
31503: (4)              """
31504: (4)              return _mupdf.fz_encode_uri_component(s)
31505: (0)          def fz_encode_uri_pathname(s):
31506: (4)              r"""
31507: (5)               Class-aware wrapper for `::fz_encode_uri_pathname()`.
31508: (4)              Return a new string representing the provided string encoded as an URI
path name.
31509: (4)              This also encodes the special reserved characters except /.
31510: (4)              """
31511: (4)              return _mupdf.fz_encode_uri_pathname(s)
31512: (0)          def fz_end_group(dev):
31513: (4)              r""" Class-aware wrapper for `::fz_end_group()`."""
31514: (4)              return _mupdf.fz_end_group(dev)
31515: (0)          def fz_end_layer(dev):
31516: (4)              r""" Class-aware wrapper for `::fz_end_layer()`."""
```

```
31517: (4)              return _mupdf.fz_end_layer(dev)
31518: (0)          def fz_end_mask(dev):
31519: (4)              r""" Class-aware wrapper for `::fz_end_mask()`."""
31520: (4)              return _mupdf.fz_end_mask(dev)
31521: (0)          def fz_end_mask_tr(dev, fn):
31522: (4)              r""" Class-aware wrapper for `::fz_end_mask_tr()`."""
31523: (4)              return _mupdf.fz_end_mask_tr(dev, fn)
31524: (0)          def fz_end_metatext(dev):
31525: (4)              r""" Class-aware wrapper for `::fz_end_metatext()`."""
31526: (4)              return _mupdf.fz_end_metatext(dev)
31527: (0)          def fz_end_page(wri):
31528: (4)              r"""
31529: (4)              Class-aware wrapper for `::fz_end_page()`.
31530: (8)                  Called to end the process of writing a page to a
31531: (8)                  document.
31532: (4)              """
31533: (4)              return _mupdf.fz_end_page(wri)
31534: (0)          def fz_end_structure(dev):
31535: (4)              r""" Class-aware wrapper for `::fz_end_structure()`."""
31536: (4)              return _mupdf.fz_end_structure(dev)
31537: (0)          def fz_end_throw_on_repair():
31538: (4)              r""" Class-aware wrapper for `::fz_end_throw_on_repair()`."""
31539: (4)              return _mupdf.fz_end_throw_on_repair()
31540: (0)          def fz_end_tile(dev):
31541: (4)              r""" Class-aware wrapper for `::fz_end_tile()`."""
31542: (4)              return _mupdf.fz_end_tile(dev)
31543: (0)          def fz_error_callback(user):
31544: (4)              r"""
31545: (4)              Class-aware wrapper for `::fz_error_callback()`.
31546: (4)              This function has out-params. Python/C# wrappers look like:
31547: (8)                  `fz_error_callback()` => `(fz_error_cb *, void *user)`
31548: (8)                  Retrieve the currently set error callback, or NULL if none
31549: (8)                  has been set. Optionally, if user is non-NULL, the user pointer
31550: (8)                  given when the warning callback was set is also passed back to
31551: (8)                  the caller.
31552: (4)              """
31553: (4)              return _mupdf.fz_error_callback(user)
31554: (0)          def fz_eval_function(func, _in, inlen, out, outlen):
31555: (4)              r"""
31556: (4)              Class-aware wrapper for `::fz_eval_function()`.
31557: (4)              This function has out-params. Python/C# wrappers look like:
31558: (8)                  `fz_eval_function(::fz_function *func, const float *in, int inlen, int
outlen)` => float out
31559: (4)              """
31560: (4)              return _mupdf.fz_eval_function(func, _in, inlen, out, outlen)
31561: (0)          def fz_expand_irect(a, expand):
31562: (4)              r""" Class-aware wrapper for `::fz_expand_irect()`."""
31563: (4)              return _mupdf.fz_expand_irect(a, expand)
31564: (0)          def fz_expand_rect(b, expand):
31565: (4)              r"""
31566: (4)              Class-aware wrapper for `::fz_expand_rect()`.
31567: (8)                  Expand a bbox by a given amount in all directions.
31568: (4)              """
31569: (4)              return _mupdf.fz_expand_rect(b, expand)
31570: (0)          def fz_extract_ttf_from_ttc(font):
31571: (4)              r""" Class-aware wrapper for `::fz_extract_ttf_from_ttc()`."""
31572: (4)              return _mupdf.fz_extract_ttf_from_ttc(font)
31573: (0)          def fz_file_exists(path):
31574: (4)              r"""
31575: (4)              Class-aware wrapper for `::fz_file_exists()`.
31576: (8)                  Return true if the named file exists and is readable.
31577: (4)              """
31578: (4)              return _mupdf.fz_file_exists(path)
31579: (0)          def fz_fill_image(dev, image, ctm, alpha, color_params):
31580: (4)              r""" Class-aware wrapper for `::fz_fill_image()`."""
31581: (4)              return _mupdf.fz_fill_image(dev, image, ctm, alpha, color_params)
31582: (0)          def fz_fill_image_mask(dev, image, ctm, colorspace, color, alpha,
color_params):
31583: (4)              r""" Class-aware wrapper for `::fz_fill_image_mask()`."""
```

```
31584: (4)              return _mupdf.fz_fill_image_mask(dev, image, ctm, colorspace, color,
alpha, color_params)
31585: (0)          def fz_fill_path(dev, path, even_odd, ctm, colorspace, color, alpha,
color_params):
31586: (4)              r"""
31587: (4)              Class-aware wrapper for `::fz_fill_path()`.
31588: (8)                  Device calls; graphics primitives and containers.
31589: (4)              """
31590: (4)              return _mupdf.fz_fill_path(dev, path, even_odd, ctm, colorspace, color,
alpha, color_params)
31591: (0)          def fz_fill_pixmap_from_display_list(list, ctm, pix):
31592: (4)              r""" Class-aware wrapper for `::fz_fill_pixmap_from_display_list()`."""
31593: (4)              return _mupdf.fz_fill_pixmap_from_display_list(list, ctm, pix)
31594: (0)          def fz_fill_pixmap_with_color(pix, colorspace, color, color_params):
31595: (4)              r"""
31596: (4)              Class-aware wrapper for `::fz_fill_pixmap_with_color()`.
31597: (4)              This function has out-params. Python/C# wrappers look like:
31598: (8)                  `fz_fill_pixmap_with_color(::fz_pixmap *pix, ::fz_colorspace
*colorspace, ::fz_color_params color_params)` => float color
31599: (8)                  Fill pixmap with solid color.
31600: (4)              """
31601: (4)              return _mupdf.fz_fill_pixmap_with_color(pix, colorspace, color,
color_params)
31602: (0)          def fz_fill_shade(dev, shade, ctm, alpha, color_params):
31603: (4)              r""" Class-aware wrapper for `::fz_fill_shade()`."""
31604: (4)              return _mupdf.fz_fill_shade(dev, shade, ctm, alpha, color_params)
31605: (0)          def fz_fill_text(dev, text, ctm, colorspace, color, alpha, color_params):
31606: (4)              r""" Class-aware wrapper for `::fz_fill_text()`."""
31607: (4)              return _mupdf.fz_fill_text(dev, text, ctm, colorspace, color, alpha,
color_params)
31608: (0)          def fz_filter_store(fn, arg, type):
31609: (4)              r"""
31610: (4)              Class-aware wrapper for `::fz_filter_store()`.
31611: (8)                  Filter every element in the store with a matching type with the
31612: (8)                  given function.
31613: (8)                  If the function returns 1 for an element, drop the element.
31614: (4)              """
31615: (4)              return _mupdf.fz_filter_store(fn, arg, type)
31616: (0)          def fz_find_item(drop, key, type):
31617: (4)              r"""
31618: (4)              Class-aware wrapper for `::fz_find_item()`.
31619: (8)                  Find an item within the store.
31620: (8)                  drop: The function used to free the value (to ensure we get a
31621: (8)                  value of the correct type).
31622: (8)                  key: The key used to index the item.
31623: (8)                  type: Functions used to manipulate the key.
31624: (8)                  Returns NULL for not found, otherwise returns a pointer to the
31625: (8)                  value indexed by key to which a reference has been taken.
31626: (4)              """
31627: (4)              return _mupdf.fz_find_item(drop, key, type)
31628: (0)          def fz_flush_output(out):
31629: (4)              r"""
31630: (4)              Class-aware wrapper for `::fz_flush_output()`.
31631: (8)                  Flush unwritten data.
31632: (4)              """
31633: (4)              return _mupdf.fz_flush_output(out)
31634: (0)          def fz_flush_warnings():
31635: (4)              r"""
31636: (4)              Class-aware wrapper for `::fz_flush_warnings()`.
31637: (8)                  Flush any repeated warnings.
31638: (8)                  Repeated warnings are buffered, counted and eventually printed
31639: (8)                  along with the number of repetitions. Call fz_flush_warnings
31640: (8)                  to force printing of the latest buffered warning and the
31641: (8)                  number of repetitions, for example to make sure that all
31642: (8)                  warnings are printed before exiting an application.
31643: (4)              """
31644: (4)              return _mupdf.fz_flush_warnings()
31645: (0)          def fz_font_ascender(font):
31646: (4)              r"""
```

```
31647: (4)             Class-aware wrapper for `::fz_font_ascender()`.
31648: (8)                 Retrieve font ascender in ems.
31649: (4)             """
31650: (4)             return _mupdf.fz_font_ascender(font)
31651: (0)         def fz_font_bbox(font):
31652: (4)             r"""
31653: (4)             Class-aware wrapper for `::fz_font_bbox()`.
31654: (8)                 Retrieve the font bbox.
31655: (8)                 font: The font to query.
31656: (8)                 Returns the font bbox by value; it is valid only if
31657: (8)                 fz_font_flags(font)->invalid_bbox is zero.
31658: (4)             """
31659: (4)             return _mupdf.fz_font_bbox(font)
31660: (0)         def fz_font_descender(font):
31661: (4)             r"""
31662: (4)             Class-aware wrapper for `::fz_font_descender()`.
31663: (8)                 Retrieve font descender in ems.
31664: (4)             """
31665: (4)             return _mupdf.fz_font_descender(font)
31666: (0)         def fz_font_digest(font, digest):
31667: (4)             r"""
31668: (4)             Class-aware wrapper for `::fz_font_digest()`.
31669: (8)                 Retrieve the MD5 digest for the font's data.
31670: (4)             """
31671: (4)             return _mupdf.fz_font_digest(font, digest)
31672: (0)         def fz_font_ft_face(font):
31673: (4)             r"""
31674: (4)             Class-aware wrapper for `::fz_font_ft_face()`.
31675: (8)                 Retrieve the FT_Face handle
31676: (8)                 for the font.
31677: (8)                 font: The font to query
31678: (8)                 Returns the FT_Face handle for the font, or NULL
31679: (8)                 if not a freetype handled font. (Cast to void *
31680: (8)                 to avoid nasty header exposure).
31681: (4)             """
31682: (4)             return _mupdf.fz_font_ft_face(font)
31683: (0)         def fz_font_is_bold(font):
31684: (4)             r"""
31685: (4)             Class-aware wrapper for `::fz_font_is_bold()`.
31686: (8)                 Query whether the font flags say that this font is bold.
31687: (4)             """
31688: (4)             return _mupdf.fz_font_is_bold(font)
31689: (0)         def fz_font_is_italic(font):
31690: (4)             r"""
31691: (4)             Class-aware wrapper for `::fz_font_is_italic()`.
31692: (8)                 Query whether the font flags say that this font is italic.
31693: (4)             """
31694: (4)             return _mupdf.fz_font_is_italic(font)
31695: (0)         def fz_font_is_monospaced(font):
31696: (4)             r"""
31697: (4)             Class-aware wrapper for `::fz_font_is_monospaced()`.
31698: (8)                 Query whether the font flags say that this font is monospaced.
31699: (4)             """
31700: (4)             return _mupdf.fz_font_is_monospaced(font)
31701: (0)         def fz_font_is_serif(font):
31702: (4)             r"""
31703: (4)             Class-aware wrapper for `::fz_font_is_serif()`.
31704: (8)                 Query whether the font flags say that this font is serif.
31705: (4)             """
31706: (4)             return _mupdf.fz_font_is_serif(font)
31707: (0)         def fz_font_name(font):
31708: (4)             r"""
31709: (4)             Class-aware wrapper for `::fz_font_name()`.
31710: (8)                 Retrieve a pointer to the name of the font.
31711: (8)                 font: The font to query.
31712: (8)                 Returns a pointer to an internal copy of the font name.
31713: (8)                 Will never be NULL, but may be the empty string.
31714: (4)             """
31715: (4)             return _mupdf.fz_font_name(font)
```

```
31716: (0)          def fz_font_t3_procs(font):
31717: (4)              r"""
31718: (4)              Class-aware wrapper for `::fz_font_t3_procs()`.
31719: (8)                  Retrieve the Type3 procs
31720: (8)                  for a font.
31721: (8)                  font: The font to query
31722: (8)                  Returns the t3_procs pointer. Will be NULL for a
31723: (8)                  non type-3 font.
31724: (4)              """
31725: (4)              return _mupdf.fz_font_t3_procs(font)
31726: (0)          def fz_format_double(fmt, value):
31727: (4)              r"""
31728: (4)              Class-aware wrapper for `::fz_format_double()`.   Swig-friendly and
typesafe way to do fz_snprintf(fmt, value). `fmt`
31729: (4)              must end with one of 'efg' otherwise we throw an exception.
31730: (4)              """
31731: (4)              return _mupdf.fz_format_double(fmt, value)
31732: (0)          def fz_format_link_uri(doc, dest):
31733: (4)              r"""
31734: (4)              Class-aware wrapper for `::fz_format_link_uri()`.
31735: (8)                  Format an internal link to a page number, location, and possible
viewing parameters,
31736: (8)                  suitable for use with fz_create_link.
31737: (8)                  Returns a newly allocated string that the caller must free.
31738: (4)              """
31739: (4)              return _mupdf.fz_format_link_uri(doc, dest)
31740: (0)          def fz_format_output_path(path, size, fmt, page):
31741: (4)              r"""
31742: (4)              Class-aware wrapper for `::fz_format_output_path()`.
31743: (8)                  create output file name using a template.
31744: (8)                  If the path contains %[0-9]*d, the first such pattern will be
31745: (8)                  replaced with the page number. If the template does not contain
31746: (8)                  such a pattern, the page number will be inserted before the
31747: (8)                  filename extension. If the template does not have a filename
31748: (8)                  extension, the page number will be added to the end.
31749: (4)              """
31750: (4)              return _mupdf.fz_format_output_path(path, size, fmt, page)
31751: (0)          def fz_free(p):
31752: (4)              r"""
31753: (4)              Class-aware wrapper for `::fz_free()`.
31754: (8)                  Free a previously allocated block of memory.
31755: (8)                  fz_free(ctx, NULL) does nothing.
31756: (8)                  Never throws exceptions.
31757: (4)              """
31758: (4)              return _mupdf.fz_free(p)
31759: (0)          def fz_ft_lock():
31760: (4)              r""" Class-aware wrapper for `::fz_ft_lock()`."""
31761: (4)              return _mupdf.fz_ft_lock()
31762: (0)          def fz_ft_lock_held():
31763: (4)              r""" Class-aware wrapper for `::fz_ft_lock_held()`."""
31764: (4)              return _mupdf.fz_ft_lock_held()
31765: (0)          def fz_ft_unlock():
31766: (4)              r""" Class-aware wrapper for `::fz_ft_unlock()`."""
31767: (4)              return _mupdf.fz_ft_unlock()
31768: (0)          def fz_function_size(func):
31769: (4)              r""" Class-aware wrapper for `::fz_function_size()`."""
31770: (4)              return _mupdf.fz_function_size(func)
31771: (0)          def fz_gamma_pixmap(pix, gamma):
31772: (4)              r"""
31773: (4)              Class-aware wrapper for `::fz_gamma_pixmap()`.
31774: (8)                  Apply gamma correction to a pixmap. All components
31775: (8)                  of all pixels are modified (except alpha, which is unchanged).
31776: (8)                  gamma: The gamma value to apply; 1.0 for no change.
31777: (4)              """
31778: (4)              return _mupdf.fz_gamma_pixmap(pix, gamma)
31779: (0)          def fz_generate_transition(tpix, opix, npix, time, trans):
31780: (4)              r"""
31781: (4)              Class-aware wrapper for `::fz_generate_transition()`.
31782: (8)                  Generate a frame of a transition.
```

```
31783: (8)                  tpix: Target pixmap
31784: (8)                  opix: Old pixmap
31785: (8)                  npix: New pixmap
31786: (8)                  time: Position within the transition (0 to 256)
31787: (8)                  trans: Transition details
31788: (8)                  Returns 1 if successfully generated a frame.
31789: (8)                  Note: Pixmaps must include alpha.
31790: (4)              """
31791: (4)              return _mupdf.fz_generate_transition(tpix, opix, npix, time, trans)
31792: (0)          def fz_get_glyph_name(font, glyph, buf, size):
31793: (4)              r"""
31794: (4)              Class-aware wrapper for `::fz_get_glyph_name()`.
31795: (8)                  Find the name of a glyph
31796: (8)                  font: The font to look for the glyph in.
31797: (8)                  glyph: The glyph id to look for.
31798: (8)                  buf: Pointer to a buffer for the name to be inserted into.
31799: (8)                  size: The size of the buffer.
31800: (8)                  If a font contains a name table, then the name of the glyph
31801: (8)                  will be returned in the supplied buffer. Otherwise a name
31802: (8)                  is synthesised. The name will be truncated to fit in
31803: (8)                  the buffer.
31804: (4)              """
31805: (4)              return _mupdf.fz_get_glyph_name(font, glyph, buf, size)
31806: (0)          def fz_get_glyph_name2(font, glyph):
31807: (4)              r"""
31808: (4)              Class-aware wrapper for `::fz_get_glyph_name2()`.
31809: (4)              C++ alternative to fz_get_glyph_name() that returns information in a
std::string.
31810: (4)              """
31811: (4)              return _mupdf.fz_get_glyph_name2(font, glyph)
31812: (0)          def fz_get_pixmap_from_image(image, subarea, ctm, w, h):
31813: (4)              r"""
31814: (4)              Class-aware wrapper for `::fz_get_pixmap_from_image()`.
31815: (4)              This function has out-params. Python/C# wrappers look like:
31816: (8)                  `fz_get_pixmap_from_image(::fz_image *image, const ::fz_irect
*subarea, ::fz_matrix *ctm)` => `(fz_pixmap *, int w, int h)`
31817: (8)                  Called to get a handle to a pixmap from an image.
31818: (8)                  image: The image to retrieve a pixmap from.
31819: (8)                  subarea: The subarea of the image that we actually care about
31820: (8)                  (or NULL to indicate the whole image).
31821: (8)                  ctm: Optional, unless subarea is given. If given, then on
31822: (8)                  entry this is the transform that will be applied to the complete
31823: (8)                  image. It should be updated on exit to the transform to apply to
31824: (8)                  the given subarea of the image. This is used to calculate the
31825: (8)                  desired width/height for subsampling.
31826: (8)                  w: If non-NULL, a pointer to an int to be updated on exit to the
31827: (8)                  width (in pixels) that the scaled output will cover.
31828: (8)                  h: If non-NULL, a pointer to an int to be updated on exit to the
31829: (8)                  height (in pixels) that the scaled output will cover.
31830: (8)                  Returns a non NULL kept pixmap pointer. May throw exceptions.
31831: (4)              """
31832: (4)              return _mupdf.fz_get_pixmap_from_image(image, subarea, ctm, w, h)
31833: (0)          def fz_get_unscaled_pixmap_from_image(image):
31834: (4)              r"""
31835: (4)              Class-aware wrapper for `::fz_get_unscaled_pixmap_from_image()`.
31836: (8)                  Calls fz_get_pixmap_from_image() with ctm, subarea, w and h all set to
NULL.
31837: (4)              """
31838: (4)              return _mupdf.fz_get_unscaled_pixmap_from_image(image)
31839: (0)          def fz_getopt(nargc, nargv, ostr):
31840: (4)              r"""
31841: (4)              Class-aware wrapper for `::fz_getopt()`.
31842: (4)              This function has out-params. Python/C# wrappers look like:
31843: (8)                  `fz_getopt(int nargc, const char *ostr)` => `(int, char *nargv)`
31844: (8)                  Identical to fz_getopt_long, but with a NULL longopts field,
signifying no long
31845: (8)                  options.
31846: (4)              """
31847: (4)              return _mupdf.fz_getopt(nargc, nargv, ostr)
```

```
31848: (0)          def fz_getopt_long(nargc, nargv, ostr, longopts):
31849: (4)              r"""
31850: (4)              Class-aware wrapper for `::fz_getopt_long()`.
31851: (4)              This function has out-params. Python/C# wrappers look like:
31852: (8)                  `fz_getopt_long(int nargc, const char *ostr, const
::fz_getopt_long_options *longopts)` => `(int, char *nargv)`
31853: (8)                  Simple functions/variables for use in tools.
31854: (8)                  ostr = option string. Comprises single letter options, followed by :
if there
31855: (8)                  is an argument to the option.
31856: (8)                  longopts: NULL (indicating no long options), or a pointer to an array
of
31857: (8)                  longoptions, terminated by an entry with option == NULL.
31858: (8)                  In the event of matching a single char option, this function will
normally
31859: (8)                  return the char. The exception to this is when the option requires an
31860: (8)                  argument and none is supplied; in this case we return ':'.
31861: (8)                  In the event of matching a long option, this function returns 0, with
fz_optlong
31862: (8)                  set to point to the matching option.
31863: (8)                  A long option entry may be followed with : to indicate there is an
argument to the
31864: (8)                  option. If the need for an argument is specified in this way, and no
argument is
31865: (8)                  given, an error will be displayed and argument processing will stop.
If an argument
31866: (8)                  is given, and the long option record contains a non-null flag pointer,
then the code
31867: (8)                  will decode the argument and fill in that flag pointer. Specifically,
31868: (8)                  case-insensitive matches to 'yes', 'no', 'true' and 'false' will cause
a value of 0
31869: (8)                  or 1 as appropriate to be written; failing this the arg will be
interpreted as a
31870: (8)                  decimal integer using atoi.
31871: (8)                  A long option entry may be followed by an list of options (e.g.
myoption=foo|bar|baz)
31872: (8)                  and the option will be passed to fz_opt_from_list. The return value of
that will be
31873: (8)                  placed in fz_optitem. If the return value of that function is -1, then
an error will
31874: (8)                  be displayed and argument processing will stop.
31875: (8)                  In the event of reaching the end of the arg list or '--', this
function returns EOF.
31876: (8)                  In the event of failing to match anything, an error is printed, and we
return '?'.
31877: (8)                  If an argument is expected for the option, then fz_optarg will be
returned pointing
31878: (8)                  at the start of the argument. Examples of supported argument formats:
'-r500', '-r 500',
31879: (8)                  '--resolution 500', '--resolution=500'.
31880: (4)              """
31881: (4)              return _mupdf.fz_getopt_long(nargc, nargv, ostr, longopts)
31882: (0)          def fz_glyph_bbox(glyph):
31883: (4)              r"""
31884: (4)              Class-aware wrapper for `::fz_glyph_bbox()`.
31885: (8)                  Return the bounding box of the glyph in pixels.
31886: (4)              """
31887: (4)              return _mupdf.fz_glyph_bbox(glyph)
31888: (0)          def fz_glyph_bbox_no_ctx(src):
31889: (4)              r""" Class-aware wrapper for `::fz_glyph_bbox_no_ctx()`."""
31890: (4)              return _mupdf.fz_glyph_bbox_no_ctx(src)
31891: (0)          def fz_glyph_cacheable(font, gid):
31892: (4)              r"""
31893: (4)              Class-aware wrapper for `::fz_glyph_cacheable()`.
31894: (8)                  Determine if a given glyph in a font
31895: (8)                  is cacheable. Certain glyphs in a type 3 font cannot safely
31896: (8)                  be cached, as their appearance depends on the enclosing
31897: (8)                  graphic state.
31898: (8)                  font: The font to look for the glyph in.
```

```
31899: (8)                  gif: The glyph to query.
31900: (8)                  Returns non-zero if cacheable, 0 if not.
31901: (4)              """
31902: (4)              return _mupdf.fz_glyph_cacheable(font, gid)
31903: (0)          def fz_glyph_height(glyph):
31904: (4)              r"""
31905: (4)              Class-aware wrapper for `::fz_glyph_height()`.
31906: (8)                  Return the height of the glyph in pixels.
31907: (4)              """
31908: (4)              return _mupdf.fz_glyph_height(glyph)
31909: (0)          def fz_glyph_name_from_unicode_sc(unicode):
31910: (4)              r""" Class-aware wrapper for `::fz_glyph_name_from_unicode_sc()`."""
31911: (4)              return _mupdf.fz_glyph_name_from_unicode_sc(unicode)
31912: (0)          def fz_glyph_width(glyph):
31913: (4)              r"""
31914: (4)              Class-aware wrapper for `::fz_glyph_width()`.
31915: (8)                  Return the width of the glyph in pixels.
31916: (4)              """
31917: (4)              return _mupdf.fz_glyph_width(glyph)
31918: (0)          def fz_graphics_aa_level():
31919: (4)              r"""
31920: (4)              Class-aware wrapper for `::fz_graphics_aa_level()`.
31921: (8)                  Get the number of bits of antialiasing we are
31922: (8)                  using for graphics. Between 0 and 8.
31923: (4)              """
31924: (4)              return _mupdf.fz_graphics_aa_level()
31925: (0)          def fz_graphics_min_line_width():
31926: (4)              r"""
31927: (4)              Class-aware wrapper for `::fz_graphics_min_line_width()`.
31928: (8)                  Get the minimum line width to be
31929: (8)                  used for stroked lines.
31930: (8)                  min_line_width: The minimum line width to use (in pixels).
31931: (4)              """
31932: (4)              return _mupdf.fz_graphics_min_line_width()
31933: (0)          def fz_gridfit_matrix(as_tiled, m):
31934: (4)              r"""
31935: (4)              Class-aware wrapper for `::fz_gridfit_matrix()`.
31936: (8)                  Grid fit a matrix.
31937: (8)                  as_tiled = 0 => adjust the matrix so that the image of the unit
31938: (8)                  square completely covers any pixel that was touched by the
31939: (8)                  image of the unit square under the original matrix.
31940: (8)                  as_tiled = 1 => adjust the matrix so that the corners of the
31941: (8)                  image of the unit square align with the closest integer corner
31942: (8)                  of the image of the unit square under the original matrix.
31943: (4)              """
31944: (4)              return _mupdf.fz_gridfit_matrix(as_tiled, m)
31945: (0)          def fz_grisu(f, s, exp):
31946: (4)              r"""
31947: (4)              Class-aware wrapper for `::fz_grisu()`.
31948: (4)              This function has out-params. Python/C# wrappers look like:
31949: (8)                  `fz_grisu(float f, char *s)` => `(int, int exp)`
31950: (4)              """
31951: (4)              return _mupdf.fz_grisu(f, s, exp)
31952: (0)          def fz_grow_buffer(buf):
31953: (4)              r"""
31954: (4)              Class-aware wrapper for `::fz_grow_buffer()`.
31955: (8)                  Make some space within a buffer (i.e. ensure that
31956: (8)                  capacity > size).
31957: (4)              """
31958: (4)              return _mupdf.fz_grow_buffer(buf)
31959: (0)          def fz_has_archive_entry(arch, name):
31960: (4)              r"""
31961: (4)              Class-aware wrapper for `::fz_has_archive_entry()`.
31962: (8)                  Check if entry by given name exists.
31963: (8)                  If named entry does not exist 0 will be returned, if it does
31964: (8)                  exist 1 is returned.
31965: (8)                  name: Entry name to look for, this must be an exact match to
31966: (8)                  the entry name in the archive.
31967: (4)              """
```

```
31968: (4)             return _mupdf.fz_has_archive_entry(arch, name)
31969: (0)         def fz_has_option(opts, key, val):
31970: (4)             r"""
31971: (4)             Class-aware wrapper for `::fz_has_option()`.
31972: (4)             This function has out-params. Python/C# wrappers look like:
31973: (8)                 `fz_has_option(const char *opts, const char *key)` => `(int, const
char *val)`
31974: (8)                 Look for a given option (key) in the opts string. Return 1 if
31975: (8)                 it has it, and update *val to point to the value within opts.
31976: (4)             """
31977: (4)             return _mupdf.fz_has_option(opts, key, val)
31978: (0)         def fz_has_permission(doc, p):
31979: (4)             r"""
31980: (4)             Class-aware wrapper for `::fz_has_permission()`.
31981: (8)                 Check permission flags on document.
31982: (4)             """
31983: (4)             return _mupdf.fz_has_permission(doc, p)
31984: (0)         def fz_hash_filter(table, state, callback):
31985: (4)             r"""
31986: (4)             Class-aware wrapper for `::fz_hash_filter()`.
31987: (8)                 Iterate over the entries in a hash table, removing all the ones where
callback returns true.
31988: (8)                 Does NOT free the value of the entry, so the caller is expected to
take care of this.
31989: (4)             """
31990: (4)             return _mupdf.fz_hash_filter(table, state, callback)
31991: (0)         def fz_hash_find(table, key):
31992: (4)             r"""
31993: (4)             Class-aware wrapper for `::fz_hash_find()`.
31994: (8)                 Search for a matching hash within the table, and return the
31995: (8)                 associated value.
31996: (4)             """
31997: (4)             return _mupdf.fz_hash_find(table, key)
31998: (0)         def fz_hash_for_each(table, state, callback):
31999: (4)             r"""
32000: (4)             Class-aware wrapper for `::fz_hash_for_each()`.
32001: (8)                 Iterate over the entries in a hash table.
32002: (4)             """
32003: (4)             return _mupdf.fz_hash_for_each(table, state, callback)
32004: (0)         def fz_hash_insert(table, key, val):
32005: (4)             r"""
32006: (4)             Class-aware wrapper for `::fz_hash_insert()`.
32007: (8)                 Insert a new key/value pair into the hash table.
32008: (8)                 If an existing entry with the same key is found, no change is
32009: (8)                 made to the hash table, and a pointer to the existing value is
32010: (8)                 returned.
32011: (8)                 If no existing entry with the same key is found, ownership of
32012: (8)                 val passes in, key is copied, and NULL is returned.
32013: (4)             """
32014: (4)             return _mupdf.fz_hash_insert(table, key, val)
32015: (0)         def fz_hash_remove(table, key):
32016: (4)             r"""
32017: (4)             Class-aware wrapper for `::fz_hash_remove()`.
32018: (8)                 Remove the entry for a given key.
32019: (8)                 The value is NOT freed, so the caller is expected to take care
32020: (8)                 of this.
32021: (4)             """
32022: (4)             return _mupdf.fz_hash_remove(table, key)
32023: (0)         def fz_hb_lock():
32024: (4)             r"""
32025: (4)             Class-aware wrapper for `::fz_hb_lock()`.
32026: (8)                 Lock against Harfbuzz being called
32027: (8)                 simultaneously in several threads. This reuses
32028: (8)                 FZ_LOCK_FREETYPE.
32029: (4)             """
32030: (4)             return _mupdf.fz_hb_lock()
32031: (0)         def fz_hb_unlock():
32032: (4)             r"""
32033: (4)             Class-aware wrapper for `::fz_hb_unlock()`.
```

```
32034: (8)                 Unlock after a Harfbuzz call. This reuses
32035: (8)                 FZ_LOCK_FREETYPE.
32036: (4)             """
32037: (4)             return _mupdf.fz_hb_unlock()
32038: (0)         def fz_highlight_selection(page, a, b, quads, max_quads):
32039: (4)             r"""
32040: (4)             Class-aware wrapper for `::fz_highlight_selection()`.
32041: (8)                 Return a list of quads to highlight lines inside the selection
32042: (8)                 points.
32043: (4)             """
32044: (4)             return _mupdf.fz_highlight_selection(page, a, b, quads, max_quads)
32045: (0)         def fz_highlight_selection2(page, a, b, max_quads):
32046: (4)             r"""
32047: (4)             Class-aware wrapper for `::fz_highlight_selection2()`.
32048: (4)             C++ alternative to fz_highlight_selection() that returns quads in a
32049: (4)             std::vector.
32050: (4)             """
32051: (4)             return _mupdf.fz_highlight_selection2(page, a, b, max_quads)
32052: (0)         def fz_ignore_error():
32053: (4)             r""" Class-aware wrapper for `::fz_ignore_error()`."""
32054: (4)             return _mupdf.fz_ignore_error()
32055: (0)         def fz_ignore_text(dev, text, ctm):
32056: (4)             r""" Class-aware wrapper for `::fz_ignore_text()`."""
32057: (4)             return _mupdf.fz_ignore_text(dev, text, ctm)
32058: (0)         def fz_image_orientation(image):
32059: (4)             r"""
32060: (4)             Class-aware wrapper for `::fz_image_orientation()`.
32061: (8)                 Request the natural orientation of an image.
32062: (8)                 This is for images (such as JPEG) that can contain internal
32063: (8)                 specifications of rotation/flips. This is ignored by all the
32064: (8)                 internal decode/rendering routines, but can be used by callers
32065: (8)                 (such as the image document handler) to respect such
32066: (8)                 specifications.
32067: (8)                 The values used by MuPDF are as follows, with the equivalent
32068: (8)                 Exif specifications given for information:
32069: (8)                 0: Undefined
32070: (8)                 1: 0 degree ccw rotation. (Exif = 1)
32071: (8)                 2: 90 degree ccw rotation. (Exif = 8)
32072: (8)                 3: 180 degree ccw rotation. (Exif = 3)
32073: (8)                 4: 270 degree ccw rotation. (Exif = 6)
32074: (8)                 5: flip on X. (Exif = 2)
32075: (8)                 6: flip on X, then rotate ccw by 90 degrees. (Exif = 5)
32076: (8)                 7: flip on X, then rotate ccw by 180 degrees. (Exif = 4)
32077: (8)                 8: flip on X, then rotate ccw by 270 degrees. (Exif = 7)
32078: (4)             """
32079: (4)             return _mupdf.fz_image_orientation(image)
32080: (0)         def fz_image_orientation_matrix(image):
32081: (4)             r""" Class-aware wrapper for `::fz_image_orientation_matrix()`."""
32082: (4)             return _mupdf.fz_image_orientation_matrix(image)
32083: (0)         def fz_image_resolution(image, xres, yres):
32084: (4)             r"""
32085: (4)             Class-aware wrapper for `::fz_image_resolution()`.
32086: (4)             This function has out-params. Python/C# wrappers look like:
32087: (8)                 `fz_image_resolution(::fz_image *image)` => `(int xres, int yres)`
32088: (8)                 Request the natural resolution
32089: (8)                 of an image.
32090: (8)                 xres, yres: Pointers to ints to be updated with the
32091: (8)                 natural resolution of an image (or a sensible default
32092: (8)                 if not encoded).
32093: (4)             """
32094: (4)             return _mupdf.fz_image_resolution(image, xres, yres)
32095: (0)         def fz_image_size(im):
32096: (4)             r"""
32097: (4)             Class-aware wrapper for `::fz_image_size()`.
32098: (8)                 Return the size of the storage used by an image.
32099: (4)             """
32100: (4)             return _mupdf.fz_image_size(im)
32101: (0)         def fz_image_type_name(type):
32102: (4)             r"""
```

```
32103: (4)             Class-aware wrapper for `::fz_image_type_name()`.
32104: (8)                 Map from FZ_IMAGE_* value to string.
32105: (8)                 The returned string is static and therefore must not be freed.
32106: (4)             """
32107: (4)             return _mupdf.fz_image_type_name(type)
32108: (0)         def fz_include_point_in_rect(r, p):
32109: (4)             r"""
32110: (4)             Class-aware wrapper for `::fz_include_point_in_rect()`.
32111: (8)                 Expand a bbox to include a given point.
32112: (8)                 To create a rectangle that encompasses a sequence of points, the
32113: (8)                 rectangle must first be set to be the empty rectangle at one of
32114: (8)                 the points before including the others.
32115: (4)             """
32116: (4)             return _mupdf.fz_include_point_in_rect(r, p)
32117: (0)         def fz_init_text_decoder(dec, encoding):
32118: (4)             r""" Class-aware wrapper for `::fz_init_text_decoder()`."""
32119: (4)             return _mupdf.fz_init_text_decoder(dec, encoding)
32120: (0)         def fz_install_load_system_font_funcs(f, f_cjk, f_fallback):
32121: (4)             r"""
32122: (4)             Class-aware wrapper for `::fz_install_load_system_font_funcs()`.
32123: (8)                 Install functions to allow MuPDF to request fonts from the
32124: (8)                 system.
32125: (8)                 Only one set of hooks can be in use at a time.
32126: (4)             """
32127: (4)             return _mupdf.fz_install_load_system_font_funcs(f, f_cjk, f_fallback)
32128: (0)         def fz_install_load_system_font_funcs2(args):
32129: (4)             r"""
32130: (4)             Class-aware wrapper for `::fz_install_load_system_font_funcs2()`.
32131: (4)             Alternative to fz_install_load_system_font_funcs() that takes args in a
32132: (4)             struct, to allow use from Python/C# via Swig Directors.
32133: (4)             """
32134: (4)             return _mupdf.fz_install_load_system_font_funcs2(args)
32135: (0)         def fz_int2_heap_insert(heap, v):
32136: (4)             r""" Class-aware wrapper for `::fz_int2_heap_insert()`."""
32137: (4)             return _mupdf.fz_int2_heap_insert(heap, v)
32138: (0)         def fz_int2_heap_sort(heap):
32139: (4)             r""" Class-aware wrapper for `::fz_int2_heap_sort()`."""
32140: (4)             return _mupdf.fz_int2_heap_sort(heap)
32141: (0)         def fz_int2_heap_uniq(heap):
32142: (4)             r""" Class-aware wrapper for `::fz_int2_heap_uniq()`."""
32143: (4)             return _mupdf.fz_int2_heap_uniq(heap)
32144: (0)         def fz_int_heap_insert(heap, v):
32145: (4)             r""" Class-aware wrapper for `::fz_int_heap_insert()`."""
32146: (4)             return _mupdf.fz_int_heap_insert(heap, v)
32147: (0)         def fz_int_heap_sort(heap):
32148: (4)             r""" Class-aware wrapper for `::fz_int_heap_sort()`."""
32149: (4)             return _mupdf.fz_int_heap_sort(heap)
32150: (0)         def fz_int_heap_uniq(heap):
32151: (4)             r""" Class-aware wrapper for `::fz_int_heap_uniq()`."""
32152: (4)             return _mupdf.fz_int_heap_uniq(heap)
32153: (0)         def fz_intersect_irect(a, b):
32154: (4)             r"""
32155: (4)             Class-aware wrapper for `::fz_intersect_irect()`.
32156: (8)                 Compute intersection of two bounding boxes.
32157: (8)                 Similar to fz_intersect_rect but operates on two bounding
32158: (8)                 boxes instead of two rectangles.
32159: (4)             """
32160: (4)             return _mupdf.fz_intersect_irect(a, b)
32161: (0)         def fz_intersect_rect(a, b):
32162: (4)             r"""
32163: (4)             Class-aware wrapper for `::fz_intersect_rect()`.
32164: (8)                 Compute intersection of two rectangles.
32165: (8)                 Given two rectangles, update the first to be the smallest
32166: (8)                 axis-aligned rectangle that covers the area covered by both
32167: (8)                 given rectangles. If either rectangle is empty then the
32168: (8)                 intersection is also empty. If either rectangle is infinite
32169: (8)                 then the intersection is simply the non-infinite rectangle.
32170: (8)                 Should both rectangles be infinite, then the intersection is
32171: (8)                 also infinite.
```

```
32172: (4)             """
32173: (4)             return _mupdf.fz_intersect_rect(a, b)
32174: (0)         def fz_intptr_heap_insert(heap, v):
32175: (4)             r""" Class-aware wrapper for `::fz_intptr_heap_insert()`."""
32176: (4)             return _mupdf.fz_intptr_heap_insert(heap, v)
32177: (0)         def fz_intptr_heap_sort(heap):
32178: (4)             r""" Class-aware wrapper for `::fz_intptr_heap_sort()`."""
32179: (4)             return _mupdf.fz_intptr_heap_sort(heap)
32180: (0)         def fz_intptr_heap_uniq(heap):
32181: (4)             r""" Class-aware wrapper for `::fz_intptr_heap_uniq()`."""
32182: (4)             return _mupdf.fz_intptr_heap_uniq(heap)
32183: (0)         def fz_invert_bitmap(bmp):
32184: (4)             r"""
32185: (4)             Class-aware wrapper for `::fz_invert_bitmap()`.
32186: (8)                 Invert bitmap.
32187: (8)                 Never throws exceptions.
32188: (4)             """
32189: (4)             return _mupdf.fz_invert_bitmap(bmp)
32190: (0)         def fz_invert_matrix(matrix):
32191: (4)             r"""
32192: (4)             Class-aware wrapper for `::fz_invert_matrix()`.
32193: (8)                 Create an inverse matrix.
32194: (8)                 inverse: Place to store inverse matrix.
32195: (8)                 matrix: Matrix to invert. A degenerate matrix, where the
32196: (8)                 determinant is equal to zero, can not be inverted and the
32197: (8)                 original matrix is returned instead.
32198: (8)                 Returns inverse.
32199: (4)             """
32200: (4)             return _mupdf.fz_invert_matrix(matrix)
32201: (0)         def fz_invert_pixmap(pix):
32202: (4)             r"""
32203: (4)             Class-aware wrapper for `::fz_invert_pixmap()`.
32204: (8)                 Invert all the pixels in a pixmap. All components (process and
32205: (8)                 spots) of all pixels are inverted (except alpha, which is
32206: (8)                 unchanged).
32207: (4)             """
32208: (4)             return _mupdf.fz_invert_pixmap(pix)
32209: (0)         def fz_invert_pixmap_alpha(pix):
32210: (4)             r"""
32211: (4)             Class-aware wrapper for `::fz_invert_pixmap_alpha()`.
32212: (8)                 Invert the alpha fo all the pixels in a pixmap.
32213: (4)             """
32214: (4)             return _mupdf.fz_invert_pixmap_alpha(pix)
32215: (0)         def fz_invert_pixmap_luminance(pix):
32216: (4)             r"""
32217: (4)             Class-aware wrapper for `::fz_invert_pixmap_luminance()`.
32218: (8)                 Transform the pixels in a pixmap so that luminance of each
32219: (8)                 pixel is inverted, and the chrominance remains unchanged (as
32220: (8)                 much as accuracy allows).
32221: (8)                 All components of all pixels are inverted (except alpha, which
32222: (8)                 is unchanged). Only supports Grey and RGB bitmaps.
32223: (4)             """
32224: (4)             return _mupdf.fz_invert_pixmap_luminance(pix)
32225: (0)         def fz_invert_pixmap_raw(pix):
32226: (4)             r"""
32227: (4)             Class-aware wrapper for `::fz_invert_pixmap_raw()`.
32228: (8)                 Invert all the pixels in a non-premultiplied pixmap in a
32229: (8)                 very naive manner.
32230: (4)             """
32231: (4)             return _mupdf.fz_invert_pixmap_raw(pix)
32232: (0)         def fz_invert_pixmap_rect(image, rect):
32233: (4)             r"""
32234: (4)             Class-aware wrapper for `::fz_invert_pixmap_rect()`.
32235: (8)                 Invert all the pixels in a given rectangle of a (premultiplied)
32236: (8)                 pixmap. All components of all pixels in the rectangle are
32237: (8)                 inverted (except alpha, which is unchanged).
32238: (4)             """
32239: (4)             return _mupdf.fz_invert_pixmap_rect(image, rect)
32240: (0)         def fz_irect_from_rect(rect):
```

```
(4)             r"""
(4)             Class-aware wrapper for `::fz_irect_from_rect()`.
(8)                 Convert a rect into the minimal bounding box
(8)                 that covers the rectangle.
(8)                 Coordinates in a bounding box are integers, so rounding of the
(8)                 rects coordinates takes place. The top left corner is rounded
(8)                 upwards and left while the bottom right corner is rounded
(8)                 downwards and to the right.
(4)             """
(4)             return _mupdf.fz_irect_from_rect(rect)
(0)         def fz_irect_height(r):
(4)             r"""
(4)             Class-aware wrapper for `::fz_irect_height()`.
(8)                 Return the height of an irect. Invalid irects return 0.
(4)             """
(4)             return _mupdf.fz_irect_height(r)
(0)         def fz_irect_width(r):
(4)             r"""
(4)             Class-aware wrapper for `::fz_irect_width()`.
(8)                 Return the width of an irect. Invalid irects return 0.
(4)             """
(4)             return _mupdf.fz_irect_width(r)
(0)         def fz_is_cfb_archive(file):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_cfb_archive()`.
(8)                 Detect if stream object is a cfb archive.
(8)                 Assumes that the stream object is seekable.
(4)             """
(4)             return _mupdf.fz_is_cfb_archive(file)
(0)         def fz_is_directory(path):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_directory()`.
(8)                 Determine if a given path is a directory.
(8)                 In the case of the path not existing, or having no access
(8)                 we will return 0.
(4)             """
(4)             return _mupdf.fz_is_directory(path)
(0)         def fz_is_document_reflowable(doc):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_document_reflowable()`.
(8)                 Is the document reflowable.
(8)                 Returns 1 to indicate reflowable documents, otherwise 0.
(4)             """
(4)             return _mupdf.fz_is_document_reflowable(doc)
(0)         def fz_is_empty_irect(r):
(4)             r""" Class-aware wrapper for `::fz_is_empty_irect()`."""
(4)             return _mupdf.fz_is_empty_irect(r)
(0)         def fz_is_empty_rect(r):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_empty_rect()`.
(8)                 Check if rectangle is empty.
(8)                 An empty rectangle is defined as one whose area is zero.
(8)                 All invalid rectangles are empty.
(4)             """
(4)             return _mupdf.fz_is_empty_rect(r)
(0)         def fz_is_eof(stm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_eof()`.
(8)                 Query if the stream has reached EOF (during normal bytewise
(8)                 reading).
(8)                 See fz_is_eof_bits for the equivalent function for bitwise
(8)                 reading.
(4)             """
(4)             return _mupdf.fz_is_eof(stm)
(0)         def fz_is_eof_bits(stm):
(4)             r"""
(4)             Class-aware wrapper for `::fz_is_eof_bits()`.
(8)                 Query if the stream has reached EOF (during bitwise
(8)                 reading).
```

```
32310: (8)                 See fz_is_eof for the equivalent function for bytewise
32311: (8)                 reading.
32312: (4)             """
32313: (4)             return _mupdf.fz_is_eof_bits(stm)
32314: (0)         def fz_is_external_link(uri):
32315: (4)             r"""
32316: (4)             Class-aware wrapper for `::fz_is_external_link()`.
32317: (8)                 Query whether a link is external to a document (determined by
32318: (8)                 uri containing a ':', intended to match with '://' which
32319: (8)                 separates the scheme from the scheme specific parts in URIs).
32320: (4)             """
32321: (4)             return _mupdf.fz_is_external_link(uri)
32322: (0)         def fz_is_identity(m):
32323: (4)             r""" Class-aware wrapper for `::fz_is_identity()`."""
32324: (4)             return _mupdf.fz_is_identity(m)
32325: (0)         def fz_is_infinite_irect(r):
32326: (4)             r"""
32327: (4)             Class-aware wrapper for `::fz_is_infinite_irect()`.
32328: (8)                 Check if an integer rectangle
32329: (8)                 is infinite.
32330: (4)             """
32331: (4)             return _mupdf.fz_is_infinite_irect(r)
32332: (0)         def fz_is_infinite_rect(r):
32333: (4)             r"""
32334: (4)             Class-aware wrapper for `::fz_is_infinite_rect()`.
32335: (8)                 Check if rectangle is infinite.
32336: (4)             """
32337: (4)             return _mupdf.fz_is_infinite_rect(r)
32338: (0)         def fz_is_libarchive_archive(file):
32339: (4)             r"""
32340: (4)             Class-aware wrapper for `::fz_is_libarchive_archive()`.
32341: (8)                 Detect if stream object is an archive supported by libarchive.
32342: (8)                 Assumes that the stream object is seekable.
32343: (4)             """
32344: (4)             return _mupdf.fz_is_libarchive_archive(file)
32345: (0)         def fz_is_page_range(s):
32346: (4)             r"""
32347: (5)              Class-aware wrapper for `::fz_is_page_range()`.
32348: (12)                    Check and parse string into page ranges:
32349: (4)             ,?(-?+|N)(-(-?+|N))?/
32350: (4)             """
32351: (4)             return _mupdf.fz_is_page_range(s)
32352: (0)         def fz_is_pixmap_monochrome(pixmap):
32353: (4)             r"""
32354: (4)             Class-aware wrapper for `::fz_is_pixmap_monochrome()`.
32355: (8)                 Check if the pixmap is a 1-channel image containing samples with
32356: (8)                 only values 0 and 255
32357: (4)             """
32358: (4)             return _mupdf.fz_is_pixmap_monochrome(pixmap)
32359: (0)         def fz_is_point_inside_irect(x, y, r):
32360: (4)             r"""
32361: (4)             Class-aware wrapper for `::fz_is_point_inside_irect()`.
32362: (8)                 Inclusion test for irects. (Rect is assumed to be open, i.e.
32363: (8)                 top right corner is not included).
32364: (4)             """
32365: (4)             return _mupdf.fz_is_point_inside_irect(x, y, r)
32366: (0)         def fz_is_point_inside_quad(p, q):
32367: (4)             r"""
32368: (4)             Class-aware wrapper for `::fz_is_point_inside_quad()`.
32369: (8)                 Inclusion test for quads.
32370: (4)             """
32371: (4)             return _mupdf.fz_is_point_inside_quad(p, q)
32372: (0)         def fz_is_point_inside_rect(p, r):
32373: (4)             r"""
32374: (4)             Class-aware wrapper for `::fz_is_point_inside_rect()`.
32375: (8)                 Inclusion test for rects. (Rect is assumed to be open, i.e.
32376: (8)                 top right corner is not included).
32377: (4)             """
32378: (4)             return _mupdf.fz_is_point_inside_rect(p, r)
```

```
32379: (0)          def fz_is_pow2(a):
32380: (4)              r""" Class-aware wrapper for `::fz_is_pow2()`."""
32381: (4)              return _mupdf.fz_is_pow2(a)
32382: (0)          def fz_is_quad_inside_quad(needle, haystack):
32383: (4)              r"""
32384: (4)              Class-aware wrapper for `::fz_is_quad_inside_quad()`.
32385: (8)                  Inclusion test for quad in quad.
32386: (8)                  This may break down if quads are not 'well formed'.
32387: (4)              """
32388: (4)              return _mupdf.fz_is_quad_inside_quad(needle, haystack)
32389: (0)          def fz_is_quad_intersecting_quad(a, b):
32390: (4)              r"""
32391: (4)              Class-aware wrapper for `::fz_is_quad_intersecting_quad()`.
32392: (8)                  Intersection test for quads.
32393: (8)                  This may break down if quads are not 'well formed'.
32394: (4)              """
32395: (4)              return _mupdf.fz_is_quad_intersecting_quad(a, b)
32396: (0)          def fz_is_rectilinear(m):
32397: (4)              r"""
32398: (4)              Class-aware wrapper for `::fz_is_rectilinear()`.
32399: (8)                  Check if a transformation is rectilinear.
32400: (8)                  Rectilinear means that no shearing is present and that any
32401: (8)                  rotations present are a multiple of 90 degrees. Usually this
32402: (8)                  is used to make sure that axis-aligned rectangles before the
32403: (8)                  transformation are still axis-aligned rectangles afterwards.
32404: (4)              """
32405: (4)              return _mupdf.fz_is_rectilinear(m)
32406: (0)          def fz_is_tar_archive(file):
32407: (4)              r"""
32408: (4)              Class-aware wrapper for `::fz_is_tar_archive()`.
32409: (8)                  Detect if stream object is a tar archive.
32410: (8)                  Assumes that the stream object is seekable.
32411: (4)              """
32412: (4)              return _mupdf.fz_is_tar_archive(file)
32413: (0)          def fz_is_valid_blend_colorspace(cs):
32414: (4)              r"""
32415: (4)              Class-aware wrapper for `::fz_is_valid_blend_colorspace()`.
32416: (8)                  Check to see that a colorspace is appropriate to be used as
32417: (8)                  a blending space (i.e. only grey, rgb or cmyk).
32418: (4)              """
32419: (4)              return _mupdf.fz_is_valid_blend_colorspace(cs)
32420: (0)          def fz_is_valid_irect(r):
32421: (4)              r"""
32422: (4)              Class-aware wrapper for `::fz_is_valid_irect()`.
32423: (8)                  Check if an integer rectangle is valid.
32424: (4)              """
32425: (4)              return _mupdf.fz_is_valid_irect(r)
32426: (0)          def fz_is_valid_rect(r):
32427: (4)              r"""
32428: (4)              Class-aware wrapper for `::fz_is_valid_rect()`.
32429: (8)                  Check if rectangle is valid.
32430: (4)              """
32431: (4)              return _mupdf.fz_is_valid_rect(r)
32432: (0)          def fz_is_zip_archive(file):
32433: (4)              r"""
32434: (4)              Class-aware wrapper for `::fz_is_zip_archive()`.
32435: (8)                  Detect if stream object is a zip archive.
32436: (8)                  Assumes that the stream object is seekable.
32437: (4)              """
32438: (4)              return _mupdf.fz_is_zip_archive(file)
32439: (0)          def fz_iso8859_1_from_unicode(u):
32440: (4)              r""" Class-aware wrapper for `::fz_iso8859_1_from_unicode()`."""
32441: (4)              return _mupdf.fz_iso8859_1_from_unicode(u)
32442: (0)          def fz_iso8859_7_from_unicode(u):
32443: (4)              r""" Class-aware wrapper for `::fz_iso8859_7_from_unicode()`."""
32444: (4)              return _mupdf.fz_iso8859_7_from_unicode(u)
32445: (0)          def fz_jbig2_globals_data(globals):
32446: (4)              r"""
32447: (4)              Class-aware wrapper for `::fz_jbig2_globals_data()`.
```

```
32448: (8)                 Return buffer containing jbig2 globals data stream.
32449: (4)             """
32450: (4)             return _mupdf.fz_jbig2_globals_data(globals)
32451: (0)         def fz_koi8u_from_unicode(u):
32452: (4)             r""" Class-aware wrapper for `::fz_koi8u_from_unicode()`."""
32453: (4)             return _mupdf.fz_koi8u_from_unicode(u)
32454: (0)         def fz_last_page(doc):
32455: (4)             r"""
32456: (4)             Class-aware wrapper for `::fz_last_page()`.
32457: (8)                 Function to get the location for the last page in the document.
32458: (8)                 Using this can be far more efficient in some cases than calling
32459: (8)                 fz_count_pages and using the page number.
32460: (4)             """
32461: (4)             return _mupdf.fz_last_page(doc)
32462: (0)         def fz_layout_document(doc, w, h, em):
32463: (4)             r"""
32464: (4)             Class-aware wrapper for `::fz_layout_document()`.
32465: (8)                 Layout reflowable document types.
32466: (8)                 w, h: Page size in points.
32467: (8)                 em: Default font size in points.
32468: (4)             """
32469: (4)             return _mupdf.fz_layout_document(doc, w, h, em)
32470: (0)         def fz_lineto(path, x, y):
32471: (4)             r"""
32472: (4)             Class-aware wrapper for `::fz_lineto()`.
32473: (8)                 Append a 'lineto' command to an open path.
32474: (8)                 path: The path to modify.
32475: (8)                 x, y: The coordinate to line to.
32476: (8)                 Throws exceptions on failure to allocate, or attempting to
32477: (8)                 modify a packed path.
32478: (4)             """
32479: (4)             return _mupdf.fz_lineto(path, x, y)
32480: (0)         def fz_list_archive_entry(arch, idx):
32481: (4)             r"""
32482: (4)             Class-aware wrapper for `::fz_list_archive_entry()`.
32483: (8)                 Get listed name of entry position idx.
32484: (8)                 idx: Must be a value >= 0 < return value from
32485: (8)                 fz_count_archive_entries. If not in range NULL will be
32486: (8)                 returned.
32487: (8)                 May throw an exception if this type of archive cannot list the
32488: (8)                 entries (such as a directory).
32489: (4)             """
32490: (4)             return _mupdf.fz_list_archive_entry(arch, idx)
32491: (0)         def fz_load_bmp_subimage(buf, len, subimage):
32492: (4)             r""" Class-aware wrapper for `::fz_load_bmp_subimage()`."""
32493: (4)             return _mupdf.fz_load_bmp_subimage(buf, len, subimage)
32494: (0)         def fz_load_bmp_subimage_count(buf, len):
32495: (4)             r""" Class-aware wrapper for `::fz_load_bmp_subimage_count()`."""
32496: (4)             return _mupdf.fz_load_bmp_subimage_count(buf, len)
32497: (0)         def fz_load_chapter_page(doc, chapter, page):
32498: (4)             r"""
32499: (4)             Class-aware wrapper for `::fz_load_chapter_page()`.
32500: (8)                 Load a page.
32501: (8)                 After fz_load_page is it possible to retrieve the size of the
32502: (8)                 page using fz_bound_page, or to render the page using
32503: (8)                 fz_run_page_*. Free the page by calling fz_drop_page.
32504: (8)                 chapter: chapter number, 0 is the first chapter of the document.
32505: (8)                 number: page number, 0 is the first page of the chapter.
32506: (4)             """
32507: (4)             return _mupdf.fz_load_chapter_page(doc, chapter, page)
32508: (0)         def fz_load_fallback_font(script, language, serif, bold, italic):
32509: (4)             r"""
32510: (4)             Class-aware wrapper for `::fz_load_fallback_font()`.
32511: (8)                 Try to load a fallback font for the
32512: (8)                 given combination of font attributes. Whether a font is
32513: (8)                 present or not will depend on the configuration in which
32514: (8)                 MuPDF is built.
32515: (8)                 script: The script desired (e.g. UCDN_SCRIPT_KATAKANA).
32516: (8)                 language: The language desired (e.g. FZ_LANG_ja).
```

```
32517: (8)                  serif: 1 if serif desired, 0 otherwise.
32518: (8)                  bold: 1 if bold desired, 0 otherwise.
32519: (8)                  italic: 1 if italic desired, 0 otherwise.
32520: (8)                  Returns a new font handle, or NULL if not available.
32521: (4)              """
32522: (4)              return _mupdf.fz_load_fallback_font(script, language, serif, bold, italic)
32523: (0)          def fz_load_jbig2_globals(buf):
32524: (4)              r"""
32525: (4)              Class-aware wrapper for `::fz_load_jbig2_globals()`.
32526: (8)                  Create a jbig2 globals record from a buffer.
32527: (8)                  Immutable once created.
32528: (4)              """
32529: (4)              return _mupdf.fz_load_jbig2_globals(buf)
32530: (0)          def fz_load_jbig2_subimage(buf, len, subimage):
32531: (4)              r""" Class-aware wrapper for `::fz_load_jbig2_subimage()`."""
32532: (4)              return _mupdf.fz_load_jbig2_subimage(buf, len, subimage)
32533: (0)          def fz_load_jbig2_subimage_count(buf, len):
32534: (4)              r""" Class-aware wrapper for `::fz_load_jbig2_subimage_count()`."""
32535: (4)              return _mupdf.fz_load_jbig2_subimage_count(buf, len)
32536: (0)          def fz_load_jpx(data, size, cs):
32537: (4)              r"""
32538: (4)              Class-aware wrapper for `::fz_load_jpx()`.
32539: (8)                  Exposed for PDF.
32540: (4)              """
32541: (4)              return _mupdf.fz_load_jpx(data, size, cs)
32542: (0)          def fz_load_links(page):
32543: (4)              r"""
32544: (4)              Class-aware wrapper for `::fz_load_links()`.
32545: (8)                  Load the list of links for a page.
32546: (8)                  Returns a linked list of all the links on the page, each with
32547: (8)                  its clickable region and link destination. Each link is
32548: (8)                  reference counted so drop and free the list of links by
32549: (8)                  calling fz_drop_link on the pointer return from fz_load_links.
32550: (8)                  page: Page obtained from fz_load_page.
32551: (4)              """
32552: (4)              return _mupdf.fz_load_links(page)
32553: (0)          def fz_load_outline(doc):
32554: (4)              r"""
32555: (4)              Class-aware wrapper for `::fz_load_outline()`.
32556: (8)                  Load the hierarchical document outline.
32557: (8)                  Should be freed by fz_drop_outline.
32558: (4)              """
32559: (4)              return _mupdf.fz_load_outline(doc)
32560: (0)          def fz_load_outline_from_iterator(iter):
32561: (4)              r"""
32562: (4)              Class-aware wrapper for `::fz_load_outline_from_iterator()`.
32563: (8)                  Routine to implement the old Structure based API from an iterator.
32564: (4)              """
32565: (4)              return _mupdf.fz_load_outline_from_iterator(iter)
32566: (0)          def fz_load_page(doc, number):
32567: (4)              r"""
32568: (4)              Class-aware wrapper for `::fz_load_page()`.
32569: (8)                  Load a given page number from a document. This may be much less
32570: (8)                  efficient than loading by location (chapter+page) for some
32571: (8)                  document types.
32572: (4)              """
32573: (4)              return _mupdf.fz_load_page(doc, number)
32574: (0)          def fz_load_pnm_subimage(buf, len, subimage):
32575: (4)              r""" Class-aware wrapper for `::fz_load_pnm_subimage()`."""
32576: (4)              return _mupdf.fz_load_pnm_subimage(buf, len, subimage)
32577: (0)          def fz_load_pnm_subimage_count(buf, len):
32578: (4)              r""" Class-aware wrapper for `::fz_load_pnm_subimage_count()`."""
32579: (4)              return _mupdf.fz_load_pnm_subimage_count(buf, len)
32580: (0)          def fz_load_system_cjk_font(name, ordering, serif):
32581: (4)              r"""
32582: (4)              Class-aware wrapper for `::fz_load_system_cjk_font()`.
32583: (8)                  Attempt to load a given font from
32584: (8)                  the system.
32585: (8)                  name: The name of the desired font.
```

```
32586: (8)                ordering: The ordering to load the font from (e.g. FZ_ADOBE_KOREA)
32587: (8)                serif: 1 if serif desired, 0 otherwise.
32588: (8)                Returns a new font handle, or NULL if no matching font was found
32589: (8)                (or on error).
32590: (4)            """
32591: (4)            return _mupdf.fz_load_system_cjk_font(name, ordering, serif)
32592: (0)        def fz_load_system_font(name, bold, italic, needs_exact_metrics):
32593: (4)            r"""
32594: (4)            Class-aware wrapper for `::fz_load_system_font()`.
32595: (8)                Attempt to load a given font from the system.
32596: (8)                name: The name of the desired font.
32597: (8)                bold: 1 if bold desired, 0 otherwise.
32598: (8)                italic: 1 if italic desired, 0 otherwise.
32599: (8)                needs_exact_metrics: 1 if an exact metrical match is required,
32600: (8)                0 otherwise.
32601: (8)                Returns a new font handle, or NULL if no matching font was found
32602: (8)                (or on error).
32603: (4)            """
32604: (4)            return _mupdf.fz_load_system_font(name, bold, italic, needs_exact_metrics)
32605: (0)        def fz_load_tiff_subimage(buf, len, subimage):
32606: (4)            r""" Class-aware wrapper for `::fz_load_tiff_subimage()`."""
32607: (4)            return _mupdf.fz_load_tiff_subimage(buf, len, subimage)
32608: (0)        def fz_load_tiff_subimage_count(buf, len):
32609: (4)            r"""
32610: (4)            Class-aware wrapper for `::fz_load_tiff_subimage_count()`.
32611: (8)                Exposed for CBZ.
32612: (4)            """
32613: (4)            return _mupdf.fz_load_tiff_subimage_count(buf, len)
32614: (0)        def fz_location_from_page_number(doc, number):
32615: (4)            r"""
32616: (4)            Class-aware wrapper for `::fz_location_from_page_number()`.
32617: (8)                Converts from page number to chapter+page. This may cause many
32618: (8)                chapters to be laid out in order to calculate the number of
32619: (8)                pages within those chapters.
32620: (4)            """
32621: (4)            return _mupdf.fz_location_from_page_number(doc, number)
32622: (0)        def fz_lock(lock):
32623: (4)            r"""
32624: (4)            Class-aware wrapper for `::fz_lock()`.
32625: (8)                Lock one of the user supplied mutexes.
32626: (4)            """
32627: (4)            return _mupdf.fz_lock(lock)
32628: (0)        def fz_log_error(str):
32629: (4)            r"""
32630: (4)            Class-aware wrapper for `::fz_log_error()`.
32631: (8)                Log a (preformatted) string to the registered
32632: (8)                error stream (stderr by default).
32633: (4)            """
32634: (4)            return _mupdf.fz_log_error(str)
32635: (0)        def fz_lookup_base14_font(name, len):
32636: (4)            r"""
32637: (4)            Class-aware wrapper for `::fz_lookup_base14_font()`.
32638: (4)            This function has out-params. Python/C# wrappers look like:
32639: (8)                `fz_lookup_base14_font(const char *name)` => `(const unsigned char *,
int len)`
32640: (8)                Search the builtin base14 fonts for a match.
32641: (8)                Whether a given font is present or not will depend on the
32642: (8)                configuration in which MuPDF is built.
32643: (8)                name: The name of the font desired.
32644: (8)                len: Pointer to a place to receive the length of the discovered
32645: (8)                font buffer.
32646: (8)                Returns a pointer to the font file data, or NULL if not present.
32647: (4)            """
32648: (4)            return _mupdf.fz_lookup_base14_font(name, len)
32649: (0)        def fz_lookup_blendmode(name):
32650: (4)            r"""
32651: (4)            Class-aware wrapper for `::fz_lookup_blendmode()`.
32652: (8)                Map from (case sensitive) blend mode string to enumeration.
32653: (4)            """
```

```
32654: (4)             return _mupdf.fz_lookup_blendmode(name)
32655: (0)         def fz_lookup_bookmark(doc, mark):
32656: (4)             r"""
32657: (4)             Class-aware wrapper for `::fz_lookup_bookmark()`.
32658: (8)                 Find a bookmark and return its page number.
32659: (4)             """
32660: (4)             return _mupdf.fz_lookup_bookmark(doc, mark)
32661: (0)         def fz_lookup_builtin_font(name, bold, italic, len):
32662: (4)             r"""
32663: (4)             Class-aware wrapper for `::fz_lookup_builtin_font()`.
32664: (4)             This function has out-params. Python/C# wrappers look like:
32665: (8)                 `fz_lookup_builtin_font(const char *name, int bold, int italic)` =>
`(const unsigned char *, int len)`
32666: (8)                 Search the builtin fonts for a match.
32667: (8)                 Whether a given font is present or not will depend on the
32668: (8)                 configuration in which MuPDF is built.
32669: (8)                 name: The name of the font desired.
32670: (8)                 bold: 1 if bold desired, 0 otherwise.
32671: (8)                 italic: 1 if italic desired, 0 otherwise.
32672: (8)                 len: Pointer to a place to receive the length of the discovered
32673: (8)                 font buffer.
32674: (8)                 Returns a pointer to the font file data, or NULL if not present.
32675: (4)             """
32676: (4)             return _mupdf.fz_lookup_builtin_font(name, bold, italic, len)
32677: (0)         def fz_lookup_cjk_font(ordering, len, index):
32678: (4)             r"""
32679: (4)             Class-aware wrapper for `::fz_lookup_cjk_font()`.
32680: (4)             This function has out-params. Python/C# wrappers look like:
32681: (8)                 `fz_lookup_cjk_font(int ordering)` => `(const unsigned char *, int
len, int index)`
32682: (8)                 Search the builtin cjk fonts for a match.
32683: (8)                 Whether a font is present or not will depend on the
32684: (8)                 configuration in which MuPDF is built.
32685: (8)                 ordering: The desired ordering of the font (e.g. FZ_ADOBE_KOREA).
32686: (8)                 len: Pointer to a place to receive the length of the discovered
32687: (8)                 font buffer.
32688: (8)                 Returns a pointer to the font file data, or NULL if not present.
32689: (4)             """
32690: (4)             return _mupdf.fz_lookup_cjk_font(ordering, len, index)
32691: (0)         def fz_lookup_cjk_font_by_language(lang, len, subfont):
32692: (4)             r"""
32693: (4)             Class-aware wrapper for `::fz_lookup_cjk_font_by_language()`.
32694: (4)             This function has out-params. Python/C# wrappers look like:
32695: (8)                 `fz_lookup_cjk_font_by_language(const char *lang)` => `(const unsigned
char *, int len, int subfont)`
32696: (8)                 Search the builtin cjk fonts for a match for a given language.
32697: (8)                 Whether a font is present or not will depend on the
32698: (8)                 configuration in which MuPDF is built.
32699: (8)                 lang: Pointer to a (case sensitive) language string (e.g.
32700: (8)                 "ja", "ko", "zh-Hant" etc).
32701: (8)                 len: Pointer to a place to receive the length of the discovered
32702: (8)                 font buffer.
32703: (8)                 subfont: Pointer to a place to store the subfont index of the
32704: (8)                 discovered font.
32705: (8)                 Returns a pointer to the font file data, or NULL if not present.
32706: (4)             """
32707: (4)             return _mupdf.fz_lookup_cjk_font_by_language(lang, len, subfont)
32708: (0)         def fz_lookup_cjk_ordering_by_language(name):
32709: (4)             r"""
32710: (4)             Class-aware wrapper for `::fz_lookup_cjk_ordering_by_language()`.
32711: (8)                 Return the matching FZ_ADOBE_* ordering
32712: (8)                 for the given language tag, such as "zh-Hant", "zh-Hans", "ja", or
"ko".
32713: (4)             """
32714: (4)             return _mupdf.fz_lookup_cjk_ordering_by_language(name)
32715: (0)         def fz_lookup_image_type(type):
32716: (4)             r"""
32717: (4)             Class-aware wrapper for `::fz_lookup_image_type()`.
32718: (8)                 Map from (case sensitive) image type string to FZ_IMAGE_*
```

```
32719: (8)                  type value.
32720: (4)              """
32721: (4)              return _mupdf.fz_lookup_image_type(type)
32722: (0)          def fz_lookup_metadata(doc, key, buf, size):
32723: (4)              r"""
32724: (4)              Class-aware wrapper for `::fz_lookup_metadata()`.
32725: (8)                  Retrieve document meta data strings.
32726: (8)                  doc: The document to query.
32727: (8)                  key: Which meta data key to retrieve...
32728: (8)                  Basic information:
32729: (12)                     'format'    -- Document format and version.
32730: (12)                     'encryption'    -- Description of the encryption used.
32731: (8)                  From the document information dictionary:
32732: (12)                     'info:Title'
32733: (12)                     'info:Author'
32734: (12)                     'info:Subject'
32735: (12)                     'info:Keywords'
32736: (12)                     'info:Creator'
32737: (12)                     'info:Producer'
32738: (12)                     'info:CreationDate'
32739: (12)                     'info:ModDate'
32740: (8)                  buf: The buffer to hold the results (a nul-terminated UTF-8
32741: (8)                  string).
32742: (8)                  size: Size of 'buf'.
32743: (8)                  Returns the number of bytes need to store the string plus terminator
32744: (8)                  (will be larger than 'size' if the output was truncated), or -1 if the
32745: (8)                  key is not recognized or found.
32746: (4)              """
32747: (4)              return _mupdf.fz_lookup_metadata(doc, key, buf, size)
32748: (0)          def fz_lookup_metadata2(doc, key):
32749: (4)              r"""
32750: (4)              Class-aware wrapper for `::fz_lookup_metadata2()`.
32751: (4)              C++ alternative to `fz_lookup_metadata()` that returns a `std::string`
32752: (4)              or calls `fz_throw()` if not found.
32753: (4)              """
32754: (4)              return _mupdf.fz_lookup_metadata2(doc, key)
32755: (0)          def fz_lookup_noto_boxes_font(len):
32756: (4)              r"""
32757: (4)              Class-aware wrapper for `::fz_lookup_noto_boxes_font()`.
32758: (4)              This function has out-params. Python/C# wrappers look like:
32759: (8)                  `fz_lookup_noto_boxes_font()` => `(const unsigned char *, int len)`
32760: (4)              """
32761: (4)              return _mupdf.fz_lookup_noto_boxes_font(len)
32762: (0)          def fz_lookup_noto_emoji_font(len):
32763: (4)              r"""
32764: (4)              Class-aware wrapper for `::fz_lookup_noto_emoji_font()`.
32765: (4)              This function has out-params. Python/C# wrappers look like:
32766: (8)                  `fz_lookup_noto_emoji_font()` => `(const unsigned char *, int len)`
32767: (4)              """
32768: (4)              return _mupdf.fz_lookup_noto_emoji_font(len)
32769: (0)          def fz_lookup_noto_font(script, lang, len, subfont):
32770: (4)              r"""
32771: (4)              Class-aware wrapper for `::fz_lookup_noto_font()`.
32772: (4)              This function has out-params. Python/C# wrappers look like:
32773: (8)                  `fz_lookup_noto_font(int script, int lang)` => `(const unsigned char
*, int len, int subfont)`
32774: (8)                  Search the builtin noto fonts for a match.
32775: (8)                  Whether a font is present or not will depend on the
32776: (8)                  configuration in which MuPDF is built.
32777: (8)                  script: The script desired (e.g. UCDN_SCRIPT_KATAKANA).
32778: (8)                  lang: The language desired (e.g. FZ_LANG_ja).
32779: (8)                  len: Pointer to a place to receive the length of the discovered
32780: (8)                  font buffer.
32781: (8)                  Returns a pointer to the font file data, or NULL if not present.
32782: (4)              """
32783: (4)              return _mupdf.fz_lookup_noto_font(script, lang, len, subfont)
32784: (0)          def fz_lookup_noto_math_font(len):
32785: (4)              r"""
32786: (4)              Class-aware wrapper for `::fz_lookup_noto_math_font()`.
```

```
32787: (4)             This function has out-params. Python/C# wrappers look like:
32788: (8)                 `fz_lookup_noto_math_font()` => `(const unsigned char *, int len)`
32789: (8)                 Search the builtin noto fonts specific symbol fonts.
32790: (8)                 Whether a font is present or not will depend on the
32791: (8)                 configuration in which MuPDF is built.
32792: (4)             """
32793: (4)             return _mupdf.fz_lookup_noto_math_font(len)
32794: (0)         def fz_lookup_noto_music_font(len):
32795: (4)             r"""
32796: (4)             Class-aware wrapper for `::fz_lookup_noto_music_font()`.
32797: (4)             This function has out-params. Python/C# wrappers look like:
32798: (8)                 `fz_lookup_noto_music_font()` => `(const unsigned char *, int len)`
32799: (4)             """
32800: (4)             return _mupdf.fz_lookup_noto_music_font(len)
32801: (0)         def fz_lookup_noto_symbol1_font(len):
32802: (4)             r"""
32803: (4)             Class-aware wrapper for `::fz_lookup_noto_symbol1_font()`.
32804: (4)             This function has out-params. Python/C# wrappers look like:
32805: (8)                 `fz_lookup_noto_symbol1_font()` => `(const unsigned char *, int len)`
32806: (4)             """
32807: (4)             return _mupdf.fz_lookup_noto_symbol1_font(len)
32808: (0)         def fz_lookup_noto_symbol2_font(len):
32809: (4)             r"""
32810: (4)             Class-aware wrapper for `::fz_lookup_noto_symbol2_font()`.
32811: (4)             This function has out-params. Python/C# wrappers look like:
32812: (8)                 `fz_lookup_noto_symbol2_font()` => `(const unsigned char *, int len)`
32813: (4)             """
32814: (4)             return _mupdf.fz_lookup_noto_symbol2_font(len)
32815: (0)         def fz_lookup_rendering_intent(name):
32816: (4)             r"""
32817: (4)             Class-aware wrapper for `::fz_lookup_rendering_intent()`.
32818: (8)                 Map from (case sensitive) rendering intent string to enumeration
32819: (8)                 value.
32820: (4)             """
32821: (4)             return _mupdf.fz_lookup_rendering_intent(name)
32822: (0)         def fz_make_bookmark(doc, loc):
32823: (4)             r"""
32824: (4)             Class-aware wrapper for `::fz_make_bookmark()`.
32825: (8)                 Create a bookmark for the given page, which can be used to find
32826: (8)                 the same location after the document has been laid out with
32827: (8)                 different parameters.
32828: (4)             """
32829: (4)             return _mupdf.fz_make_bookmark(doc, loc)
32830: (0)         def fz_make_irect(x0, y0, x1, y1):
32831: (4)             r""" Class-aware wrapper for `::fz_make_irect()`."""
32832: (4)             return _mupdf.fz_make_irect(x0, y0, x1, y1)
32833: (0)         def fz_make_location(chapter, page):
32834: (4)             r"""
32835: (4)             Class-aware wrapper for `::fz_make_location()`.
32836: (8)                 Simple constructor for fz_locations.
32837: (4)             """
32838: (4)             return _mupdf.fz_make_location(chapter, page)
32839: (0)         def fz_make_matrix(a, b, c, d, e, f):
32840: (4)             r""" Class-aware wrapper for `::fz_make_matrix()`."""
32841: (4)             return _mupdf.fz_make_matrix(a, b, c, d, e, f)
32842: (0)         def fz_make_point(x, y):
32843: (4)             r""" Class-aware wrapper for `::fz_make_point()`."""
32844: (4)             return _mupdf.fz_make_point(x, y)
32845: (0)         def fz_make_quad(ul_x, ul_y, ur_x, ur_y, ll_x, ll_y, lr_x, lr_y):
32846: (4)             r"""
32847: (4)             Class-aware wrapper for `::fz_make_quad()`.
32848: (8)                 Inline convenience construction function.
32849: (4)             """
32850: (4)             return _mupdf.fz_make_quad(ul_x, ul_y, ur_x, ur_y, ll_x, ll_y, lr_x, lr_y)
32851: (0)         def fz_make_rect(x0, y0, x1, y1):
32852: (4)             r""" Class-aware wrapper for `::fz_make_rect()`."""
32853: (4)             return _mupdf.fz_make_rect(x0, y0, x1, y1)
32854: (0)         def fz_malloc(size):
32855: (4)             r"""
```

```
(4)             Class-aware wrapper for `::fz_malloc()`.
(8)                 Allocate uninitialized memory of a given size.
(8)                 Does NOT clear the memory!
(8)                 May return NULL for size = 0.
(8)                 Throws exception in the event of failure to allocate.
(4)             """
(4)             return _mupdf.fz_malloc(size)
(0)         def fz_malloc_no_throw(size):
(4)             r"""
(4)             Class-aware wrapper for `::fz_malloc_no_throw()`.
(8)                 fz_malloc equivalent that returns NULL rather than throwing
(8)                 exceptions.
(4)             """
(4)             return _mupdf.fz_malloc_no_throw(size)
(0)         def fz_matrix_expansion(m):
(4)             r"""
(4)             Class-aware wrapper for `::fz_matrix_expansion()`.
(8)                 Calculate average scaling factor of matrix.
(4)             """
(4)             return _mupdf.fz_matrix_expansion(m)
(0)         def fz_matrix_max_expansion(m):
(4)             r"""
(4)             Class-aware wrapper for `::fz_matrix_max_expansion()`.
(8)                 Find the largest expansion performed by this matrix.
(8)                 (i.e. max(abs(m.a),abs(m.b),abs(m.c),abs(m.d))
(4)             """
(4)             return _mupdf.fz_matrix_max_expansion(m)
(0)         def fz_max(a, b):
(4)             r""" Class-aware wrapper for `::fz_max()`."""
(4)             return _mupdf.fz_max(a, b)
(0)         def fz_maxi(a, b):
(4)             r""" Class-aware wrapper for `::fz_maxi()`."""
(4)             return _mupdf.fz_maxi(a, b)
(0)         def fz_maxi64(a, b):
(4)             r""" Class-aware wrapper for `::fz_maxi64()`."""
(4)             return _mupdf.fz_maxi64(a, b)
(0)         def fz_maxz(a, b):
(4)             r""" Class-aware wrapper for `::fz_maxz()`."""
(4)             return _mupdf.fz_maxz(a, b)
(0)         def fz_md5_buffer(buffer, digest):
(4)             r"""
(4)             Class-aware wrapper for `::fz_md5_buffer()`.
(8)                 Create an MD5 digest from buffer contents.
(8)                 Never throws exceptions.
(4)             """
(4)             return _mupdf.fz_md5_buffer(buffer, digest)
(0)         def fz_md5_final(state, digest):
(4)             r"""
(4)             Class-aware wrapper for `::fz_md5_final()`.
(8)                 MD5 finalization. Ends an MD5 message-digest operation, writing
(8)                 the message digest and zeroizing the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_md5_final(state, digest)
(0)         def fz_md5_final2(md5):
(4)             r"""
(4)             Class-aware wrapper for `::fz_md5_final2()`.
(4)             C++ alternative to fz_md5_final() that returns the digest by value.
(4)             """
(4)             return _mupdf.fz_md5_final2(md5)
(0)         def fz_md5_init(state):
(4)             r"""
(4)             Class-aware wrapper for `::fz_md5_init()`.
(8)                 MD5 initialization. Begins an MD5 operation, writing a new
(8)                 context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_md5_init(state)
(0)         def fz_md5_pixmap(pixmap, digest):
```

```
32925: (4)             r""" Class-aware wrapper for `::fz_md5_pixmap()`."""
32926: (4)             return _mupdf.fz_md5_pixmap(pixmap, digest)
32927: (0)         def fz_md5_pixmap2(pixmap):
32928: (4)             r"""
32929: (4)             Class-aware wrapper for `::fz_md5_pixmap2()`.
32930: (4)             C++ alternative to `fz_md5_pixmap()` that returns the digest by value.
32931: (4)             """
32932: (4)             return _mupdf.fz_md5_pixmap2(pixmap)
32933: (0)         def fz_md5_update(state, input, inlen):
32934: (4)             r"""
32935: (4)             Class-aware wrapper for `::fz_md5_update()`.
32936: (8)                 MD5 block update operation. Continues an MD5 message-digest
32937: (8)                 operation, processing another message block, and updating the
32938: (8)                 context.
32939: (8)                 Never throws an exception.
32940: (4)             """
32941: (4)             return _mupdf.fz_md5_update(state, input, inlen)
32942: (0)         def fz_md5_update_int64(state, i):
32943: (4)             r"""
32944: (4)             Class-aware wrapper for `::fz_md5_update_int64()`.
32945: (8)                 MD5 block update operation. Continues an MD5 message-digest
32946: (8)                 operation, processing an int64, and updating the context.
32947: (8)                 Never throws an exception.
32948: (4)             """
32949: (4)             return _mupdf.fz_md5_update_int64(state, i)
32950: (0)         def fz_measure_string(user_font, trm, s, wmode, bidi_level, markup_dir,
language):
32951: (4)             r"""
32952: (4)             Class-aware wrapper for `::fz_measure_string()`.
32953: (8)                 Measure the advance width of a UTF8 string should it be added to a
text object.
32954: (8)                 This uses the same layout algorithms as fz_show_string, and can be
used
32955: (8)                 to calculate text alignment adjustments.
32956: (4)             """
32957: (4)             return _mupdf.fz_measure_string(user_font, trm, s, wmode, bidi_level,
markup_dir, language)
32958: (0)         def fz_memmem(haystack, haystacklen, needle, needlelen):
32959: (4)             r"""
32960: (4)             Class-aware wrapper for `::fz_memmem()`.
32961: (8)                 Find the start of the first occurrence of the substring needle in
haystack.
32962: (4)             """
32963: (4)             return _mupdf.fz_memmem(haystack, haystacklen, needle, needlelen)
32964: (0)         def fz_memrnd(block, len):
32965: (4)             r"""
32966: (4)             Class-aware wrapper for `::fz_memrnd()`.
32967: (8)                 Fill block with len bytes of pseudo-randomness.
32968: (4)             """
32969: (4)             return _mupdf.fz_memrnd(block, len)
32970: (0)         def fz_min(a, b):
32971: (4)             r""" Class-aware wrapper for `::fz_min()`."""
32972: (4)             return _mupdf.fz_min(a, b)
32973: (0)         def fz_mini(a, b):
32974: (4)             r""" Class-aware wrapper for `::fz_mini()`."""
32975: (4)             return _mupdf.fz_mini(a, b)
32976: (0)         def fz_mini64(a, b):
32977: (4)             r""" Class-aware wrapper for `::fz_mini64()`."""
32978: (4)             return _mupdf.fz_mini64(a, b)
32979: (0)         def fz_minz(a, b):
32980: (4)             r""" Class-aware wrapper for `::fz_minz()`."""
32981: (4)             return _mupdf.fz_minz(a, b)
32982: (0)         def fz_mkdir(path):
32983: (4)             r""" Class-aware wrapper for `::fz_mkdir()`."""
32984: (4)             return _mupdf.fz_mkdir(path)
32985: (0)         def fz_morph_error(fromcode, tocode):
32986: (4)             r"""
32987: (4)             Class-aware wrapper for `::fz_morph_error()`.
32988: (8)                 Called within a catch block this modifies the current
```

```
32989: (8)                 exception's code. If it's of type 'fromcode' it is
32990: (8)                 modified to 'tocode'. Typically used for 'downgrading'
32991: (8)                 exception severity.
32992: (4)             """
32993: (4)             return _mupdf.fz_morph_error(fromcode, tocode)
32994: (0)         def fz_mount_multi_archive(arch_, sub, path):
32995: (4)             r"""
32996: (4)             Class-aware wrapper for `::fz_mount_multi_archive()`.
32997: (8)                 Add an archive to the set of archives handled by a multi
32998: (8)                 archive.
32999: (8)                 If path is NULL, then the archive contents will appear at the
33000: (8)                 top level, otherwise, the archives contents will appear prefixed
33001: (8)                 by path.
33002: (4)             """
33003: (4)             return _mupdf.fz_mount_multi_archive(arch_, sub, path)
33004: (0)         def fz_moveto(path, x, y):
33005: (4)             r"""
33006: (4)             Class-aware wrapper for `::fz_moveto()`.
33007: (8)                 Append a 'moveto' command to a path.
33008: (8)                 This 'opens' a path.
33009: (8)                 path: The path to modify.
33010: (8)                 x, y: The coordinate to move to.
33011: (8)                 Throws exceptions on failure to allocate, or attempting to
33012: (8)                 modify a packed path.
33013: (4)             """
33014: (4)             return _mupdf.fz_moveto(path, x, y)
33015: (0)         def fz_mul255(a, b):
33016: (4)             r"""
33017: (4)             Class-aware wrapper for `::fz_mul255()`.
33018: (8)                 Multiply scaled two integers in the 0..255 range
33019: (4)             """
33020: (4)             return _mupdf.fz_mul255(a, b)
33021: (0)         def fz_needs_password(doc):
33022: (4)             r"""
33023: (4)             Class-aware wrapper for `::fz_needs_password()`.
33024: (8)                 Check if a document is encrypted with a
33025: (8)                 non-blank password.
33026: (4)             """
33027: (4)             return _mupdf.fz_needs_password(doc)
33028: (0)         def fz_new_archive_of_size(file, size):
33029: (4)             r""" Class-aware wrapper for `::fz_new_archive_of_size()`."""
33030: (4)             return _mupdf.fz_new_archive_of_size(file, size)
33031: (0)         def fz_new_base14_font(name):
33032: (4)             r"""
33033: (4)             Class-aware wrapper for `::fz_new_base14_font()`.
33034: (8)                 Create a new font from one of the built-in fonts.
33035: (4)             """
33036: (4)             return _mupdf.fz_new_base14_font(name)
33037: (0)         def fz_new_bbox_device(rectp):
33038: (4)             r"""
33039: (4)             Class-aware wrapper for `::fz_new_bbox_device()`.
33040: (8)                 Create a device to compute the bounding
33041: (8)                 box of all marks on a page.
33042: (8)                 The returned bounding box will be the union of all bounding
33043: (8)                 boxes of all objects on a page.
33044: (4)             """
33045: (4)             return _mupdf.fz_new_bbox_device(rectp)
33046: (0)         def fz_new_bitmap(w, h, n, xres, yres):
33047: (4)             r"""
33048: (4)             Class-aware wrapper for `::fz_new_bitmap()`.
33049: (8)                 Create a new bitmap.
33050: (8)                 w, h: Width and Height for the bitmap
33051: (8)                 n: Number of color components (assumed to be a divisor of 8)
33052: (8)                 xres, yres: X and Y resolutions (in pixels per inch).
33053: (8)                 Returns pointer to created bitmap structure. The bitmap
33054: (8)                 data is uninitialised.
33055: (4)             """
33056: (4)             return _mupdf.fz_new_bitmap(w, h, n, xres, yres)
33057: (0)         def fz_new_bitmap_from_pixmap(pix, ht):
```

```
33058: (4)              r"""
33059: (4)              Class-aware wrapper for `::fz_new_bitmap_from_pixmap()`.
33060: (8)                  Make a bitmap from a pixmap and a halftone.
33061: (8)                  pix: The pixmap to generate from. Currently must be a single
33062: (8)                  color component with no alpha.
33063: (8)                  ht: The halftone to use. NULL implies the default halftone.
33064: (8)                  Returns the resultant bitmap. Throws exceptions in the case of
33065: (8)                  failure to allocate.
33066: (4)              """
33067: (4)              return _mupdf.fz_new_bitmap_from_pixmap(pix, ht)
33068: (0)          def fz_new_bitmap_from_pixmap_band(pix, ht, band_start):
33069: (4)              r"""
33070: (4)              Class-aware wrapper for `::fz_new_bitmap_from_pixmap_band()`.
33071: (8)                  Make a bitmap from a pixmap and a
33072: (8)                  halftone, allowing for the position of the pixmap within an
33073: (8)                  overall banded rendering.
33074: (8)                  pix: The pixmap to generate from. Currently must be a single
33075: (8)                  color component with no alpha.
33076: (8)                  ht: The halftone to use. NULL implies the default halftone.
33077: (8)                  band_start: Vertical offset within the overall banded rendering
33078: (8)                  (in pixels)
33079: (8)                  Returns the resultant bitmap. Throws exceptions in the case of
33080: (8)                  failure to allocate.
33081: (4)              """
33082: (4)              return _mupdf.fz_new_bitmap_from_pixmap_band(pix, ht, band_start)
33083: (0)          def fz_new_buffer(capacity):
33084: (4)              r""" Class-aware wrapper for `::fz_new_buffer()`."""
33085: (4)              return _mupdf.fz_new_buffer(capacity)
33086: (0)          def fz_new_buffer_from_base64(data, size):
33087: (4)              r"""
33088: (4)              Class-aware wrapper for `::fz_new_buffer_from_base64()`.
33089: (8)                  Create a new buffer with data decoded from a base64 input string.
33090: (4)              """
33091: (4)              return _mupdf.fz_new_buffer_from_base64(data, size)
33092: (0)          def fz_new_buffer_from_copied_data(data, size):
33093: (4)              r"""
33094: (4)              Class-aware wrapper for `::fz_new_buffer_from_copied_data()`.
33095: (8)                  Create a new buffer containing a copy of the passed data.
33096: (4)              """
33097: (4)              return _mupdf.fz_new_buffer_from_copied_data(data, size)
33098: (0)          def fz_new_buffer_from_data(data, size):
33099: (4)              r"""
33100: (4)              Class-aware wrapper for `::fz_new_buffer_from_data()`.
33101: (8)                  Create a new buffer with existing data.
33102: (8)                  data: Pointer to existing data.
33103: (8)                  size: Size of existing data.
33104: (8)                  Takes ownership of data. Does not make a copy. Calls fz_free on
33105: (8)                  the data when the buffer is deallocated. Do not use 'data' after
33106: (8)                  passing to this function.
33107: (8)                  Returns pointer to new buffer. Throws exception on allocation
33108: (8)                  failure.
33109: (4)              """
33110: (4)              return _mupdf.fz_new_buffer_from_data(data, size)
33111: (0)          def fz_new_buffer_from_display_list(list, options):
33112: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_display_list()`."""
33113: (4)              return _mupdf.fz_new_buffer_from_display_list(list, options)
33114: (0)          def fz_new_buffer_from_image_as_jpeg(image, color_params, quality,
invert_cmyk):
33115: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_jpeg()`."""
33116: (4)              return _mupdf.fz_new_buffer_from_image_as_jpeg(image, color_params,
quality, invert_cmyk)
33117: (0)          def fz_new_buffer_from_image_as_jpx(image, color_params, quality):
33118: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_jpx()`."""
33119: (4)              return _mupdf.fz_new_buffer_from_image_as_jpx(image, color_params,
quality)
33120: (0)          def fz_new_buffer_from_image_as_pam(image, color_params):
33121: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pam()`."""
33122: (4)              return _mupdf.fz_new_buffer_from_image_as_pam(image, color_params)
33123: (0)          def fz_new_buffer_from_image_as_png(image, color_params):
```

```
33124: (4)              r"""
33125: (4)              Class-aware wrapper for `::fz_new_buffer_from_image_as_png()`.
33126: (8)                  Reencode a given image as a PNG into a buffer.
33127: (8)                  Ownership of the buffer is returned.
33128: (4)              """
33129: (4)              return _mupdf.fz_new_buffer_from_image_as_png(image, color_params)
33130: (0)          def fz_new_buffer_from_image_as_pnm(image, color_params):
33131: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_pnm()`."""
33132: (4)              return _mupdf.fz_new_buffer_from_image_as_pnm(image, color_params)
33133: (0)          def fz_new_buffer_from_image_as_psd(image, color_params):
33134: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_image_as_psd()`."""
33135: (4)              return _mupdf.fz_new_buffer_from_image_as_psd(image, color_params)
33136: (0)          def fz_new_buffer_from_page(page, options):
33137: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_page()`."""
33138: (4)              return _mupdf.fz_new_buffer_from_page(page, options)
33139: (0)          def fz_new_buffer_from_page_number(doc, number, options):
33140: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_page_number()`."""
33141: (4)              return _mupdf.fz_new_buffer_from_page_number(doc, number, options)
33142: (0)          def fz_new_buffer_from_page_with_format(page, format, options, transform,
cookie):
33143: (4)              r"""
33144: (4)              Class-aware wrapper for `::fz_new_buffer_from_page_with_format()`.
33145: (8)                  Returns an fz_buffer containing a page after conversion to specified
format.
33146: (8)                  page: The page to convert.
33147: (8)                  format, options: Passed to fz_new_document_writer_with_output()
internally.
33148: (8)                  transform, cookie: Passed to fz_run_page() internally.
33149: (4)              """
33150: (4)              return _mupdf.fz_new_buffer_from_page_with_format(page, format, options,
transform, cookie)
33151: (0)          def fz_new_buffer_from_pixmap_as_jpeg(pixmap, color_params, quality,
invert_cmyk):
33152: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_jpeg()`."""
33153: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_jpeg(pixmap, color_params,
quality, invert_cmyk)
33154: (0)          def fz_new_buffer_from_pixmap_as_jpx(pix, color_params, quality):
33155: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_jpx()`."""
33156: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_jpx(pix, color_params, quality)
33157: (0)          def fz_new_buffer_from_pixmap_as_pam(pixmap, color_params):
33158: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_pam()`."""
33159: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_pam(pixmap, color_params)
33160: (0)          def fz_new_buffer_from_pixmap_as_png(pixmap, color_params):
33161: (4)              r"""
33162: (4)              Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_png()`.
33163: (8)                  Reencode a given pixmap as a PNG into a buffer.
33164: (8)                  Ownership of the buffer is returned.
33165: (4)              """
33166: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_png(pixmap, color_params)
33167: (0)          def fz_new_buffer_from_pixmap_as_pnm(pixmap, color_params):
33168: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_pnm()`."""
33169: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_pnm(pixmap, color_params)
33170: (0)          def fz_new_buffer_from_pixmap_as_psd(pix, color_params):
33171: (4)              r""" Class-aware wrapper for `::fz_new_buffer_from_pixmap_as_psd()`."""
33172: (4)              return _mupdf.fz_new_buffer_from_pixmap_as_psd(pix, color_params)
33173: (0)          def fz_new_buffer_from_shared_data(data, size):
33174: (4)              r"""
33175: (4)              Class-aware wrapper for `::fz_new_buffer_from_shared_data()`.
33176: (8)                  Like fz_new_buffer, but does not take ownership.
33177: (4)              """
33178: (4)              return _mupdf.fz_new_buffer_from_shared_data(data, size)
33179: (0)          def fz_new_buffer_from_stext_page(text):
33180: (4)              r"""
33181: (4)              Class-aware wrapper for `::fz_new_buffer_from_stext_page()`.
33182: (8)                  Convert structured text into plain text.
33183: (4)              """
33184: (4)              return _mupdf.fz_new_buffer_from_stext_page(text)
33185: (0)          def fz_new_builtin_font(name, is_bold, is_italic):
33186: (4)              r""" Class-aware wrapper for `::fz_new_builtin_font()`."""
```

```
33187: (4)              return _mupdf.fz_new_builtin_font(name, is_bold, is_italic)
33188: (0)          def fz_new_cal_gray_colorspace(wp, bp, gamma):
33189: (4)              r"""
33190: (4)              Class-aware wrapper for `::fz_new_cal_gray_colorspace()`.
33191: (8)                  Create a calibrated gray colorspace.
33192: (8)                  The returned reference should be dropped when it is finished
33193: (8)                  with.
33194: (8)                  Colorspaces are immutable once created.
33195: (4)              """
33196: (4)              return _mupdf.fz_new_cal_gray_colorspace(wp, bp, gamma)
33197: (0)          def fz_new_cal_rgb_colorspace(wp, bp, gamma, matrix):
33198: (4)              r"""
33199: (4)              Class-aware wrapper for `::fz_new_cal_rgb_colorspace()`.
33200: (8)                  Create a calibrated rgb colorspace.
33201: (8)                  The returned reference should be dropped when it is finished
33202: (8)                  with.
33203: (8)                  Colorspaces are immutable once created.
33204: (4)              """
33205: (4)              return _mupdf.fz_new_cal_rgb_colorspace(wp, bp, gamma, matrix)
33206: (0)          def fz_new_cjk_font(ordering):
33207: (4)              r""" Class-aware wrapper for `::fz_new_cjk_font()`."""
33208: (4)              return _mupdf.fz_new_cjk_font(ordering)
33209: (0)          def fz_new_colorspace(type, flags, n, name):
33210: (4)              r"""
33211: (4)              Class-aware wrapper for `::fz_new_colorspace()`.
33212: (8)                  Creates a new colorspace instance and returns a reference.
33213: (8)                  No internal checking is done that the colorspace type (e.g.
33214: (8)                  CMYK) matches with the flags (e.g. FZ_COLORSPACE_HAS_CMYK) or
33215: (8)                  colorant count (n) or name.
33216: (8)                  The reference should be dropped when it is finished with.
33217: (8)                  Colorspaces are immutable once created (with the exception of
33218: (8)                  setting up colorant names for separation spaces).
33219: (4)              """
33220: (4)              return _mupdf.fz_new_colorspace(type, flags, n, name)
33221: (0)          def fz_new_compressed_buffer():
33222: (4)              r"""
33223: (4)              Class-aware wrapper for `::fz_new_compressed_buffer()`.
33224: (8)                  Create a new, UNKNOWN format, compressed_buffer.
33225: (4)              """
33226: (4)              return _mupdf.fz_new_compressed_buffer()
33227: (0)          def fz_new_default_colorspaces():
33228: (4)              r"""
33229: (4)              Class-aware wrapper for `::fz_new_default_colorspaces()`.
33230: (8)                  Create a new default colorspace structure with values inherited
33231: (8)                  from the context, and return a reference to it.
33232: (8)                  These can be overridden using fz_set_default_xxxx.
33233: (8)                  These should not be overridden while more than one caller has
33234: (8)                  the reference for fear of race conditions.
33235: (8)                  The caller should drop this reference once finished with it.
33236: (4)              """
33237: (4)              return _mupdf.fz_new_default_colorspaces()
33238: (0)          def fz_new_deflated_data(compressed_length, source, source_length, level):
33239: (4)              r"""
33240: (5)               Class-aware wrapper for `::fz_new_deflated_data()`.
33241: (8)                  This function has out-params. Python/C# wrappers look like:
33242: (12)                     `fz_new_deflated_data(const unsigned char *source, size_t
source_length, ::fz_deflate_level level)` => `(unsigned char *, size_t compressed_length)`
33243: (12)                     Compress source_length bytes of data starting
33244: (12)                     at source, into a new memory block malloced for that purpose.
33245: (4)              compressed_length is updated on exit to contain the size used.
33246: (12)                     Ownership of the block is returned from this function, and the
33247: (12)                     caller is therefore responsible for freeing it. The block may be
33248: (12)                     considerably larger than is actually required. The caller is
33249: (12)                     free to fz_realloc it down if it wants to.
33250: (4)              """
33251: (4)              return _mupdf.fz_new_deflated_data(compressed_length, source,
source_length, level)
33252: (0)          def fz_new_deflated_data_from_buffer(compressed_length, buffer, level):
33253: (4)              r"""
```

```
33254: (4)              Class-aware wrapper for `::fz_new_deflated_data_from_buffer()`.
33255: (4)              This function has out-params. Python/C# wrappers look like:
33256: (8)                  `fz_new_deflated_data_from_buffer(::fz_buffer *buffer,
::fz_deflate_level level)` => `(unsigned char *, size_t compressed_length)`
33257: (8)                  Compress the contents of a fz_buffer into a
33258: (8)                  new block malloced for that purpose. *compressed_length is
33259: (8)                  updated on exit to contain the size used. Ownership of the block
33260: (8)                  is returned from this function, and the caller is therefore
33261: (8)                  responsible for freeing it. The block may be considerably larger
33262: (8)                  than is actually required. The caller is free to fz_realloc it
33263: (8)                  down if it wants to.
33264: (4)              """
33265: (4)              return _mupdf.fz_new_deflated_data_from_buffer(compressed_length, buffer,
level)
33266: (0)          def fz_new_device_of_size(size):
33267: (4)              r"""
33268: (4)              Class-aware wrapper for `::fz_new_device_of_size()`.
33269: (8)                  Devices are created by calls to device implementations, for
33270: (8)                  instance: foo_new_device(). These will be implemented by calling
33271: (8)                  fz_new_derived_device(ctx, foo_device) where foo_device is a
33272: (8)                  structure "derived from" fz_device, for instance
33273: (8)                  typedef struct { fz_device base;  ...extras...} foo_device;
33274: (4)              """
33275: (4)              return _mupdf.fz_new_device_of_size(size)
33276: (0)          def fz_new_display_list(mediabox):
33277: (4)              r"""
33278: (4)              Class-aware wrapper for `::fz_new_display_list()`.
33279: (8)                  Create an empty display list.
33280: (8)                  A display list contains drawing commands (text, images, etc.).
33281: (8)                  Use fz_new_list_device for populating the list.
33282: (8)                  mediabox: Bounds of the page (in points) represented by the
33283: (8)                  display list.
33284: (4)              """
33285: (4)              return _mupdf.fz_new_display_list(mediabox)
33286: (0)          def fz_new_display_list_from_page(page):
33287: (4)              r"""
33288: (4)              Class-aware wrapper for `::fz_new_display_list_from_page()`.
33289: (8)                  Create a display list.
33290: (8)                  Ownership of the display list is returned to the caller.
33291: (4)              """
33292: (4)              return _mupdf.fz_new_display_list_from_page(page)
33293: (0)          def fz_new_display_list_from_page_contents(page):
33294: (4)              r"""
33295: (4)              Class-aware wrapper for `::fz_new_display_list_from_page_contents()`.
33296: (8)                  Create a display list from page contents (no annotations).
33297: (8)                  Ownership of the display list is returned to the caller.
33298: (4)              """
33299: (4)              return _mupdf.fz_new_display_list_from_page_contents(page)
33300: (0)          def fz_new_display_list_from_page_number(doc, number):
33301: (4)              r""" Class-aware wrapper for
`::fz_new_display_list_from_page_number()`."""
33302: (4)              return _mupdf.fz_new_display_list_from_page_number(doc, number)
33303: (0)          def fz_new_display_list_from_svg(buf, base_uri, dir, w, h):
33304: (4)              r"""
33305: (4)              Class-aware wrapper for `::fz_new_display_list_from_svg()`.
33306: (4)              This function has out-params. Python/C# wrappers look like:
33307: (8)                  `fz_new_display_list_from_svg(::fz_buffer *buf, const char *base_uri,
::fz_archive *dir)` => `(fz_display_list *, float w, float h)`
33308: (8)                  Parse an SVG document into a display-list.
33309: (4)              """
33310: (4)              return _mupdf.fz_new_display_list_from_svg(buf, base_uri, dir, w, h)
33311: (0)          def fz_new_display_list_from_svg_xml(xmldoc, xml, base_uri, dir, w, h):
33312: (4)              r"""
33313: (4)              Class-aware wrapper for `::fz_new_display_list_from_svg_xml()`.
33314: (4)              This function has out-params. Python/C# wrappers look like:
33315: (8)                  `fz_new_display_list_from_svg_xml(::fz_xml_doc *xmldoc, ::fz_xml *xml,
const char *base_uri, ::fz_archive *dir)` => `(fz_display_list *, float w, float h)`
33316: (8)                  Parse an SVG document into a display-list.
33317: (4)              """
```

```
33318: (4)             return _mupdf.fz_new_display_list_from_svg_xml(xmldoc, xml, base_uri, dir,
w, h)
33319: (0)         def fz_new_document_of_size(size):
33320: (4)             r"""
33321: (4)             Class-aware wrapper for `::fz_new_document_of_size()`.
33322: (8)                 New documents are typically created by calls like
33323: (8)                 foo_new_document(fz_context *ctx, ...). These work by
33324: (8)                 deriving a new document type from fz_document, for instance:
33325: (8)                 typedef struct { fz_document base; ...extras... } foo_document;
33326: (8)                 These are allocated by calling
33327: (8)                 fz_new_derived_document(ctx, foo_document)
33328: (4)             """
33329: (4)             return _mupdf.fz_new_document_of_size(size)
33330: (0)         def fz_new_draw_device(transform, dest):
33331: (4)             r"""
33332: (4)             Class-aware wrapper for `::fz_new_draw_device()`.
33333: (8)                 Create a device to draw on a pixmap.
33334: (8)                 dest: Target pixmap for the draw device. See fz_new_pixmap*
33335: (8)                 for how to obtain a pixmap. The pixmap is not cleared by the
33336: (8)                 draw device, see fz_clear_pixmap* for how to clear it prior to
33337: (8)                 calling fz_new_draw_device. Free the device by calling
33338: (8)                 fz_drop_device.
33339: (8)                 transform: Transform from user space in points to device space
33340: (8)                 in pixels.
33341: (4)             """
33342: (4)             return _mupdf.fz_new_draw_device(transform, dest)
33343: (0)         def fz_new_draw_device_type3(transform, dest):
33344: (4)             """ Class-aware wrapper for `::fz_new_draw_device_type3()`."""
33345: (4)             return _mupdf.fz_new_draw_device_type3(transform, dest)
33346: (0)         def fz_new_draw_device_with_bbox(transform, dest, clip):
33347: (4)             r"""
33348: (4)             Class-aware wrapper for `::fz_new_draw_device_with_bbox()`.
33349: (8)                 Create a device to draw on a pixmap.
33350: (8)                 dest: Target pixmap for the draw device. See fz_new_pixmap*
33351: (8)                 for how to obtain a pixmap. The pixmap is not cleared by the
33352: (8)                 draw device, see fz_clear_pixmap* for how to clear it prior to
33353: (8)                 calling fz_new_draw_device. Free the device by calling
33354: (8)                 fz_drop_device.
33355: (8)                 transform: Transform from user space in points to device space
33356: (8)                 in pixels.
33357: (8)                 clip: Bounding box to restrict any marking operations of the
33358: (8)                 draw device.
33359: (4)             """
33360: (4)             return _mupdf.fz_new_draw_device_with_bbox(transform, dest, clip)
33361: (0)         def fz_new_draw_device_with_bbox_proof(transform, dest, clip, cs):
33362: (4)             r"""
33363: (4)             Class-aware wrapper for `::fz_new_draw_device_with_bbox_proof()`.
33364: (8)                 Create a device to draw on a pixmap.
33365: (8)                 dest: Target pixmap for the draw device. See fz_new_pixmap*
33366: (8)                 for how to obtain a pixmap. The pixmap is not cleared by the
33367: (8)                 draw device, see fz_clear_pixmap* for how to clear it prior to
33368: (8)                 calling fz_new_draw_device. Free the device by calling
33369: (8)                 fz_drop_device.
33370: (8)                 transform: Transform from user space in points to device space
33371: (8)                 in pixels.
33372: (8)                 clip: Bounding box to restrict any marking operations of the
33373: (8)                 draw device.
33374: (8)                 proof_cs: Color space to render to prior to mapping to color
33375: (8)                 space defined by pixmap.
33376: (4)             """
33377: (4)             return _mupdf.fz_new_draw_device_with_bbox_proof(transform, dest, clip,
cs)
33378: (0)         def fz_new_draw_device_with_options(options, mediabox, pixmap):
33379: (4)             r"""
33380: (4)             Class-aware wrapper for `::fz_new_draw_device_with_options()`.
33381: (4)             This function has out-params. Python/C# wrappers look like:
33382: (8)                 `fz_new_draw_device_with_options(const ::fz_draw_options *options,
::fz_rect mediabox, ::fz_pixmap **pixmap)` => `(fz_device *)`
33383: (8)                 Create a new pixmap and draw device, using the specified options.
```

```
33384: (8)                  options: Options to configure the draw device, and choose the
33385: (8)                  resolution and colorspace.
33386: (8)                  mediabox: The bounds of the page in points.
33387: (8)                  pixmap: An out parameter containing the newly created pixmap.
33388: (4)              """
33389: (4)              return _mupdf.fz_new_draw_device_with_options(options, mediabox, pixmap)
33390: (0)          def fz_new_draw_device_with_proof(transform, dest, proof_cs):
33391: (4)              r"""
33392: (4)              Class-aware wrapper for `::fz_new_draw_device_with_proof()`.
33393: (8)                  Create a device to draw on a pixmap.
33394: (8)                  dest: Target pixmap for the draw device. See fz_new_pixmap*
33395: (8)                  for how to obtain a pixmap. The pixmap is not cleared by the
33396: (8)                  draw device, see fz_clear_pixmap* for how to clear it prior to
33397: (8)                  calling fz_new_draw_device. Free the device by calling
33398: (8)                  fz_drop_device.
33399: (8)                  transform: Transform from user space in points to device space
33400: (8)                  in pixels.
33401: (8)                  proof_cs: Intermediate color space to map though when mapping to
33402: (8)                  color space defined by pixmap.
33403: (4)              """
33404: (4)              return _mupdf.fz_new_draw_device_with_proof(transform, dest, proof_cs)
33405: (0)          def fz_new_font_from_buffer(name, buffer, index, use_glyph_bbox):
33406: (4)              r"""
33407: (4)              Class-aware wrapper for `::fz_new_font_from_buffer()`.
33408: (8)                  Create a new font from a font file in a fz_buffer.
33409: (8)                  Fonts created in this way, will be eligible for embedding by default.
33410: (8)                  name: Name of font (leave NULL to use name from font).
33411: (8)                  buffer: Buffer to load from.
33412: (8)                  index: Which font from the file to load (0 for default).
33413: (8)                  use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
33414: (8)                  Returns new font handle, or throws exception on error.
33415: (4)              """
33416: (4)              return _mupdf.fz_new_font_from_buffer(name, buffer, index, use_glyph_bbox)
33417: (0)          def fz_new_font_from_file(name, path, index, use_glyph_bbox):
33418: (4)              r"""
33419: (4)              Class-aware wrapper for `::fz_new_font_from_file()`.
33420: (8)                  Create a new font from a font file.
33421: (8)                  Fonts created in this way, will be eligible for embedding by default.
33422: (8)                  name: Name of font (leave NULL to use name from font).
33423: (8)                  path: File path to load from.
33424: (8)                  index: Which font from the file to load (0 for default).
33425: (8)                  use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
33426: (8)                  Returns new font handle, or throws exception on error.
33427: (4)              """
33428: (4)              return _mupdf.fz_new_font_from_file(name, path, index, use_glyph_bbox)
33429: (0)          def fz_new_font_from_memory(name, data, len, index, use_glyph_bbox):
33430: (4)              r"""
33431: (4)              Class-aware wrapper for `::fz_new_font_from_memory()`.
33432: (8)                  Create a new font from a font file in memory.
33433: (8)                  Fonts created in this way, will be eligible for embedding by default.
33434: (8)                  name: Name of font (leave NULL to use name from font).
33435: (8)                  data: Pointer to the font file data.
33436: (8)                  len: Length of the font file data.
33437: (8)                  index: Which font from the file to load (0 for default).
33438: (8)                  use_glyph_box: 1 if we should use the glyph bbox, 0 otherwise.
33439: (8)                  Returns new font handle, or throws exception on error.
33440: (4)              """
33441: (4)              return _mupdf.fz_new_font_from_memory(name, data, len, index,
use_glyph_bbox)
33442: (0)          def fz_new_function_of_size(size, size2, m, n, eval, drop):
33443: (4)              r""" Class-aware wrapper for `::fz_new_function_of_size()`."""
33444: (4)              return _mupdf.fz_new_function_of_size(size, size2, m, n, eval, drop)
33445: (0)          def fz_new_icc_colorspace(type, flags, name, buf):
33446: (4)              r"""
33447: (4)              Class-aware wrapper for `::fz_new_icc_colorspace()`.
33448: (8)                  Create a colorspace from an ICC profile supplied in buf.
33449: (8)                  Limited checking is done to ensure that the colorspace type is
33450: (8)                  appropriate for the supplied ICC profile.
33451: (8)                  An additional reference is taken to buf, which will be dropped
```

```
33452: (8)                  on destruction. Ownership is NOT passed in.
33453: (8)                  The returned reference should be dropped when it is finished
33454: (8)                  with.
33455: (8)                  Colorspaces are immutable once created.
33456: (4)              """
33457: (4)              return _mupdf.fz_new_icc_colorspace(type, flags, name, buf)
33458: (0)          def fz_new_image_from_buffer(buffer):
33459: (4)              r"""
33460: (4)              Class-aware wrapper for `::fz_new_image_from_buffer()`.
33461: (8)                  Create a new image from a
33462: (8)                  buffer of data, inferring its type from the format
33463: (8)                  of the data.
33464: (4)              """
33465: (4)              return _mupdf.fz_new_image_from_buffer(buffer)
33466: (0)          def fz_new_image_from_compressed_buffer(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, buffer, mask):
33467: (4)              r"""
33468: (4)              Class-aware wrapper for `::fz_new_image_from_compressed_buffer()`.
33469: (8)                  Create an image based on
33470: (8)                  the data in the supplied compressed buffer.
33471: (8)                  w,h: Width and height of the created image.
33472: (8)                  bpc: Bits per component.
33473: (8)                  colorspace: The colorspace (determines the number of components,
33474: (8)                  and any color conversions required while decoding).
33475: (8)                  xres, yres: The X and Y resolutions respectively.
33476: (8)                  interpolate: 1 if interpolation should be used when decoding
33477: (8)                  this image, 0 otherwise.
33478: (8)                  imagemask: 1 if this is an imagemask (i.e. transparency bitmap
33479: (8)                  mask), 0 otherwise.
33480: (8)                  decode: NULL, or a pointer to to a decode array. The default
33481: (8)                  decode array is [0 1] (repeated n times, for n color components).
33482: (8)                  colorkey: NULL, or a pointer to a colorkey array. The default
33483: (8)                  colorkey array is [0 255] (repeated n times, for n color
33484: (8)                  components).
33485: (8)                  buffer: Buffer of compressed data and compression parameters.
33486: (8)                  Ownership of this reference is passed in.
33487: (8)                  mask: NULL, or another image to use as a mask for this one.
33488: (8)                  A new reference is taken to this image. Supplying a masked
33489: (8)                  image as a mask to another image is illegal!
33490: (4)              """
33491: (4)              return _mupdf.fz_new_image_from_compressed_buffer(w, h, bpc, colorspace,
xres, yres, interpolate, imagemask, decode, colorkey, buffer, mask)
33492: (0)          def fz_new_image_from_compressed_buffer2(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, buffer, mask):
33493: (4)              r"""
33494: (4)              Class-aware wrapper for `::fz_new_image_from_compressed_buffer2()`.
Swig-friendly wrapper for fz_new_image_from_compressed_buffer(),
33495: (4)              uses specified `decode` and `colorkey` if they are not null (in which
33496: (4)              case we assert that they have size `2*fz_colorspace_n(colorspace)`).
33497: (4)              """
33498: (4)              return _mupdf.fz_new_image_from_compressed_buffer2(w, h, bpc, colorspace,
xres, yres, interpolate, imagemask, decode, colorkey, buffer, mask)
33499: (0)          def fz_new_image_from_display_list(w, h, list):
33500: (4)              r"""
33501: (4)              Class-aware wrapper for `::fz_new_image_from_display_list()`.
33502: (8)                  Create a new image from a display list.
33503: (8)                  w, h: The conceptual width/height of the image.
33504: (8)                  transform: The matrix that needs to be applied to the given
33505: (8)                  list to make it render to the unit square.
33506: (8)                  list: The display list.
33507: (4)              """
33508: (4)              return _mupdf.fz_new_image_from_display_list(w, h, list)
33509: (0)          def fz_new_image_from_file(path):
33510: (4)              r"""
33511: (4)              Class-aware wrapper for `::fz_new_image_from_file()`.
33512: (8)                  Create a new image from the contents
33513: (8)                  of a file, inferring its type from the format of the
33514: (8)                  data.
33515: (4)              """
```

```
33516: (4)             return _mupdf.fz_new_image_from_file(path)
33517: (0)         def fz_new_image_from_pixmap(pixmap, mask):
33518: (4)             r"""
33519: (4)             Class-aware wrapper for `::fz_new_image_from_pixmap()`.
33520: (8)                 Create an image from the given
33521: (8)                 pixmap.
33522: (8)                 pixmap: The pixmap to base the image upon. A new reference
33523: (8)                 to this is taken.
33524: (8)                 mask: NULL, or another image to use as a mask for this one.
33525: (8)                 A new reference is taken to this image. Supplying a masked
33526: (8)                 image as a mask to another image is illegal!
33527: (4)             """
33528: (4)             return _mupdf.fz_new_image_from_pixmap(pixmap, mask)
33529: (0)         def fz_new_image_from_svg(buf, base_uri, dir):
33530: (4)             r"""
33531: (4)             Class-aware wrapper for `::fz_new_image_from_svg()`.
33532: (8)                 Create a scalable image from an SVG document.
33533: (4)             """
33534: (4)             return _mupdf.fz_new_image_from_svg(buf, base_uri, dir)
33535: (0)         def fz_new_image_from_svg_xml(xmldoc, xml, base_uri, dir):
33536: (4)             r"""
33537: (4)             Class-aware wrapper for `::fz_new_image_from_svg_xml()`.
33538: (8)                 Create a scalable image from an SVG document.
33539: (4)             """
33540: (4)             return _mupdf.fz_new_image_from_svg_xml(xmldoc, xml, base_uri, dir)
33541: (0)         def fz_new_image_of_size(w, h, bpc, colorspace, xres, yres, interpolate,
imagemask, decode, colorkey, mask, size, get_pixmap, get_size, drop):
33542: (4)             r"""
33543: (4)             Class-aware wrapper for `::fz_new_image_of_size()`.
33544: (8)                 Internal function to make a new fz_image structure
33545: (8)                 for a derived class.
33546: (8)                 w,h: Width and height of the created image.
33547: (8)                 bpc: Bits per component.
33548: (8)                 colorspace: The colorspace (determines the number of components,
33549: (8)                 and any color conversions required while decoding).
33550: (8)                 xres, yres: The X and Y resolutions respectively.
33551: (8)                 interpolate: 1 if interpolation should be used when decoding
33552: (8)                 this image, 0 otherwise.
33553: (8)                 imagemask: 1 if this is an imagemask (i.e. transparent), 0
33554: (8)                 otherwise.
33555: (8)                 decode: NULL, or a pointer to to a decode array. The default
33556: (8)                 decode array is [0 1] (repeated n times, for n color components).
33557: (8)                 colorkey: NULL, or a pointer to a colorkey array. The default
33558: (8)                 colorkey array is [0 255] (repeated n times, for n color
33559: (8)                 components).
33560: (8)                 mask: NULL, or another image to use as a mask for this one.
33561: (8)                 A new reference is taken to this image. Supplying a masked
33562: (8)                 image as a mask to another image is illegal!
33563: (8)                 size: The size of the required allocated structure (the size of
33564: (8)                 the derived structure).
33565: (8)                 get: The function to be called to obtain a decoded pixmap.
33566: (8)                 get_size: The function to be called to return the storage size
33567: (8)                 used by this image.
33568: (8)                 drop: The function to be called to dispose of this image once
33569: (8)                 the last reference is dropped.
33570: (8)                 Returns a pointer to an allocated structure of the required size,
33571: (8)                 with the first sizeof(fz_image) bytes initialised as appropriate
33572: (8)                 given the supplied parameters, and the other bytes set to zero.
33573: (4)             """
33574: (4)             return _mupdf.fz_new_image_of_size(w, h, bpc, colorspace, xres, yres,
interpolate, imagemask, decode, colorkey, mask, size, get_pixmap, get_size, drop)
33575: (0)         def fz_new_indexed_colorspace(base, high, lookup):
33576: (4)             r"""
33577: (4)             Class-aware wrapper for `::fz_new_indexed_colorspace()`.
33578: (8)                 Create an indexed colorspace.
33579: (8)                 The supplied lookup table is high palette entries long. Each
33580: (8)                 entry is n bytes long, where n is given by the number of
33581: (8)                 colorants in the base colorspace, one byte per colorant.
33582: (8)                 Ownership of lookup is passed it; it will be freed on
```

```
33583: (8)                 destruction, so must be heap allocated.
33584: (8)                 The colorspace will keep an additional reference to the base
33585: (8)                 colorspace that will be dropped on destruction.
33586: (8)                 The returned reference should be dropped when it is finished
33587: (8)                 with.
33588: (8)                 Colorspaces are immutable once created.
33589: (4)             """
33590: (4)             return _mupdf.fz_new_indexed_colorspace(base, high, lookup)
33591: (0)         def fz_new_link_of_size(size, rect, uri):
33592: (4)             r"""
33593: (4)             Class-aware wrapper for `::fz_new_link_of_size()`.
33594: (8)                 Create a new link record.
33595: (8)                 next is set to NULL with the expectation that the caller will
33596: (8)                 handle the linked list setup. Internal function.
33597: (8)                 Different document types will be implemented by deriving from
33598: (8)                 fz_link. This macro allocates such derived structures, and
33599: (8)                 initialises the base sections.
33600: (4)             """
33601: (4)             return _mupdf.fz_new_link_of_size(size, rect, uri)
33602: (0)         def fz_new_list_device(list):
33603: (4)             r"""
33604: (4)             Class-aware wrapper for `::fz_new_list_device()`.
33605: (8)                 Create a rendering device for a display list.
33606: (8)                 When the device is rendering a page it will populate the
33607: (8)                 display list with drawing commands (text, images, etc.). The
33608: (8)                 display list can later be reused to render a page many times
33609: (8)                 without having to re-interpret the page from the document file
33610: (8)                 for each rendering. Once the device is no longer needed, free
33611: (8)                 it with fz_drop_device.
33612: (8)                 list: A display list that the list device takes a reference to.
33613: (4)             """
33614: (4)             return _mupdf.fz_new_list_device(list)
33615: (0)         def fz_new_multi_archive():
33616: (4)             r"""
33617: (4)             Class-aware wrapper for `::fz_new_multi_archive()`.
33618: (8)                 Create a new multi archive (initially empty).
33619: (4)             """
33620: (4)             return _mupdf.fz_new_multi_archive()
33621: (0)         def fz_new_ocr_device(target, ctm, mediabox, with_list, language, datadir,
progress, progress_arg):
33622: (4)             r"""
33623: (4)             Class-aware wrapper for `::fz_new_ocr_device()`.
33624: (8)                 Create a device to OCR the text on the page.
33625: (8)                 Renders the page internally to a bitmap that is then OCRd. Text
33626: (8)                 is then forwarded onto the target device.
33627: (8)                 target: The target device to receive the OCRd text.
33628: (8)                 ctm: The transform to apply to the mediabox to get the size for
33629: (8)                 the rendered page image. Also used to calculate the resolution
33630: (8)                 for the page image. In general, this will be the same as the CTM
33631: (8)                 that you pass to fz_run_page (or fz_run_display_list) to feed
33632: (8)                 this device.
33633: (8)                 mediabox: The mediabox (in points). Combined with the CTM to get
33634: (8)                 the bounds of the pixmap used internally for the rendered page
33635: (8)                 image.
33636: (8)                 with_list: If with_list is false, then all non-text operations
33637: (8)                 are forwarded instantly to the target device. This results in
33638: (8)                 the target device seeing all NON-text operations, followed by
33639: (8)                 all the text operations (derived from OCR).
33640: (8)                 If with_list is true, then all the marking operations are
33641: (8)                 collated into a display list which is then replayed to the
33642: (8)                 target device at the end.
33643: (8)                 language: NULL (for "eng"), or a pointer to a string to describe
33644: (8)                 the languages/scripts that should be used for OCR (e.g.
33645: (8)                 "eng,ara").
33646: (8)                 datadir: NULL (for ""), or a pointer to a path string otherwise
33647: (8)                 provided to Tesseract in the TESSDATA_PREFIX environment variable.
33648: (8)                 progress: NULL, or function to be called periodically to indicate
33649: (8)                 progress. Return 0 to continue, or 1 to cancel. progress_arg is
33650: (8)                 returned as the void *. The int is a value between 0 and 100 to
```

```
33651: (8)                  indicate progress.
33652: (8)                  progress_arg: A void * value to be parrotted back to the progress
33653: (8)                  function.
33654: (4)              """
33655: (4)              return _mupdf.fz_new_ocr_device(target, ctm, mediabox, with_list,
language, datadir, progress, progress_arg)
33656: (0)          def fz_new_outline():
33657: (4)              r"""
33658: (4)              Class-aware wrapper for `::fz_new_outline()`.
33659: (8)                  Create a new outline entry with zeroed fields for the caller
33660: (8)                  to fill in.
33661: (4)              """
33662: (4)              return _mupdf.fz_new_outline()
33663: (0)          def fz_new_page_of_size(size, doc):
33664: (4)              r"""
33665: (4)              Class-aware wrapper for `::fz_new_page_of_size()`.
33666: (8)                  Different document types will be implemented by deriving from
33667: (8)                  fz_page. This macro allocates such derived structures, and
33668: (8)                  initialises the base sections.
33669: (4)              """
33670: (4)              return _mupdf.fz_new_page_of_size(size, doc)
33671: (0)          def fz_new_path():
33672: (4)              r"""
33673: (4)              Class-aware wrapper for `::fz_new_path()`.
33674: (8)                  Create a new (empty) path structure.
33675: (4)              """
33676: (4)              return _mupdf.fz_new_path()
33677: (0)          def fz_new_pdf_document_from_fz_document(ptr):
33678: (4)              r""" Class-aware wrapper for
`::fz_new_pdf_document_from_fz_document()`."""
33679: (4)              return _mupdf.fz_new_pdf_document_from_fz_document(ptr)
33680: (0)          def fz_new_pixmap(cs, w, h, seps, alpha):
33681: (4)              r"""
33682: (4)              Class-aware wrapper for `::fz_new_pixmap()`.
33683: (8)                  Create a new pixmap, with its origin at (0,0)
33684: (8)                  cs: The colorspace to use for the pixmap, or NULL for an alpha
33685: (8)                  plane/mask.
33686: (8)                  w: The width of the pixmap (in pixels)
33687: (8)                  h: The height of the pixmap (in pixels)
33688: (8)                  seps: Details of separations.
33689: (8)                  alpha: 0 for no alpha, 1 for alpha.
33690: (8)                  Returns a pointer to the new pixmap. Throws exception on failure
33691: (8)                  to allocate.
33692: (4)              """
33693: (4)              return _mupdf.fz_new_pixmap(cs, w, h, seps, alpha)
33694: (0)          def fz_new_pixmap_from_alpha_channel(src):
33695: (4)              r""" Class-aware wrapper for `::fz_new_pixmap_from_alpha_channel()`."""
33696: (4)              return _mupdf.fz_new_pixmap_from_alpha_channel(src)
33697: (0)          def fz_new_pixmap_from_color_and_mask(color, mask):
33698: (4)              r""" Class-aware wrapper for `::fz_new_pixmap_from_color_and_mask()`."""
33699: (4)              return _mupdf.fz_new_pixmap_from_color_and_mask(color, mask)
33700: (0)          def fz_new_pixmap_from_display_list(list, ctm, cs, alpha):
33701: (4)              r"""
33702: (4)              Class-aware wrapper for `::fz_new_pixmap_from_display_list()`.
33703: (8)                  Render the page to a pixmap using the transform and colorspace.
33704: (8)                  Ownership of the pixmap is returned to the caller.
33705: (4)              """
33706: (4)              return _mupdf.fz_new_pixmap_from_display_list(list, ctm, cs, alpha)
33707: (0)          def fz_new_pixmap_from_display_list_with_separations(list, ctm, cs, seps,
alpha):
33708: (4)              r"""
33709: (4)              Class-aware wrapper for
`::fz_new_pixmap_from_display_list_with_separations()`.
33710: (8)                  Render the page contents with control over spot colors.
33711: (8)                  Ownership of the pixmap is returned to the caller.
33712: (4)              """
33713: (4)              return _mupdf.fz_new_pixmap_from_display_list_with_separations(list, ctm,
cs, seps, alpha)
33714: (0)          def fz_new_pixmap_from_page(page, ctm, cs, alpha):
```

```
33715: (4)              r""" Class-aware wrapper for `::fz_new_pixmap_from_page()`."""
33716: (4)              return _mupdf.fz_new_pixmap_from_page(page, ctm, cs, alpha)
33717: (0)          def fz_new_pixmap_from_page_contents(page, ctm, cs, alpha):
33718: (4)              r"""
33719: (4)              Class-aware wrapper for `::fz_new_pixmap_from_page_contents()`.
33720: (8)                  Render the page contents without annotations.
33721: (8)                  Ownership of the pixmap is returned to the caller.
33722: (4)              """
33723: (4)              return _mupdf.fz_new_pixmap_from_page_contents(page, ctm, cs, alpha)
33724: (0)          def fz_new_pixmap_from_page_contents_with_separations(page, ctm, cs, seps,
alpha):
33725: (4)              r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_contents_with_separations()`."""
33726: (4)              return _mupdf.fz_new_pixmap_from_page_contents_with_separations(page, ctm,
cs, seps, alpha)
33727: (0)          def fz_new_pixmap_from_page_number(doc, number, ctm, cs, alpha):
33728: (4)              r""" Class-aware wrapper for `::fz_new_pixmap_from_page_number()`."""
33729: (4)              return _mupdf.fz_new_pixmap_from_page_number(doc, number, ctm, cs, alpha)
33730: (0)          def fz_new_pixmap_from_page_number_with_separations(doc, number, ctm, cs,
seps, alpha):
33731: (4)              r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_number_with_separations()`."""
33732: (4)              return _mupdf.fz_new_pixmap_from_page_number_with_separations(doc, number,
ctm, cs, seps, alpha)
33733: (0)          def fz_new_pixmap_from_page_with_separations(page, ctm, cs, seps, alpha):
33734: (4)              r""" Class-aware wrapper for
`::fz_new_pixmap_from_page_with_separations()`."""
33735: (4)              return _mupdf.fz_new_pixmap_from_page_with_separations(page, ctm, cs,
seps, alpha)
33736: (0)          def fz_new_pixmap_from_pixmap(pixmap, rect):
33737: (4)              r"""
33738: (4)              Class-aware wrapper for `::fz_new_pixmap_from_pixmap()`.
33739: (8)                  Create a new pixmap that represents a subarea of the specified
33740: (8)                  pixmap. A reference is taken to this pixmap that will be dropped
33741: (8)                  on destruction.
33742: (8)                  The supplied rectangle must be wholly contained within the
33743: (8)                  original pixmap.
33744: (8)                  Returns a pointer to the new pixmap. Throws exception on failure
33745: (8)                  to allocate.
33746: (4)              """
33747: (4)              return _mupdf.fz_new_pixmap_from_pixmap(pixmap, rect)
33748: (0)          def fz_new_pixmap_with_bbox(colorspace, bbox, seps, alpha):
33749: (4)              r"""
33750: (4)              Class-aware wrapper for `::fz_new_pixmap_with_bbox()`.
33751: (8)                  Create a pixmap of a given size, location and pixel format.
33752: (8)                  The bounding box specifies the size of the created pixmap and
33753: (8)                  where it will be located. The colorspace determines the number
33754: (8)                  of components per pixel. Alpha is always present. Pixmaps are
33755: (8)                  reference counted, so drop references using fz_drop_pixmap.
33756: (8)                  colorspace: Colorspace format used for the created pixmap. The
33757: (8)                  pixmap will keep a reference to the colorspace.
33758: (8)                  bbox: Bounding box specifying location/size of created pixmap.
33759: (8)                  seps: Details of separations.
33760: (8)                  alpha: 0 for no alpha, 1 for alpha.
33761: (8)                  Returns a pointer to the new pixmap. Throws exception on failure
33762: (8)                  to allocate.
33763: (4)              """
33764: (4)              return _mupdf.fz_new_pixmap_with_bbox(colorspace, bbox, seps, alpha)
33765: (0)          def fz_new_pixmap_with_bbox_and_data(colorspace, rect, seps, alpha, samples):
33766: (4)              r"""
33767: (4)              Class-aware wrapper for `::fz_new_pixmap_with_bbox_and_data()`.
33768: (8)                  Create a pixmap of a given size, location and pixel format,
33769: (8)                  using the supplied data block.
33770: (8)                  The bounding box specifies the size of the created pixmap and
33771: (8)                  where it will be located. The colorspace determines the number
33772: (8)                  of components per pixel. Alpha is always present. Pixmaps are
33773: (8)                  reference counted, so drop references using fz_drop_pixmap.
33774: (8)                  colorspace: Colorspace format used for the created pixmap. The
33775: (8)                  pixmap will keep a reference to the colorspace.
```

```
(8)                 rect: Bounding box specifying location/size of created pixmap.
(8)                 seps: Details of separations.
(8)                 alpha: Number of alpha planes (0 or 1).
(8)                 samples: The data block to keep the samples in.
(8)                 Returns a pointer to the new pixmap. Throws exception on failure
(8)                 to allocate.
(4)             """
(4)             return _mupdf.fz_new_pixmap_with_bbox_and_data(colorspace, rect, seps, alpha, samples)
(0)         def fz_new_pixmap_with_data(colorspace, w, h, seps, alpha, stride, samples):
(4)             r"""
(4)             Class-aware wrapper for `::fz_new_pixmap_with_data()`.
(8)                 Create a new pixmap, with its origin at
(8)                 (0,0) using the supplied data block.
(8)                 cs: The colorspace to use for the pixmap, or NULL for an alpha
(8)                 plane/mask.
(8)                 w: The width of the pixmap (in pixels)
(8)                 h: The height of the pixmap (in pixels)
(8)                 seps: Details of separations.
(8)                 alpha: 0 for no alpha, 1 for alpha.
(8)                 stride: The byte offset from the pixel data in a row to the
(8)                 pixel data in the next row.
(8)                 samples: The data block to keep the samples in.
(8)                 Returns a pointer to the new pixmap. Throws exception on failure to
(8)                 allocate.
(4)             """
(4)             return _mupdf.fz_new_pixmap_with_data(colorspace, w, h, seps, alpha, stride, samples)
(0)         def fz_new_separations(controllable):
(4)             r"""
(4)             Class-aware wrapper for `::fz_new_separations()`.
(8)                 Create a new separations structure (initially empty)
(4)             """
(4)             return _mupdf.fz_new_separations(controllable)
(0)         def fz_new_stext_device(page, options):
(4)             r"""
(4)             Class-aware wrapper for `::fz_new_stext_device()`.
(8)                 Create a device to extract the text on a page.
(8)                 Gather the text on a page into blocks and lines.
(8)                 The reading order is taken from the order the text is drawn in
(8)                 the source file, so may not be accurate.
(8)                 page: The text page to which content should be added. This will
(8)                 usually be a newly created (empty) text page, but it can be one
(8)                 containing data already (for example when merging multiple
(8)                 pages, or watermarking).
(8)                 options: Options to configure the stext device.
(4)             """
(4)             return _mupdf.fz_new_stext_device(page, options)
(0)         def fz_new_store_context(max):
(4)             r"""
(4)             Class-aware wrapper for `::fz_new_store_context()`.
(8)                 Create a new store inside the context
(8)                 max: The maximum size (in bytes) that the store is allowed to
(8)                 grow to. FZ_STORE_UNLIMITED means no limit.
(4)             """
(4)             return _mupdf.fz_new_store_context(max)
(0)         def fz_new_stream(state, next, drop):
(4)             r"""
(4)             Class-aware wrapper for `::fz_new_stream()`.
(8)                 Create a new stream object with the given
(8)                 internal state and function pointers.
(8)                 state: Internal state (opaque to everything but implementation).
(8)                 next: Should provide the next set of bytes (up to max) of stream
(8)                 data. Return the number of bytes read, or EOF when there is no
(8)                 more data.
(8)                 drop: Should clean up and free the internal state. May not
(8)                 throw exceptions.
(4)             """
(4)             return _mupdf.fz_new_stream(state, next, drop)
```

```
33843: (0)          def fz_new_string(str):
33844: (4)              r""" Class-aware wrapper for `::fz_new_string()`."""
33845: (4)              return _mupdf.fz_new_string(str)
33846: (0)          def fz_new_stroke_state():
33847: (4)              r"""
33848: (4)              Class-aware wrapper for `::fz_new_stroke_state()`.
33849: (8)                  Create a new (empty) stroke state structure (with no dash
33850: (8)                  data) and return a reference to it.
33851: (8)                  Throws exception on failure to allocate.
33852: (4)              """
33853: (4)              return _mupdf.fz_new_stroke_state()
33854: (0)          def fz_new_stroke_state_with_dash_len(len):
33855: (4)              r"""
33856: (4)              Class-aware wrapper for `::fz_new_stroke_state_with_dash_len()`.
33857: (8)                  Create a new (empty) stroke state structure, with room for
33858: (8)                  dash data of the given length, and return a reference to it.
33859: (8)                  len: The number of dash elements to allow room for.
33860: (8)                  Throws exception on failure to allocate.
33861: (4)              """
33862: (4)              return _mupdf.fz_new_stroke_state_with_dash_len(len)
33863: (0)          def fz_new_svg_device(out, page_width, page_height, text_format,
reuse_images):
33864: (4)              r"""
33865: (4)              Class-aware wrapper for `::fz_new_svg_device()`.
33866: (8)                  Create a device that outputs (single page) SVG files to
33867: (8)                  the given output stream.
33868: (8)                  Equivalent to fz_new_svg_device_with_id passing id = NULL.
33869: (4)              """
33870: (4)              return _mupdf.fz_new_svg_device(out, page_width, page_height, text_format,
reuse_images)
33871: (0)          def fz_new_svg_device_with_id(out, page_width, page_height, text_format,
reuse_images, id):
33872: (4)              r"""
33873: (4)              Class-aware wrapper for `::fz_new_svg_device_with_id()`.
33874: (4)              This function has out-params. Python/C# wrappers look like:
33875: (8)                  `fz_new_svg_device_with_id(::fz_output *out, float page_width, float
page_height, int text_format, int reuse_images)` => `(fz_device *, int id)`
33876: (8)                  Create a device that outputs (single page) SVG files to
33877: (8)                  the given output stream.
33878: (8)                  output: The output stream to send the constructed SVG page to.
33879: (8)                  page_width, page_height: The page dimensions to use (in points).
33880: (8)                  text_format: How to emit text. One of the following values:
33881: (12)                     FZ_SVG_TEXT_AS_TEXT: As <text> elements with possible
33882: (12)                     layout errors and mismatching fonts.
33883: (12)                     FZ_SVG_TEXT_AS_PATH: As <path> elements with exact
33884: (12)                     visual appearance.
33885: (8)                  reuse_images: Share image resources using <symbol> definitions.
33886: (8)                  id: ID parameter to keep generated IDs unique across SVG files.
33887: (4)              """
33888: (4)              return _mupdf.fz_new_svg_device_with_id(out, page_width, page_height,
text_format, reuse_images, id)
33889: (0)          def fz_new_test_device(is_color, threshold, options, passthrough):
33890: (4)              r"""
33891: (4)              Class-aware wrapper for `::fz_new_test_device()`.
33892: (4)              This function has out-params. Python/C# wrappers look like:
33893: (8)                  `fz_new_test_device(float threshold, int options, ::fz_device
*passthrough)` => `(fz_device *, int is_color)`
33894: (8)                  Create a device to test for features.
33895: (8)                  Currently only tests for the presence of non-grayscale colors.
33896: (8)                  is_color: Possible values returned:
33897: (12)                     0: Definitely greyscale
33898: (12)                     1: Probably color (all colors were grey, but there
33899: (12)                     were images or shadings in a non grey colorspace).
33900: (12)                     2: Definitely color
33901: (8)                  threshold: The difference from grayscale that will be tolerated.
33902: (8)                  Typical values to use are either 0 (be exact) and 0.02 (allow an
33903: (8)                  imperceptible amount of slop).
33904: (8)                  options: A set of bitfield options, from the FZ_TEST_OPT set.
33905: (8)                  passthrough: A device to pass all calls through to, or NULL.
```

```
33906: (8)                  If set, then the test device can both test and pass through to
33907: (8)                  an underlying device (like, say, the display list device). This
33908: (8)                  means that a display list can be created and at the end we'll
33909: (8)                  know if it's colored or not.
33910: (8)                  In the absence of a passthrough device, the device will throw
33911: (8)                  an exception to stop page interpretation when color is found.
33912: (4)              """
33913: (4)              return _mupdf.fz_new_test_device(is_color, threshold, options,
passthrough)
33914: (0)          def fz_new_text():
33915: (4)              r"""
33916: (4)              Class-aware wrapper for `::fz_new_text()`.
33917: (8)                  Create a new empty fz_text object.
33918: (8)                  Throws exception on failure to allocate.
33919: (4)              """
33920: (4)              return _mupdf.fz_new_text()
33921: (0)          def fz_new_trace_device(out):
33922: (4)              r"""
33923: (4)              Class-aware wrapper for `::fz_new_trace_device()`.
33924: (8)                  Create a device to print a debug trace of all device calls.
33925: (4)              """
33926: (4)              return _mupdf.fz_new_trace_device(out)
33927: (0)          def fz_new_tree_archive(tree):
33928: (4)              r"""
33929: (4)              Class-aware wrapper for `::fz_new_tree_archive()`.
33930: (8)                  Create an archive that holds named buffers.
33931: (8)                  tree can either be a preformed tree with fz_buffers as values,
33932: (8)                  or it can be NULL for an empty tree.
33933: (4)              """
33934: (4)              return _mupdf.fz_new_tree_archive(tree)
33935: (0)          def fz_new_type3_font(name, matrix):
33936: (4)              r"""
33937: (4)              Class-aware wrapper for `::fz_new_type3_font()`.
33938: (8)                  Create a new (empty) type3 font.
33939: (8)                  name: Name of font (or NULL).
33940: (8)                  matrix: Font matrix.
33941: (8)                  Returns a new font handle, or throws exception on
33942: (8)                  allocation failure.
33943: (4)              """
33944: (4)              return _mupdf.fz_new_type3_font(name, matrix)
33945: (0)          def fz_new_xhtml_document_from_document(old_doc, opts):
33946: (4)              r"""
33947: (4)              Class-aware wrapper for `::fz_new_xhtml_document_from_document()`.
33948: (8)                  Use text extraction to convert the input document into XHTML,
33949: (8)                  then open the result as a new document that can be reflowed.
33950: (4)              """
33951: (4)              return _mupdf.fz_new_xhtml_document_from_document(old_doc, opts)
33952: (0)          def fz_new_xmltext_device(out):
33953: (4)              r"""
33954: (4)              Class-aware wrapper for `::fz_new_xmltext_device()`.
33955: (8)                  Create a device to output raw information.
33956: (4)              """
33957: (4)              return _mupdf.fz_new_xmltext_device(out)
33958: (0)          def fz_next_page(doc, loc):
33959: (4)              r"""
33960: (4)              Class-aware wrapper for `::fz_next_page()`.
33961: (8)                  Function to get the location of the next page (allowing for the
33962: (8)                  end of chapters etc). If at the end of the document, returns the
33963: (8)                  current location.
33964: (4)              """
33965: (4)              return _mupdf.fz_next_page(doc, loc)
33966: (0)          def fz_normalize_vector(p):
33967: (4)              r"""
33968: (4)              Class-aware wrapper for `::fz_normalize_vector()`.
33969: (8)                  Normalize a vector to length one.
33970: (4)              """
33971: (4)              return _mupdf.fz_normalize_vector(p)
33972: (0)          def fz_open_a85d(chain):
33973: (4)              r"""
```

```
33974: (4)              Class-aware wrapper for `::fz_open_a85d()`.
33975: (8)                  a85d filter performs ASCII 85 Decoding of data read
33976: (8)                  from the chained filter.
33977: (4)              """
33978: (4)              return _mupdf.fz_open_a85d(chain)
33979: (0)          def fz_open_accelerated_document(filename, accel):
33980: (4)              r"""
33981: (4)              Class-aware wrapper for `::fz_open_accelerated_document()`.
33982: (8)                  Open a document file and read its basic structure so pages and
33983: (8)                  objects can be located. MuPDF will try to repair broken
33984: (8)                  documents (without actually changing the file contents).
33985: (8)                  The returned fz_document is used when calling most other
33986: (8)                  document related functions.
33987: (8)                  filename: a path to a file as it would be given to open(2).
33988: (4)              """
33989: (4)              return _mupdf.fz_open_accelerated_document(filename, accel)
33990: (0)          def fz_open_accelerated_document_with_stream(magic, stream, accel):
33991: (4)              r"""
33992: (4)              Class-aware wrapper for `::fz_open_accelerated_document_with_stream()`.
33993: (8)                  Open a document using the specified stream object rather than
33994: (8)                  opening a file on disk.
33995: (8)                  magic: a string used to detect document type; either a file name
33996: (8)                  or mime-type.
33997: (8)                  stream: a stream of the document contents.
33998: (8)                  accel: NULL, or a stream of the 'accelerator' contents for this
document.
33999: (8)                  NOTE: The caller retains ownership of 'stream' and 'accel' - the
document will
34000: (8)                  take its own references if required.
34001: (4)              """
34002: (4)              return _mupdf.fz_open_accelerated_document_with_stream(magic, stream,
accel)
34003: (0)          def fz_open_accelerated_document_with_stream_and_dir(magic, stream, accel,
dir):
34004: (4)              r"""
34005: (4)              Class-aware wrapper for
`::fz_open_accelerated_document_with_stream_and_dir()`.
34006: (8)                  Open a document using the specified stream object rather than
34007: (8)                  opening a file on disk.
34008: (8)                  magic: a string used to detect document type; either a file name
34009: (8)                  or mime-type.
34010: (8)                  stream: a stream of the document contents.
34011: (8)                  accel: NULL, or a stream of the 'accelerator' contents for this
document.
34012: (8)                  dir: NULL, or the 'directory context' for the stream contents.
34013: (8)                  NOTE: The caller retains ownership of 'stream', 'accel' and 'dir' -
the document will
34014: (8)                  take its own references if required.
34015: (4)              """
34016: (4)              return _mupdf.fz_open_accelerated_document_with_stream_and_dir(magic,
stream, accel, dir)
34017: (0)          def fz_open_aesd(chain, key, keylen):
34018: (4)              r"""
34019: (4)              Class-aware wrapper for `::fz_open_aesd()`.
34020: (8)                  aesd filter performs AES decoding of data read from the chained
34021: (8)                  filter using the supplied key.
34022: (4)              """
34023: (4)              return _mupdf.fz_open_aesd(chain, key, keylen)
34024: (0)          def fz_open_ahxd(chain):
34025: (4)              r"""
34026: (4)              Class-aware wrapper for `::fz_open_ahxd()`.
34027: (8)                  ahxd filter performs ASCII Hex decoding of data read
34028: (8)                  from the chained filter.
34029: (4)              """
34030: (4)              return _mupdf.fz_open_ahxd(chain)
34031: (0)          def fz_open_arc4(chain, key, keylen):
34032: (4)              r"""
34033: (4)              Class-aware wrapper for `::fz_open_arc4()`.
34034: (8)                  arc4 filter performs RC4 decoding of data read from the chained
```

```
34035: (8)                 filter using the supplied key.
34036: (4)             """
34037: (4)             return _mupdf.fz_open_arc4(chain, key, keylen)
34038: (0)         def fz_open_archive(filename):
34039: (4)             r"""
34040: (4)             Class-aware wrapper for `::fz_open_archive()`.
34041: (8)                 Open a zip or tar archive
34042: (8)                 Open a file and identify its archive type based on the archive
34043: (8)                 signature contained inside.
34044: (8)                 filename: a path to a file as it would be given to open(2).
34045: (4)             """
34046: (4)             return _mupdf.fz_open_archive(filename)
34047: (0)         def fz_open_archive_entry(arch, name):
34048: (4)             r"""
34049: (4)             Class-aware wrapper for `::fz_open_archive_entry()`.
34050: (8)                 Opens an archive entry as a stream.
34051: (8)                 name: Entry name to look for, this must be an exact match to
34052: (8)                 the entry name in the archive.
34053: (8)                 Throws an exception if a matching entry cannot be found.
34054: (4)             """
34055: (4)             return _mupdf.fz_open_archive_entry(arch, name)
34056: (0)         def fz_open_archive_with_stream(file):
34057: (4)             r"""
34058: (4)             Class-aware wrapper for `::fz_open_archive_with_stream()`.
34059: (8)                 Open zip or tar archive stream.
34060: (8)                 Open an archive using a seekable stream object rather than
34061: (8)                 opening a file or directory on disk.
34062: (4)             """
34063: (4)             return _mupdf.fz_open_archive_with_stream(file)
34064: (0)         def fz_open_buffer(buf):
34065: (4)             r"""
34066: (4)             Class-aware wrapper for `::fz_open_buffer()`.
34067: (8)                 Open a buffer as a stream.
34068: (8)                 buf: The buffer to open. Ownership of the buffer is NOT passed
34069: (8)                 in (this function takes its own reference).
34070: (8)                 Returns pointer to newly created stream. May throw exceptions on
34071: (8)                 failure to allocate.
34072: (4)             """
34073: (4)             return _mupdf.fz_open_buffer(buf)
34074: (0)         def fz_open_cfb_archive(filename):
34075: (4)             r"""
34076: (4)             Class-aware wrapper for `::fz_open_cfb_archive()`.
34077: (8)                 Open a cfb file as an archive.
34078: (8)                 An exception is thrown if the file is not recognised as a cfb.
34079: (8)                 filename: a path to an archive file as it would be given to
34080: (8)                 open(2).
34081: (4)             """
34082: (4)             return _mupdf.fz_open_cfb_archive(filename)
34083: (0)         def fz_open_cfb_archive_with_stream(file):
34084: (4)             r"""
34085: (4)             Class-aware wrapper for `::fz_open_cfb_archive_with_stream()`.
34086: (8)                 Open a cfb file as an archive.
34087: (8)                 Open an archive using a seekable stream object rather than
34088: (8)                 opening a file or directory on disk.
34089: (8)                 An exception is thrown if the file is not recognised as a chm.
34090: (4)             """
34091: (4)             return _mupdf.fz_open_cfb_archive_with_stream(file)
34092: (0)         def fz_open_compressed_buffer(arg_0):
34093: (4)             r"""
34094: (4)             Class-aware wrapper for `::fz_open_compressed_buffer()`.
34095: (8)                 Open a stream to read the decompressed version of a buffer.
34096: (4)             """
34097: (4)             return _mupdf.fz_open_compressed_buffer(arg_0)
34098: (0)         def fz_open_concat(max, pad):
34099: (4)             r"""
34100: (4)             Class-aware wrapper for `::fz_open_concat()`.
34101: (8)                 Concat filter concatenates several streams into one.
34102: (4)             """
34103: (4)             return _mupdf.fz_open_concat(max, pad)
```

```
34104: (0)          def fz_open_dctd(chain, color_transform, invert_cmyk, l2factor, jpegtables):
34105: (4)              r"""
34106: (4)              Class-aware wrapper for `::fz_open_dctd()`.
34107: (8)                  dctd filter performs DCT (JPEG) decoding of data read
34108: (8)                  from the chained filter.
34109: (8)                  color_transform implements the PDF color_transform option
34110: (12)                     use -1 for default behavior
34111: (12)                     use 0 to disable YUV-RGB / YCCK-CMYK transforms
34112: (12)                     use 1 to enable YUV-RGB / YCCK-CMYK transforms
34113: (8)                  invert_cmyk implements the necessary inversion for Photoshop CMYK
images
34114: (12)                     use 0 if embedded in PDF
34115: (12)                     use 1 if not embedded in PDF
34116: (8)                  For subsampling on decode, set l2factor to the log2 of the
34117: (8)                  reduction required (therefore 0 = full size decode).
34118: (8)                  jpegtables is an optional stream from which the JPEG tables
34119: (8)                  can be read. Use NULL if not required.
34120: (4)              """
34121: (4)              return _mupdf.fz_open_dctd(chain, color_transform, invert_cmyk, l2factor,
jpegtables)
34122: (0)          def fz_open_directory(path):
34123: (4)              r"""
34124: (4)              Class-aware wrapper for `::fz_open_directory()`.
34125: (8)                  Open a directory as if it was an archive.
34126: (8)                  A special case where a directory is opened as if it was an
34127: (8)                  archive.
34128: (8)                  Note that for directories it is not possible to retrieve the
34129: (8)                  number of entries or list the entries. It is however possible
34130: (8)                  to check if the archive has a particular entry.
34131: (8)                  path: a path to a directory as it would be given to opendir(3).
34132: (4)              """
34133: (4)              return _mupdf.fz_open_directory(path)
34134: (0)          def fz_open_document(filename):
34135: (4)              r"""
34136: (4)              Class-aware wrapper for `::fz_open_document()`.
34137: (8)                  Open a document file and read its basic structure so pages and
34138: (8)                  objects can be located. MuPDF will try to repair broken
34139: (8)                  documents (without actually changing the file contents).
34140: (8)                  The returned fz_document is used when calling most other
34141: (8)                  document related functions.
34142: (8)                  filename: a path to a file as it would be given to open(2).
34143: (4)              """
34144: (4)              return _mupdf.fz_open_document(filename)
34145: (0)          def fz_open_document_with_buffer(magic, buffer):
34146: (4)              r"""
34147: (4)              Class-aware wrapper for `::fz_open_document_with_buffer()`.
34148: (8)                  Open a document using a buffer rather than opening a file on disk.
34149: (4)              """
34150: (4)              return _mupdf.fz_open_document_with_buffer(magic, buffer)
34151: (0)          def fz_open_document_with_stream(magic, stream):
34152: (4)              r"""
34153: (4)              Class-aware wrapper for `::fz_open_document_with_stream()`.
34154: (8)                  Open a document using the specified stream object rather than
34155: (8)                  opening a file on disk.
34156: (8)                  magic: a string used to detect document type; either a file name
34157: (8)                  or mime-type.
34158: (8)                  stream: a stream representing the contents of the document file.
34159: (8)                  NOTE: The caller retains ownership of 'stream' - the document will
take its
34160: (8)                  own reference if required.
34161: (4)              """
34162: (4)              return _mupdf.fz_open_document_with_stream(magic, stream)
34163: (0)          def fz_open_document_with_stream_and_dir(magic, stream, dir):
34164: (4)              r"""
34165: (4)              Class-aware wrapper for `::fz_open_document_with_stream_and_dir()`.
34166: (8)                  Open a document using the specified stream object rather than
34167: (8)                  opening a file on disk.
34168: (8)                  magic: a string used to detect document type; either a file name
34169: (8)                  or mime-type.
```

```
34170: (8)                  stream: a stream representing the contents of the document file.
34171: (8)                  dir: a 'directory context' for those filetypes that need it.
34172: (8)                  NOTE: The caller retains ownership of 'stream' and 'dir' - the
document will
34173: (8)                  take its own references if required.
34174: (4)              """
34175: (4)              return _mupdf.fz_open_document_with_stream_and_dir(magic, stream, dir)
34176: (0)          def fz_open_endstream_filter(chain, len, offset):
34177: (4)              r"""
34178: (4)              Class-aware wrapper for `::fz_open_endstream_filter()`.
34179: (8)                  The endstream filter reads a PDF substream, and starts to look
34180: (8)                  for an 'endstream' token after the specified length.
34181: (4)              """
34182: (4)              return _mupdf.fz_open_endstream_filter(chain, len, offset)
34183: (0)          def fz_open_faxd(chain, k, end_of_line, encoded_byte_align, columns, rows,
end_of_block, black_is_1):
34184: (4)              r"""
34185: (4)              Class-aware wrapper for `::fz_open_faxd()`.
34186: (8)                  faxd filter performs FAX decoding of data read from
34187: (8)                  the chained filter.
34188: (8)                  k: see fax specification (fax default is 0).
34189: (8)                  end_of_line: whether we expect end of line markers (fax default
34190: (8)                  is 0).
34191: (8)                  encoded_byte_align: whether we align to bytes after each line
34192: (8)                  (fax default is 0).
34193: (8)                  columns: how many columns in the image (fax default is 1728).
34194: (8)                  rows: 0 for unspecified or the number of rows of data to expect.
34195: (8)                  end_of_block: whether we expect end of block markers (fax
34196: (8)                  default is 1).
34197: (8)                  black_is_1: determines the polarity of the image (fax default is
34198: (8)                  0).
34199: (4)              """
34200: (4)              return _mupdf.fz_open_faxd(chain, k, end_of_line, encoded_byte_align,
columns, rows, end_of_block, black_is_1)
34201: (0)          def fz_open_file(filename):
34202: (4)              r"""
34203: (4)              Class-aware wrapper for `::fz_open_file()`.
34204: (8)                  Open the named file and wrap it in a stream.
34205: (8)                  filename: Path to a file. On non-Windows machines the filename
34206: (8)                  should be exactly as it would be passed to fopen(2). On Windows
34207: (8)                  machines, the path should be UTF-8 encoded so that non-ASCII
34208: (8)                  characters can be represented. Other platforms do the encoding
34209: (8)                  as standard anyway (and in most cases, particularly for MacOS
34210: (8)                  and Linux, the encoding they use is UTF-8 anyway).
34211: (4)              """
34212: (4)              return _mupdf.fz_open_file(filename)
34213: (0)          def fz_open_file_autodelete(filename):
34214: (4)              r"""
34215: (4)              Class-aware wrapper for `::fz_open_file_autodelete()`.
34216: (8)                  Do the same as fz_open_file, but delete the file upon close.
34217: (4)              """
34218: (4)              return _mupdf.fz_open_file_autodelete(filename)
34219: (0)          def fz_open_file_ptr_no_close(file):
34220: (4)              r"""
34221: (4)              Class-aware wrapper for `::fz_open_file_ptr_no_close()`.
34222: (8)                  Create a stream from a FILE * that will not be closed
34223: (8)                  when the stream is dropped.
34224: (4)              """
34225: (4)              return _mupdf.fz_open_file_ptr_no_close(file)
34226: (0)          def fz_open_flated(chain, window_bits):
34227: (4)              r"""
34228: (4)              Class-aware wrapper for `::fz_open_flated()`.
34229: (8)                  flated filter performs LZ77 decoding (inflating) of data read
34230: (8)                  from the chained filter.
34231: (8)                  window_bits: How large a decompression window to use. Typically
34232: (8)                  15. A negative number, -n, means to use n bits, but to expect
34233: (8)                  raw data with no header.
34234: (4)              """
34235: (4)              return _mupdf.fz_open_flated(chain, window_bits)
```

```
34236: (0)          def fz_open_image_decomp_stream(arg_0, arg_1, l2factor):
34237: (4)              r"""
34238: (4)              Class-aware wrapper for `::fz_open_image_decomp_stream()`.
34239: (4)              This function has out-params. Python/C# wrappers look like:
34240: (8)                  `fz_open_image_decomp_stream(::fz_stream *arg_0,
::fz_compression_params *arg_1)` => `(fz_stream *, int l2factor)`
34241: (8)                  Open a stream to read the decompressed version of another stream
34242: (8)                  with optional log2 subsampling.
34243: (4)              """
34244: (4)              return _mupdf.fz_open_image_decomp_stream(arg_0, arg_1, l2factor)
34245: (0)          def fz_open_image_decomp_stream_from_buffer(arg_0, l2factor):
34246: (4)              r"""
34247: (4)              Class-aware wrapper for `::fz_open_image_decomp_stream_from_buffer()`.
34248: (4)              This function has out-params. Python/C# wrappers look like:
34249: (8)                  `fz_open_image_decomp_stream_from_buffer(::fz_compressed_buffer
*arg_0)` => `(fz_stream *, int l2factor)`
34250: (8)                  Open a stream to read the decompressed version of a buffer,
34251: (8)                  with optional log2 subsampling.
34252: (8)                  l2factor = NULL for no subsampling, or a pointer to an integer
34253: (8)                  containing the maximum log2 subsample factor acceptable (0 =
34254: (8)                  none, 1 = halve dimensions, 2 = quarter dimensions etc). If
34255: (8)                  non-NULL, then *l2factor will be updated on exit with the actual
34256: (8)                  log2 subsample factor achieved.
34257: (4)              """
34258: (4)              return _mupdf.fz_open_image_decomp_stream_from_buffer(arg_0, l2factor)
34259: (0)          def fz_open_jbig2d(chain, globals, embedded):
34260: (4)              r"""
34261: (4)              Class-aware wrapper for `::fz_open_jbig2d()`.
34262: (8)                  Open a filter that performs jbig2 decompression on the chained
34263: (8)                  stream, using the optional globals record.
34264: (4)              """
34265: (4)              return _mupdf.fz_open_jbig2d(chain, globals, embedded)
34266: (0)          def fz_open_leecher(chain, buf):
34267: (4)              r"""
34268: (4)              Class-aware wrapper for `::fz_open_leecher()`.
34269: (8)                  Attach a filter to a stream that will store any
34270: (8)                  characters read from the stream into the supplied buffer.
34271: (8)                  chain: The underlying stream to leech from.
34272: (8)                  buf: The buffer into which the read data should be appended.
34273: (8)                  The buffer will be resized as required.
34274: (8)                  Returns pointer to newly created stream. May throw exceptions on
34275: (8)                  failure to allocate.
34276: (4)              """
34277: (4)              return _mupdf.fz_open_leecher(chain, buf)
34278: (0)          def fz_open_libarchive_archive(filename):
34279: (4)              r"""
34280: (4)              Class-aware wrapper for `::fz_open_libarchive_archive()`.
34281: (8)                  Open an archive using libarchive.
34282: (8)                  An exception is thrown if the file is not supported by libarchive.
34283: (8)                  filename: a path to an archive file as it would be given to
34284: (8)                  open(2).
34285: (4)              """
34286: (4)              return _mupdf.fz_open_libarchive_archive(filename)
34287: (0)          def fz_open_libarchive_archive_with_stream(file):
34288: (4)              r"""
34289: (4)              Class-aware wrapper for `::fz_open_libarchive_archive_with_stream()`.
34290: (8)                  Open an archive using libarchive.
34291: (8)                  Open an archive using a seekable stream object rather than
34292: (8)                  opening a file or directory on disk.
34293: (8)                  An exception is thrown if the stream is not supported by libarchive.
34294: (4)              """
34295: (4)              return _mupdf.fz_open_libarchive_archive_with_stream(file)
34296: (0)          def fz_open_libarchived(chain):
34297: (4)              r"""
34298: (4)              Class-aware wrapper for `::fz_open_libarchived()`.
34299: (8)                  libarchived filter performs generic compressed decoding of data
34300: (8)                  in any format understood by libarchive from the chained filter.
34301: (8)                  This will throw an exception if libarchive is not built in, or
34302: (8)                  if the compression format is not recognised.
```

```
34303: (4)             """
34304: (4)             return _mupdf.fz_open_libarchived(chain)
34305: (0)         def fz_open_lzwd(chain, early_change, min_bits, reverse_bits, old_tiff):
34306: (4)             r"""
34307: (4)             Class-aware wrapper for `::fz_open_lzwd()`.
34308: (8)                 lzwd filter performs LZW decoding of data read from the chained
34309: (8)                 filter.
34310: (8)                 early_change: (Default 1) specifies whether to change codes 1
34311: (8)                 bit early.
34312: (8)                 min_bits: (Default 9) specifies the minimum number of bits to
34313: (8)                 use.
34314: (8)                 reverse_bits: (Default 0) allows for compatibility with gif and
34315: (8)                 old style tiffs (1).
34316: (8)                 old_tiff: (Default 0) allows for different handling of the clear
34317: (8)                 code, as found in old style tiffs.
34318: (4)             """
34319: (4)             return _mupdf.fz_open_lzwd(chain, early_change, min_bits, reverse_bits,
old_tiff)
34320: (0)         def fz_open_memory(data, len):
34321: (4)             r"""
34322: (4)             Class-aware wrapper for `::fz_open_memory()`.
34323: (8)                 Open a block of memory as a stream.
34324: (8)                 data: Pointer to start of data block. Ownership of the data
34325: (8)                 block is NOT passed in.
34326: (8)                 len: Number of bytes in data block.
34327: (8)                 Returns pointer to newly created stream. May throw exceptions on
34328: (8)                 failure to allocate.
34329: (4)             """
34330: (4)             return _mupdf.fz_open_memory(data, len)
34331: (0)         def fz_open_null_filter(chain, len, offset):
34332: (4)             r"""
34333: (4)             Class-aware wrapper for `::fz_open_null_filter()`.
34334: (8)                 The null filter reads a specified amount of data from the
34335: (8)                 substream.
34336: (4)             """
34337: (4)             return _mupdf.fz_open_null_filter(chain, len, offset)
34338: (0)         def fz_open_predict(chain, predictor, columns, colors, bpc):
34339: (4)             r"""
34340: (4)             Class-aware wrapper for `::fz_open_predict()`.
34341: (8)                 predict filter performs pixel prediction on data read from
34342: (8)                 the chained filter.
34343: (8)                 predictor: 1 = copy, 2 = tiff, other = inline PNG predictor
34344: (8)                 columns: width of image in pixels
34345: (8)                 colors: number of components.
34346: (8)                 bpc: bits per component (typically 8)
34347: (4)             """
34348: (4)             return _mupdf.fz_open_predict(chain, predictor, columns, colors, bpc)
34349: (0)         def fz_open_range_filter(chain, ranges, nranges):
34350: (4)             r"""
34351: (4)             Class-aware wrapper for `::fz_open_range_filter()`.
34352: (8)                 The range filter copies data from specified ranges of the
34353: (8)                 chained stream.
34354: (4)             """
34355: (4)             return _mupdf.fz_open_range_filter(chain, ranges, nranges)
34356: (0)         def fz_open_reflowed_document(underdoc, opts):
34357: (4)             r""" Class-aware wrapper for `::fz_open_reflowed_document()`."""
34358: (4)             return _mupdf.fz_open_reflowed_document(underdoc, opts)
34359: (0)         def fz_open_rld(chain):
34360: (4)             r"""
34361: (4)             Class-aware wrapper for `::fz_open_rld()`.
34362: (8)                 rld filter performs Run Length Decoding of data read
34363: (8)                 from the chained filter.
34364: (4)             """
34365: (4)             return _mupdf.fz_open_rld(chain)
34366: (0)         def fz_open_sgilog16(chain, w):
34367: (4)             r"""
34368: (4)             Class-aware wrapper for `::fz_open_sgilog16()`.
34369: (8)                 SGI Log 16bit (greyscale) decode from the chained filter.
34370: (8)                 Decodes lines of w pixels to 8bpp greyscale.
```

```
34371: (4)             """
34372: (4)             return _mupdf.fz_open_sgilog16(chain, w)
34373: (0)         def fz_open_sgilog24(chain, w):
34374: (4)             r"""
34375: (4)             Class-aware wrapper for `::fz_open_sgilog24()`.
34376: (8)                 SGI Log 24bit (LUV) decode from the chained filter.
34377: (8)                 Decodes lines of w pixels to 8bpc rgb.
34378: (4)             """
34379: (4)             return _mupdf.fz_open_sgilog24(chain, w)
34380: (0)         def fz_open_sgilog32(chain, w):
34381: (4)             r"""
34382: (4)             Class-aware wrapper for `::fz_open_sgilog32()`.
34383: (8)                 SGI Log 32bit (LUV) decode from the chained filter.
34384: (8)                 Decodes lines of w pixels to 8bpc rgb.
34385: (4)             """
34386: (4)             return _mupdf.fz_open_sgilog32(chain, w)
34387: (0)         def fz_open_tar_archive(filename):
34388: (4)             r"""
34389: (4)             Class-aware wrapper for `::fz_open_tar_archive()`.
34390: (8)                 Open a tar archive file.
34391: (8)                 An exception is thrown if the file is not a tar archive as
34392: (8)                 indicated by the presence of a tar signature.
34393: (8)                 filename: a path to a tar archive file as it would be given to
34394: (8)                 open(2).
34395: (4)             """
34396: (4)             return _mupdf.fz_open_tar_archive(filename)
34397: (0)         def fz_open_tar_archive_with_stream(file):
34398: (4)             r"""
34399: (4)             Class-aware wrapper for `::fz_open_tar_archive_with_stream()`.
34400: (8)                 Open a tar archive stream.
34401: (8)                 Open an archive using a seekable stream object rather than
34402: (8)                 opening a file or directory on disk.
34403: (8)                 An exception is thrown if the stream is not a tar archive as
34404: (8)                 indicated by the presence of a tar signature.
34405: (4)             """
34406: (4)             return _mupdf.fz_open_tar_archive_with_stream(file)
34407: (0)         def fz_open_thunder(chain, w):
34408: (4)             r"""
34409: (4)             Class-aware wrapper for `::fz_open_thunder()`.
34410: (8)                 4bit greyscale Thunderscan decoding from the chained filter.
34411: (8)                 Decodes lines of w pixels to 8bpp greyscale.
34412: (4)             """
34413: (4)             return _mupdf.fz_open_thunder(chain, w)
34414: (0)         def fz_open_zip_archive(path):
34415: (4)             r"""
34416: (4)             Class-aware wrapper for `::fz_open_zip_archive()`.
34417: (8)                 Open a zip archive file.
34418: (8)                 An exception is thrown if the file is not a zip archive as
34419: (8)                 indicated by the presence of a zip signature.
34420: (8)                 filename: a path to a zip archive file as it would be given to
34421: (8)                 open(2).
34422: (4)             """
34423: (4)             return _mupdf.fz_open_zip_archive(path)
34424: (0)         def fz_open_zip_archive_with_stream(file):
34425: (4)             r"""
34426: (4)             Class-aware wrapper for `::fz_open_zip_archive_with_stream()`.
34427: (8)                 Open a zip archive stream.
34428: (8)                 Open an archive using a seekable stream object rather than
34429: (8)                 opening a file or directory on disk.
34430: (8)                 An exception is thrown if the stream is not a zip archive as
34431: (8)                 indicated by the presence of a zip signature.
34432: (4)             """
34433: (4)             return _mupdf.fz_open_zip_archive_with_stream(file)
34434: (0)         def fz_opt_from_list(opt, optlist):
34435: (4)             r"""
34436: (4)             Class-aware wrapper for `::fz_opt_from_list()`.
34437: (8)                 Return the index of a (case-insensitive) option within an optlist.
34438: (8)                 For instance for optlist = "Foo|Bar|Baz", and  opt = "bar",
34439: (8)                 this would return 1.
```

```
34440: (8)                  If the optlist ends with "|*" then that is a catch all case and
34441: (8)                  matches all options allowing the caller to process it itself.
34442: (8)                  fz_optarg will be set to point to the option, and the return
34443: (8)                  value will be the index of the '*' option within that list.
34444: (8)                  If an optlist entry ends with ':' (e.g. "Foo:") then that option
34445: (8)                  may have suboptions appended to it (for example "JPG:80") and
34446: (8)                  fz_optarg will be set to point at "80". Otherwise fz_optarg will
34447: (8)                  be set to NULL.
34448: (8)                  In the event of no-match found, prints an error and returns -1.
34449: (4)              """
34450: (4)              return _mupdf.fz_opt_from_list(opt, optlist)
34451: (0)          def fz_option_eq(a, b):
34452: (4)              r"""
34453: (4)              Class-aware wrapper for `::fz_option_eq()`.
34454: (8)                  Check to see if an option, a, from a string matches a reference
34455: (8)                  option, b.
34456: (8)                  (i.e. a could be 'foo' or 'foo,bar...' etc, but b can only be
34457: (8)                  'foo'.)
34458: (4)              """
34459: (4)              return _mupdf.fz_option_eq(a, b)
34460: (0)          def fz_outline_glyph(font, gid, ctm):
34461: (4)              r"""
34462: (4)              Class-aware wrapper for `::fz_outline_glyph()`.
34463: (8)                  Look a glyph up from a font, and return the outline of the
34464: (8)                  glyph using the given transform.
34465: (8)                  The caller owns the returned path, and so is responsible for
34466: (8)                  ensuring that it eventually gets dropped.
34467: (4)              """
34468: (4)              return _mupdf.fz_outline_glyph(font, gid, ctm)
34469: (0)          def fz_outline_iterator_delete(iter):
34470: (4)              r"""
34471: (4)              Class-aware wrapper for `::fz_outline_iterator_delete()`.
34472: (8)                  Delete the current item.
34473: (8)                  This implicitly moves us to the 'next' item, and the return code is as
for fz_outline_iterator_next.
34474: (4)              """
34475: (4)              return _mupdf.fz_outline_iterator_delete(iter)
34476: (0)          def fz_outline_iterator_down(iter):
34477: (4)              r""" Class-aware wrapper for `::fz_outline_iterator_down()`."""
34478: (4)              return _mupdf.fz_outline_iterator_down(iter)
34479: (0)          def fz_outline_iterator_item(iter):
34480: (4)              r"""
34481: (4)              Class-aware wrapper for `::fz_outline_iterator_item()`.
34482: (8)                  Call to get the current outline item.
34483: (8)                  Can return NULL. The item is only valid until the next call.
34484: (4)              """
34485: (4)              return _mupdf.fz_outline_iterator_item(iter)
34486: (0)          def fz_outline_iterator_next(iter):
34487: (4)              r"""
34488: (4)              Class-aware wrapper for `::fz_outline_iterator_next()`.
34489: (8)                  Calls to move the iterator position.
34490: (8)                  A negative return value means we could not move as requested.
Otherwise:
34491: (8)                  0 = the final position has a valid item.
34492: (8)                  1 = not a valid item, but we can insert an item here.
34493: (4)              """
34494: (4)              return _mupdf.fz_outline_iterator_next(iter)
34495: (0)          def fz_outline_iterator_prev(iter):
34496: (4)              r""" Class-aware wrapper for `::fz_outline_iterator_prev()`."""
34497: (4)              return _mupdf.fz_outline_iterator_prev(iter)
34498: (0)          def fz_outline_iterator_up(iter):
34499: (4)              r""" Class-aware wrapper for `::fz_outline_iterator_up()`."""
34500: (4)              return _mupdf.fz_outline_iterator_up(iter)
34501: (0)          def fz_output_accelerator(doc, accel):
34502: (4)              r"""
34503: (4)              Class-aware wrapper for `::fz_output_accelerator()`.
34504: (8)                  Output accelerator data for the document to a given output
34505: (8)                  stream.
34506: (4)              """
```

```
34507: (4)             return _mupdf.fz_output_accelerator(doc, accel)
34508: (0)         def fz_output_supports_stream(out):
34509: (4)             r"""
34510: (4)             Class-aware wrapper for `::fz_output_supports_stream()`.
34511: (8)                 Query whether a given fz_output supports fz_stream_from_output.
34512: (4)             """
34513: (4)             return _mupdf.fz_output_supports_stream(out)
34514: (0)         def fz_output_xml(out, item, level):
34515: (4)             r"""
34516: (4)             Class-aware wrapper for `::fz_output_xml()`.
34517: (8)                 Pretty-print an XML tree to given output.
34518: (4)             """
34519: (4)             return _mupdf.fz_output_xml(out, item, level)
34520: (0)         def fz_pack_path(pack, path):
34521: (4)             r"""
34522: (4)             Class-aware wrapper for `::fz_pack_path()`.
34523: (8)                 Pack a path into the given block.
34524: (8)                 To minimise the size of paths, this function allows them to be
34525: (8)                 packed into a buffer with other information. Paths can be used
34526: (8)                 interchangeably regardless of how they are packed.
34527: (8)                 pack: Pointer to a block of data to pack the path into. Should
34528: (8)                 be aligned by the caller to the same alignment as required for
34529: (8)                 a fz_path pointer.
34530: (8)                 path: The path to pack.
34531: (8)                 Returns the number of bytes within the block used. Callers can
34532: (8)                 access the packed path data by casting the value of pack on
34533: (8)                 entry to be a fz_path *.
34534: (8)                 Throws exceptions on failure to allocate.
34535: (8)                 Implementation details: Paths can be 'unpacked', 'flat', or
34536: (8)                 'open'. Standard paths, as created are 'unpacked'. Paths
34537: (8)                 will be packed as 'flat', unless they are too large
34538: (8)                 (where large indicates that they exceed some private
34539: (8)                 implementation defined limits, currently including having
34540: (8)                 more than 256 coordinates or commands).
34541: (8)                 Large paths are 'open' packed as a header into the given block,
34542: (8)                 plus pointers to other data blocks.
34543: (8)                 Users should not have to care about whether paths are 'open'
34544: (8)                 or 'flat' packed. Simply pack a path (if required), and then
34545: (8)                 forget about the details.
34546: (4)             """
34547: (4)             return _mupdf.fz_pack_path(pack, path)
34548: (0)         def fz_packed_path_size(path):
34549: (4)             r"""
34550: (4)             Class-aware wrapper for `::fz_packed_path_size()`.
34551: (8)                 Return the number of bytes required to pack a path.
34552: (4)             """
34553: (4)             return _mupdf.fz_packed_path_size(path)
34554: (0)         def fz_page_label(page, buf, size):
34555: (4)             r"""
34556: (4)             Class-aware wrapper for `::fz_page_label()`.
34557: (8)                 Get page label for a given page.
34558: (4)             """
34559: (4)             return _mupdf.fz_page_label(page, buf, size)
34560: (0)         def fz_page_number_from_location(doc, loc):
34561: (4)             r"""
34562: (4)             Class-aware wrapper for `::fz_page_number_from_location()`.
34563: (8)                 Converts from chapter+page to page number. This may cause many
34564: (8)                 chapters to be laid out in order to calculate the number of
34565: (8)                 pages within those chapters.
34566: (4)             """
34567: (4)             return _mupdf.fz_page_number_from_location(doc, loc)
34568: (0)         def fz_page_presentation(page, transition, duration):
34569: (4)             r"""
34570: (4)             Class-aware wrapper for `::fz_page_presentation()`.
34571: (4)             This function has out-params. Python/C# wrappers look like:
34572: (8)                 `fz_page_presentation(::fz_page *page, ::fz_transition *transition)`
=> `(fz_transition *, float duration)`
34573: (8)                 Get the presentation details for a given page.
34574: (8)                 transition: A pointer to a transition struct to fill out.
```

```
34575: (8)                  duration: A pointer to a place to set the page duration in
34576: (8)                  seconds. Will be set to 0 if no transition is specified for the
34577: (8)                  page.
34578: (8)                  Returns: a pointer to the transition structure, or NULL if there
34579: (8)                  is no transition specified for the page.
34580: (4)              """
34581: (4)              return _mupdf.fz_page_presentation(page, transition, duration)
34582: (0)          def fz_page_separations(page):
34583: (4)              r"""
34584: (4)              Class-aware wrapper for `::fz_page_separations()`.
34585: (8)                  Get the separations details for a page.
34586: (8)                  This will be NULL, unless the format specifically supports
34587: (8)                  separations (such as PDF files). May be NULL even
34588: (8)                  so, if there are no separations on a page.
34589: (8)                  Returns a reference that must be dropped.
34590: (4)              """
34591: (4)              return _mupdf.fz_page_separations(page)
34592: (0)          def fz_page_uses_overprint(page):
34593: (4)              r"""
34594: (4)              Class-aware wrapper for `::fz_page_uses_overprint()`.
34595: (8)                  Query if a given page requires overprint.
34596: (4)              """
34597: (4)              return _mupdf.fz_page_uses_overprint(page)
34598: (0)          def fz_paint_shade(shade, override_cs, ctm, dest, color_params, bbox, eop,
cache):
34599: (4)              r"""
34600: (4)              Class-aware wrapper for `::fz_paint_shade()`.
34601: (4)              This function has out-params. Python/C# wrappers look like:
34602: (8)                  `fz_paint_shade(::fz_shade *shade, ::fz_colorspace *override_cs,
::fz_matrix ctm, ::fz_pixmap *dest, ::fz_color_params color_params, ::fz_irect bbox, const
::fz_overprint *eop, ::fz_shade_color_cache **cache)` =>
34603: (8)                  Render a shade to a given pixmap.
34604: (8)                  shade: The shade to paint.
34605: (8)                  override_cs: NULL, or colorspace to override the shades
34606: (8)                  inbuilt colorspace.
34607: (8)                  ctm: The transform to apply.
34608: (8)                  dest: The pixmap to render into.
34609: (8)                  color_params: The color rendering settings
34610: (8)                  bbox: Pointer to a bounding box to limit the rendering
34611: (8)                  of the shade.
34612: (8)                  eop: NULL, or pointer to overprint bitmap.
34613: (8)                  cache: *cache is used to cache color information. If *cache is NULL it
34614: (8)                  is set to point to a new fz_shade_color_cache. If cache is NULL it is
34615: (8)                  ignored.
34616: (4)              """
34617: (4)              return _mupdf.fz_paint_shade(shade, override_cs, ctm, dest, color_params,
bbox, eop, cache)
34618: (0)          def fz_parse_page_range(s, a, b, n):
34619: (4)              r"""
34620: (4)              Class-aware wrapper for `::fz_parse_page_range()`.
34621: (4)              This function has out-params. Python/C# wrappers look like:
34622: (8)                  `fz_parse_page_range(const char *s, int n)` => `(const char *, int a,
int b)`
34623: (4)              """
34624: (4)              return _mupdf.fz_parse_page_range(s, a, b, n)
34625: (0)          def fz_parse_pdfocr_options(opts, args):
34626: (4)              r"""
34627: (4)              Class-aware wrapper for `::fz_parse_pdfocr_options()`.
34628: (8)                  Parse PDFOCR options.
34629: (8)                  Currently defined options and values are as follows:
34630: (12)                     compression=none: No compression
34631: (12)                     compression=flate: Flate compression
34632: (12)                     strip-height=n: Strip height (default 16)
34633: (12)                     ocr-language=<lang>: OCR Language (default eng)
34634: (12)                     ocr-datadir=<datadir>: OCR data path (default rely on
TESSDATA_PREFIX)
34635: (4)              """
34636: (4)              return _mupdf.fz_parse_pdfocr_options(opts, args)
34637: (0)          def fz_parse_stext_options(opts, string):
```

```
34638: (4)              r"""
34639: (4)              Class-aware wrapper for `::fz_parse_stext_options()`.
34640: (8)                  Parse stext device options from a comma separated key-value
34641: (8)                  string.
34642: (4)              """
34643: (4)              return _mupdf.fz_parse_stext_options(opts, string)
34644: (0)          def fz_parse_xml(buf, preserve_white):
34645: (4)              r"""
34646: (4)              Class-aware wrapper for `::fz_parse_xml()`.
34647: (8)                  Parse the contents of buffer into a tree of xml nodes.
34648: (8)                  preserve_white: whether to keep or delete all-whitespace nodes.
34649: (4)              """
34650: (4)              return _mupdf.fz_parse_xml(buf, preserve_white)
34651: (0)          def fz_parse_xml_archive_entry(dir, filename, preserve_white):
34652: (4)              r"""
34653: (4)              Class-aware wrapper for `::fz_parse_xml_archive_entry()`.
34654: (8)                  Parse the contents of an archive entry into a tree of xml nodes.
34655: (8)                  preserve_white: whether to keep or delete all-whitespace nodes.
34656: (4)              """
34657: (4)              return _mupdf.fz_parse_xml_archive_entry(dir, filename, preserve_white)
34658: (0)          def fz_parse_xml_from_html5(buf):
34659: (4)              r"""
34660: (4)              Class-aware wrapper for `::fz_parse_xml_from_html5()`.
34661: (8)                  Parse the contents of a buffer into a tree of XML nodes,
34662: (8)                  using the HTML5 parsing algorithm.
34663: (4)              """
34664: (4)              return _mupdf.fz_parse_xml_from_html5(buf)
34665: (0)          def fz_parse_xml_stream(stream, preserve_white):
34666: (4)              r"""
34667: (4)              Class-aware wrapper for `::fz_parse_xml_stream()`.
34668: (8)                  Parse the contents of buffer into a tree of xml nodes.
34669: (8)                  preserve_white: whether to keep or delete all-whitespace nodes.
34670: (4)              """
34671: (4)              return _mupdf.fz_parse_xml_stream(stream, preserve_white)
34672: (0)          def fz_pcl_preset(opts, preset):
34673: (4)              r"""
34674: (4)              Class-aware wrapper for `::fz_pcl_preset()`.
34675: (8)                  Initialize PCL option struct for a given preset.
34676: (8)                  Currently defined presets include:
34677: (12)                     generic    Generic PCL printer
34678: (12)                     ljet4    HP DeskJet
34679: (12)                     dj500    HP DeskJet 500
34680: (12)                     fs600    Kyocera FS-600
34681: (12)                     lj    HP LaserJet, HP LaserJet Plus
34682: (12)                     lj2    HP LaserJet IIp, HP LaserJet IId
34683: (12)                     lj3    HP LaserJet III
34684: (12)                     lj3d    HP LaserJet IIId
34685: (12)                     lj4    HP LaserJet 4
34686: (12)                     lj4pl    HP LaserJet 4 PL
34687: (12)                     lj4d    HP LaserJet 4d
34688: (12)                     lp2563b    HP 2563B line printer
34689: (12)                     oce9050    Oce 9050 Line printer
34690: (4)              """
34691: (4)              return _mupdf.fz_pcl_preset(opts, preset)
34692: (0)          def fz_pdfocr_band_writer_set_progress(writer, progress_fn, progress_arg):
34693: (4)              r"""
34694: (4)              Class-aware wrapper for `::fz_pdfocr_band_writer_set_progress()`.
34695: (8)                  Set the progress callback for a pdfocr bandwriter.
34696: (4)              """
34697: (4)              return _mupdf.fz_pdfocr_band_writer_set_progress(writer, progress_fn,
progress_arg)
34698: (0)          def fz_pdfocr_writer_set_progress(writer, progress, arg_2):
34699: (4)              r""" Class-aware wrapper for `::fz_pdfocr_writer_set_progress()`."""
34700: (4)              return _mupdf.fz_pdfocr_writer_set_progress(writer, progress, arg_2)
34701: (0)          def fz_peek_byte(stm):
34702: (4)              r"""
34703: (4)              Class-aware wrapper for `::fz_peek_byte()`.
34704: (8)                  Peek at the next byte in a stream.
34705: (8)                  stm: The stream to peek at.
```

```
34706: (8)                 Returns -1 for EOF, or the next byte that will be read.
34707: (4)             """
34708: (4)             return _mupdf.fz_peek_byte(stm)
34709: (0)         def fz_pixmap_alpha(pix):
34710: (4)             r"""
34711: (4)             Class-aware wrapper for `::fz_pixmap_alpha()`.
34712: (8)                 Return the number of alpha planes in a pixmap.
34713: (8)                 Returns the number of alphas. Does not throw exceptions.
34714: (4)             """
34715: (4)             return _mupdf.fz_pixmap_alpha(pix)
34716: (0)         def fz_pixmap_bbox(pix):
34717: (4)             r"""
34718: (4)             Class-aware wrapper for `::fz_pixmap_bbox()`.
34719: (8)                 Return the bounding box for a pixmap.
34720: (4)             """
34721: (4)             return _mupdf.fz_pixmap_bbox(pix)
34722: (0)         def fz_pixmap_colorants(pix):
34723: (4)             r"""
34724: (4)             Class-aware wrapper for `::fz_pixmap_colorants()`.
34725: (8)                 Return the number of colorants in a pixmap.
34726: (8)                 Returns the number of colorants (components, less any spots and
34727: (8)                 alpha).
34728: (4)             """
34729: (4)             return _mupdf.fz_pixmap_colorants(pix)
34730: (0)         def fz_pixmap_colorspace(pix):
34731: (4)             r"""
34732: (4)             Class-aware wrapper for `::fz_pixmap_colorspace()`.
34733: (8)                 Return the colorspace of a pixmap
34734: (8)                 Returns colorspace.
34735: (4)             """
34736: (4)             return _mupdf.fz_pixmap_colorspace(pix)
34737: (0)         def fz_pixmap_components(pix):
34738: (4)             r"""
34739: (4)             Class-aware wrapper for `::fz_pixmap_components()`.
34740: (8)                 Return the number of components in a pixmap.
34741: (8)                 Returns the number of components (including spots and alpha).
34742: (4)             """
34743: (4)             return _mupdf.fz_pixmap_components(pix)
34744: (0)         def fz_pixmap_height(pix):
34745: (4)             r"""
34746: (4)             Class-aware wrapper for `::fz_pixmap_height()`.
34747: (8)                 Return the height of the pixmap in pixels.
34748: (4)             """
34749: (4)             return _mupdf.fz_pixmap_height(pix)
34750: (0)         def fz_pixmap_image_tile(cimg):
34751: (4)             r"""
34752: (4)             Class-aware wrapper for `::fz_pixmap_image_tile()`.
34753: (8)                 Retrieve the underlying fz_pixmap for an image.
34754: (8)                 Returns a pointer to the underlying fz_pixmap for an image,
34755: (8)                 or NULL if this image is not based upon an fz_pixmap.
34756: (8)                 No reference is returned. Lifespan is limited to that of
34757: (8)                 the image itself. If required, use fz_keep_pixmap to take
34758: (8)                 a reference to keep it longer.
34759: (4)             """
34760: (4)             return _mupdf.fz_pixmap_image_tile(cimg)
34761: (0)         def fz_pixmap_samples(pix):
34762: (4)             r"""
34763: (4)             Class-aware wrapper for `::fz_pixmap_samples()`.
34764: (8)                 Returns a pointer to the pixel data of a pixmap.
34765: (8)                 Returns the pointer.
34766: (4)             """
34767: (4)             return _mupdf.fz_pixmap_samples(pix)
34768: (0)         def fz_pixmap_samples_int(pixmap):
34769: (4)             r""" Class-aware wrapper for `::fz_pixmap_samples_int()`."""
34770: (4)             return _mupdf.fz_pixmap_samples_int(pixmap)
34771: (0)         def fz_pixmap_size(pix):
34772: (4)             r"""
34773: (4)             Class-aware wrapper for `::fz_pixmap_size()`.
34774: (8)                 Return sizeof fz_pixmap plus size of data, in bytes.
```

```
34775: (4)             """
34776: (4)             return _mupdf.fz_pixmap_size(pix)
34777: (0)         def fz_pixmap_spots(pix):
34778: (4)             r"""
34779: (4)             Class-aware wrapper for `::fz_pixmap_spots()`.
34780: (8)                 Return the number of spots in a pixmap.
34781: (8)                 Returns the number of spots (components, less colorants and
34782: (8)                 alpha). Does not throw exceptions.
34783: (4)             """
34784: (4)             return _mupdf.fz_pixmap_spots(pix)
34785: (0)         def fz_pixmap_stride(pix):
34786: (4)             r"""
34787: (4)             Class-aware wrapper for `::fz_pixmap_stride()`.
34788: (8)                 Return the number of bytes in a row in the pixmap.
34789: (4)             """
34790: (4)             return _mupdf.fz_pixmap_stride(pix)
34791: (0)         def fz_pixmap_width(pix):
34792: (4)             r"""
34793: (4)             Class-aware wrapper for `::fz_pixmap_width()`.
34794: (8)                 Return the width of the pixmap in pixels.
34795: (4)             """
34796: (4)             return _mupdf.fz_pixmap_width(pix)
34797: (0)         def fz_pixmap_x(pix):
34798: (4)             r"""
34799: (4)             Class-aware wrapper for `::fz_pixmap_x()`.
34800: (8)                 Return the x value of the pixmap in pixels.
34801: (4)             """
34802: (4)             return _mupdf.fz_pixmap_x(pix)
34803: (0)         def fz_pixmap_y(pix):
34804: (4)             r"""
34805: (4)             Class-aware wrapper for `::fz_pixmap_y()`.
34806: (8)                 Return the y value of the pixmap in pixels.
34807: (4)             """
34808: (4)             return _mupdf.fz_pixmap_y(pix)
34809: (0)         def fz_place_story(story, where, filled):
34810: (4)             r""" Class-aware wrapper for `::fz_place_story()`."""
34811: (4)             return _mupdf.fz_place_story(story, where, filled)
34812: (0)         def fz_place_story_flags(story, where, filled, flags):
34813: (4)             r""" Class-aware wrapper for `::fz_place_story_flags()`."""
34814: (4)             return _mupdf.fz_place_story_flags(story, where, filled, flags)
34815: (0)         def fz_pool_alloc(pool, size):
34816: (4)             r"""
34817: (4)             Class-aware wrapper for `::fz_pool_alloc()`.
34818: (8)                 Allocate a block of size bytes from the pool.
34819: (4)             """
34820: (4)             return _mupdf.fz_pool_alloc(pool, size)
34821: (0)         def fz_pool_size(pool):
34822: (4)             r"""
34823: (4)             Class-aware wrapper for `::fz_pool_size()`.
34824: (8)                 The current size of the pool.
34825: (8)                 The number of bytes of storage currently allocated to the pool.
34826: (8)                 This is the total of the storage used for the blocks making
34827: (8)                 up the pool, rather then total of the allocated blocks so far,
34828: (8)                 so it will increase in 'lumps'.
34829: (8)                 from the pool, then the pool size may still be X
34830: (4)             """
34831: (4)             return _mupdf.fz_pool_size(pool)
34832: (0)         def fz_pool_strdup(pool, s):
34833: (4)             r"""
34834: (4)             Class-aware wrapper for `::fz_pool_strdup()`.
34835: (8)                 strdup equivalent allocating from the pool.
34836: (4)             """
34837: (4)             return _mupdf.fz_pool_strdup(pool, s)
34838: (0)         def fz_pop_clip(dev):
34839: (4)             r""" Class-aware wrapper for `::fz_pop_clip()`."""
34840: (4)             return _mupdf.fz_pop_clip(dev)
34841: (0)         def fz_post_scale(m, sx, sy):
34842: (4)             r"""
34843: (4)             Class-aware wrapper for `::fz_post_scale()`.
```

```
(8)                 Scale a matrix by postmultiplication.
(8)                 m: Pointer to the matrix to scale
(8)                 sx, sy: Scaling factors along the X- and Y-axes. A scaling
(8)                 factor of 1.0 will not cause any scaling along the relevant
(8)                 axis.
(8)                 Returns m (updated).
(4)             """
(4)             return _mupdf.fz_post_scale(m, sx, sy)
(0)         def fz_pre_rotate(m, degrees):
(4)             r"""
(4)             Class-aware wrapper for `::fz_pre_rotate()`.
(8)                 Rotate a transformation by premultiplying.
(8)                 The premultiplied matrix is of the form
(8)                 [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
(8)                 m: Pointer to matrix to premultiply.
(8)                 degrees: Degrees of counter clockwise rotation. Values less
(8)                 than zero and greater than 360 are handled as expected.
(8)                 Returns m (updated).
(4)             """
(4)             return _mupdf.fz_pre_rotate(m, degrees)
(0)         def fz_pre_scale(m, sx, sy):
(4)             r"""
(4)             Class-aware wrapper for `::fz_pre_scale()`.
(8)                 Scale a matrix by premultiplication.
(8)                 m: Pointer to the matrix to scale
(8)                 sx, sy: Scaling factors along the X- and Y-axes. A scaling
(8)                 factor of 1.0 will not cause any scaling along the relevant
(8)                 axis.
(8)                 Returns m (updated).
(4)             """
(4)             return _mupdf.fz_pre_scale(m, sx, sy)
(0)         def fz_pre_shear(m, sx, sy):
(4)             r"""
(4)             Class-aware wrapper for `::fz_pre_shear()`.
(8)                 Premultiply a matrix with a shearing matrix.
(8)                 The shearing matrix is of the form [ 1 sy sx 1 0 0 ].
(8)                 m: pointer to matrix to premultiply
(8)                 sx, sy: Shearing factors. A shearing factor of 0.0 will not
(8)                 cause any shearing along the relevant axis.
(8)                 Returns m (updated).
(4)             """
(4)             return _mupdf.fz_pre_shear(m, sx, sy)
(0)         def fz_pre_translate(m, tx, ty):
(4)             r"""
(4)             Class-aware wrapper for `::fz_pre_translate()`.
(8)                 Translate a matrix by premultiplication.
(8)                 m: The matrix to translate
(8)                 tx, ty: Translation distances along the X- and Y-axes. A
(8)                 translation of 0 will not cause any translation along the
(8)                 relevant axis.
(8)                 Returns m.
(4)             """
(4)             return _mupdf.fz_pre_translate(m, tx, ty)
(0)         def fz_prepare_t3_glyph(font, gid):
(4)             r"""
(4)             Class-aware wrapper for `::fz_prepare_t3_glyph()`.
(8)                 Force a type3 font to cache the displaylist for a given glyph
(8)                 id.
(8)                 This caching can involve reading the underlying file, so must
(8)                 happen ahead of time, so we aren't suddenly forced to read the
(8)                 file while playing a displaylist back.
(4)             """
(4)             return _mupdf.fz_prepare_t3_glyph(font, gid)
(0)         def fz_previous_page(doc, loc):
(4)             r"""
(4)             Class-aware wrapper for `::fz_previous_page()`.
(8)                 Function to get the location of the previous page (allowing for
(8)                 the end of chapters etc). If already at the start of the
(8)                 document, returns the current page.
```

```
(4)             """
(4)             return _mupdf.fz_previous_page(doc, loc)
(0)         def fz_print_stext_header_as_html(out):
(4)             r""" Class-aware wrapper for `::fz_print_stext_header_as_html()`."""
(4)             return _mupdf.fz_print_stext_header_as_html(out)
(0)         def fz_print_stext_header_as_xhtml(out):
(4)             r""" Class-aware wrapper for `::fz_print_stext_header_as_xhtml()`."""
(4)             return _mupdf.fz_print_stext_header_as_xhtml(out)
(0)         def fz_print_stext_page_as_html(out, page, id):
(4)             r"""
(4)             Class-aware wrapper for `::fz_print_stext_page_as_html()`.
(8)                 Output structured text to a file in HTML (visual) format.
(4)             """
(4)             return _mupdf.fz_print_stext_page_as_html(out, page, id)
(0)         def fz_print_stext_page_as_json(out, page, scale):
(4)             r"""
(4)             Class-aware wrapper for `::fz_print_stext_page_as_json()`.
(8)                 Output structured text to a file in JSON format.
(4)             """
(4)             return _mupdf.fz_print_stext_page_as_json(out, page, scale)
(0)         def fz_print_stext_page_as_text(out, page):
(4)             r"""
(4)             Class-aware wrapper for `::fz_print_stext_page_as_text()`.
(8)                 Output structured text to a file in plain-text UTF-8 format.
(4)             """
(4)             return _mupdf.fz_print_stext_page_as_text(out, page)
(0)         def fz_print_stext_page_as_xhtml(out, page, id):
(4)             r"""
(4)             Class-aware wrapper for `::fz_print_stext_page_as_xhtml()`.
(8)                 Output structured text to a file in XHTML (semantic) format.
(4)             """
(4)             return _mupdf.fz_print_stext_page_as_xhtml(out, page, id)
(0)         def fz_print_stext_page_as_xml(out, page, id):
(4)             r"""
(4)             Class-aware wrapper for `::fz_print_stext_page_as_xml()`.
(8)                 Output structured text to a file in XML format.
(4)             """
(4)             return _mupdf.fz_print_stext_page_as_xml(out, page, id)
(0)         def fz_print_stext_trailer_as_html(out):
(4)             r""" Class-aware wrapper for `::fz_print_stext_trailer_as_html()`."""
(4)             return _mupdf.fz_print_stext_trailer_as_html(out)
(0)         def fz_print_stext_trailer_as_xhtml(out):
(4)             r""" Class-aware wrapper for `::fz_print_stext_trailer_as_xhtml()`."""
(4)             return _mupdf.fz_print_stext_trailer_as_xhtml(out)
(0)         def fz_process_opened_pages(doc, process_openend_page, state):
(4)             r"""
(4)             Class-aware wrapper for `::fz_process_opened_pages()`.
(8)                 Iterates over all opened pages of the document, calling the
(8)                 provided callback for each page for processing. If the callback
(8)                 returns non-NULL then the iteration stops and that value is returned
(8)                 to the called of fz_process_opened_pages().
(8)                 The state pointer provided to fz_process_opened_pages() is
(8)                 passed on to the callback but is owned by the caller.
(8)                 Returns the first non-NULL value returned by the callback,
(8)                 or NULL if the callback returned NULL for all opened pages.
(4)             """
(4)             return _mupdf.fz_process_opened_pages(doc, process_openend_page, state)
(0)         def fz_process_shade(shade, ctm, scissor, prepare, process, process_arg):
(4)             r"""
(4)             Class-aware wrapper for `::fz_process_shade()`.
(8)                 Process a shade, using supplied callback functions. This
(8)                 decomposes the shading to a mesh (even ones that are not
(8)                 natively meshes, such as linear or radial shadings), and
(8)                 processes triangles from those meshes.
(8)                 shade: The shade to process.
(8)                 ctm: The transform to use
(8)                 prepare: Callback function to 'prepare' each vertex.
(8)                 This function is passed an array of floats, and populates
(8)                 a fz_vertex structure.
```

```
34982: (8)                 process: This function is passed 3 pointers to vertex
34983: (8)                 structures, and actually performs the processing (typically
34984: (8)                 filling the area between the vertexes).
34985: (8)                 process_arg: An opaque argument passed through from caller
34986: (8)                 to callback functions.
34987: (4)             """
34988: (4)             return _mupdf.fz_process_shade(shade, ctm, scissor, prepare, process,
process_arg)
34989: (0)         def fz_ptr_heap_insert(heap, v, HEAP_CMP):
34990: (4)             r""" Class-aware wrapper for `::fz_ptr_heap_insert()`."""
34991: (4)             return _mupdf.fz_ptr_heap_insert(heap, v, HEAP_CMP)
34992: (0)         def fz_ptr_heap_sort(heap, HEAP_CMP):
34993: (4)             r""" Class-aware wrapper for `::fz_ptr_heap_sort()`."""
34994: (4)             return _mupdf.fz_ptr_heap_sort(heap, HEAP_CMP)
34995: (0)         def fz_ptr_heap_uniq(heap, HEAP_CMP):
34996: (4)             r""" Class-aware wrapper for `::fz_ptr_heap_uniq()`."""
34997: (4)             return _mupdf.fz_ptr_heap_uniq(heap, HEAP_CMP)
34998: (0)         def fz_purge_glyph_cache():
34999: (4)             r"""
35000: (4)             Class-aware wrapper for `::fz_purge_glyph_cache()`.
35001: (8)                 Purge all the glyphs from the cache.
35002: (4)             """
35003: (4)             return _mupdf.fz_purge_glyph_cache()
35004: (0)         def fz_quad_from_rect(r):
35005: (4)             r"""
35006: (4)             Class-aware wrapper for `::fz_quad_from_rect()`.
35007: (8)                 Convert a rect to a quad (losslessly).
35008: (4)             """
35009: (4)             return _mupdf.fz_quad_from_rect(r)
35010: (0)         def fz_quadto(path, x0, y0, x1, y1):
35011: (4)             r"""
35012: (4)             Class-aware wrapper for `::fz_quadto()`.
35013: (8)                 Append a 'quadto' command to an open path. (For a
35014: (8)                 quadratic bezier).
35015: (8)                 path: The path to modify.
35016: (8)                 x0, y0: The control coordinates for the quadratic curve.
35017: (8)                 x1, y1: The end coordinates for the quadratic curve.
35018: (8)                 Throws exceptions on failure to allocate, or attempting to
35019: (8)                 modify a packed path.
35020: (4)             """
35021: (4)             return _mupdf.fz_quadto(path, x0, y0, x1, y1)
35022: (0)         def fz_read(stm, data, len):
35023: (4)             r"""
35024: (4)             Class-aware wrapper for `::fz_read()`.
35025: (8)                 Read from a stream into a given data block.
35026: (8)                 stm: The stream to read from.
35027: (8)                 data: The data block to read into.
35028: (8)                 len: The length of the data block (in bytes).
35029: (8)                 Returns the number of bytes read. May throw exceptions.
35030: (4)             """
35031: (4)             return _mupdf.fz_read(stm, data, len)
35032: (0)         def fz_read_all(stm, initial):
35033: (4)             r"""
35034: (4)             Class-aware wrapper for `::fz_read_all()`.
35035: (8)                 Read all of a stream into a buffer.
35036: (8)                 stm: The stream to read from
35037: (8)                 initial: Suggested initial size for the buffer.
35038: (8)                 Returns a buffer created from reading from the stream. May throw
35039: (8)                 exceptions on failure to allocate.
35040: (4)             """
35041: (4)             return _mupdf.fz_read_all(stm, initial)
35042: (0)         def fz_read_archive_entry(arch, name):
35043: (4)             r"""
35044: (4)             Class-aware wrapper for `::fz_read_archive_entry()`.
35045: (8)                 Reads all bytes in an archive entry
35046: (8)                 into a buffer.
35047: (8)                 name: Entry name to look for, this must be an exact match to
35048: (8)                 the entry name in the archive.
35049: (8)                 Throws an exception if a matching entry cannot be found.
```

```
35050: (4)              """
35051: (4)              return _mupdf.fz_read_archive_entry(arch, name)
35052: (0)          def fz_read_best(stm, initial, truncated, worst_case):
35053: (4)              r"""
35054: (4)              Class-aware wrapper for `::fz_read_best()`.
35055: (4)              This function has out-params. Python/C# wrappers look like:
35056: (8)                  `fz_read_best(::fz_stream *stm, size_t initial, size_t worst_case)` =>
`(fz_buffer *, int truncated)`
35057: (8)                  Attempt to read a stream into a buffer. If truncated
35058: (8)                  is NULL behaves as fz_read_all, sets a truncated flag in case of
35059: (8)                  error.
35060: (8)                  stm: The stream to read from.
35061: (8)                  initial: Suggested initial size for the buffer.
35062: (8)                  truncated: Flag to store success/failure indication in.
35063: (8)                  worst_case: 0 for unknown, otherwise an upper bound for the
35064: (8)                  size of the stream.
35065: (8)                  Returns a buffer created from reading from the stream.
35066: (4)              """
35067: (4)              return _mupdf.fz_read_best(stm, initial, truncated, worst_case)
35068: (0)          def fz_read_bits(stm, n):
35069: (4)              r"""
35070: (4)              Class-aware wrapper for `::fz_read_bits()`.
35071: (8)                  Read the next n bits from a stream (assumed to
35072: (8)                  be packed most significant bit first).
35073: (8)                  stm: The stream to read from.
35074: (8)                  n: The number of bits to read, between 1 and 8*sizeof(int)
35075: (8)                  inclusive.
35076: (8)                  Returns -1 for EOF, or the required number of bits.
35077: (4)              """
35078: (4)              return _mupdf.fz_read_bits(stm, n)
35079: (0)          def fz_read_byte(stm):
35080: (4)              r"""
35081: (4)              Class-aware wrapper for `::fz_read_byte()`.
35082: (8)                  Read the next byte from a stream.
35083: (8)                  stm: The stream t read from.
35084: (8)                  Returns -1 for end of stream, or the next byte. May
35085: (8)                  throw exceptions.
35086: (4)              """
35087: (4)              return _mupdf.fz_read_byte(stm)
35088: (0)          def fz_read_file(filename):
35089: (4)              r"""
35090: (4)              Class-aware wrapper for `::fz_read_file()`.
35091: (8)                  Read all the contents of a file into a buffer.
35092: (4)              """
35093: (4)              return _mupdf.fz_read_file(filename)
35094: (0)          def fz_read_float(stm):
35095: (4)              r""" Class-aware wrapper for `::fz_read_float()`."""
35096: (4)              return _mupdf.fz_read_float(stm)
35097: (0)          def fz_read_float_le(stm):
35098: (4)              r""" Class-aware wrapper for `::fz_read_float_le()`."""
35099: (4)              return _mupdf.fz_read_float_le(stm)
35100: (0)          def fz_read_int16(stm):
35101: (4)              r""" Class-aware wrapper for `::fz_read_int16()`."""
35102: (4)              return _mupdf.fz_read_int16(stm)
35103: (0)          def fz_read_int16_le(stm):
35104: (4)              r""" Class-aware wrapper for `::fz_read_int16_le()`."""
35105: (4)              return _mupdf.fz_read_int16_le(stm)
35106: (0)          def fz_read_int32(stm):
35107: (4)              r""" Class-aware wrapper for `::fz_read_int32()`."""
35108: (4)              return _mupdf.fz_read_int32(stm)
35109: (0)          def fz_read_int32_le(stm):
35110: (4)              r""" Class-aware wrapper for `::fz_read_int32_le()`."""
35111: (4)              return _mupdf.fz_read_int32_le(stm)
35112: (0)          def fz_read_int64(stm):
35113: (4)              r""" Class-aware wrapper for `::fz_read_int64()`."""
35114: (4)              return _mupdf.fz_read_int64(stm)
35115: (0)          def fz_read_int64_le(stm):
35116: (4)              r""" Class-aware wrapper for `::fz_read_int64_le()`."""
35117: (4)              return _mupdf.fz_read_int64_le(stm)
```

```
35118: (0)          def fz_read_line(stm, buf, max):
35119: (4)              r"""
35120: (4)              Class-aware wrapper for `::fz_read_line()`.
35121: (8)                  Read a line from stream into the buffer until either a
35122: (8)                  terminating newline or EOF, which it replaces with a null byte
35123: (8)                  ('').
35124: (8)                  Returns buf on success, and NULL when end of file occurs while
35125: (8)                  no characters have been read.
35126: (4)              """
35127: (4)              return _mupdf.fz_read_line(stm, buf, max)
35128: (0)          def fz_read_rbits(stm, n):
35129: (4)              r"""
35130: (4)              Class-aware wrapper for `::fz_read_rbits()`.
35131: (8)                  Read the next n bits from a stream (assumed to
35132: (8)                  be packed least significant bit first).
35133: (8)                  stm: The stream to read from.
35134: (8)                  n: The number of bits to read, between 1 and 8*sizeof(int)
35135: (8)                  inclusive.
35136: (8)                  Returns (unsigned int)-1 for EOF, or the required number of bits.
35137: (4)              """
35138: (4)              return _mupdf.fz_read_rbits(stm, n)
35139: (0)          def fz_read_rune(_in):
35140: (4)              r"""
35141: (4)              Class-aware wrapper for `::fz_read_rune()`.
35142: (8)                  Read a utf-8 rune from a stream.
35143: (8)                  In the event of encountering badly formatted utf-8 codes
35144: (8)                  (such as a leading code with an unexpected number of following
35145: (8)                  codes) no error/exception is given, but undefined values may be
35146: (8)                  returned.
35147: (4)              """
35148: (4)              return _mupdf.fz_read_rune(_in)
35149: (0)          def fz_read_string(stm, buffer, len):
35150: (4)              r"""
35151: (4)              Class-aware wrapper for `::fz_read_string()`.
35152: (8)                  Read a null terminated string from the stream into
35153: (8)                  a buffer of a given length. The buffer will be null terminated.
35154: (8)                  Throws on failure (including the failure to fit the entire
35155: (8)                  string including the terminator into the buffer).
35156: (4)              """
35157: (4)              return _mupdf.fz_read_string(stm, buffer, len)
35158: (0)          def fz_read_uint16(stm):
35159: (4)              r"""
35160: (4)              Class-aware wrapper for `::fz_read_uint16()`.
35161: (8)                  fz_read_[u]int(16|24|32|64)(_le)?
35162: (8)                  Read a 16/32/64 bit signed/unsigned integer from stream,
35163: (8)                  in big or little-endian byte orders.
35164: (8)                  Throws an exception if EOF is encountered.
35165: (4)              """
35166: (4)              return _mupdf.fz_read_uint16(stm)
35167: (0)          def fz_read_uint16_le(stm):
35168: (4)              r""" Class-aware wrapper for `::fz_read_uint16_le()`."""
35169: (4)              return _mupdf.fz_read_uint16_le(stm)
35170: (0)          def fz_read_uint24(stm):
35171: (4)              r""" Class-aware wrapper for `::fz_read_uint24()`."""
35172: (4)              return _mupdf.fz_read_uint24(stm)
35173: (0)          def fz_read_uint24_le(stm):
35174: (4)              r""" Class-aware wrapper for `::fz_read_uint24_le()`."""
35175: (4)              return _mupdf.fz_read_uint24_le(stm)
35176: (0)          def fz_read_uint32(stm):
35177: (4)              r""" Class-aware wrapper for `::fz_read_uint32()`."""
35178: (4)              return _mupdf.fz_read_uint32(stm)
35179: (0)          def fz_read_uint32_le(stm):
35180: (4)              r""" Class-aware wrapper for `::fz_read_uint32_le()`."""
35181: (4)              return _mupdf.fz_read_uint32_le(stm)
35182: (0)          def fz_read_uint64(stm):
35183: (4)              r""" Class-aware wrapper for `::fz_read_uint64()`."""
35184: (4)              return _mupdf.fz_read_uint64(stm)
35185: (0)          def fz_read_uint64_le(stm):
35186: (4)              r""" Class-aware wrapper for `::fz_read_uint64_le()`."""
```

```
35187: (4)              return _mupdf.fz_read_uint64_le(stm)
35188: (0)          def fz_read_utf16_be(stm):
35189: (4)              r""" Class-aware wrapper for `::fz_read_utf16_be()`."""
35190: (4)              return _mupdf.fz_read_utf16_be(stm)
35191: (0)          def fz_read_utf16_le(stm):
35192: (4)              r"""
35193: (4)              Class-aware wrapper for `::fz_read_utf16_le()`.
35194: (8)                  Read a utf-16 rune from a stream. (little endian and
35195: (8)                  big endian respectively).
35196: (8)                  In the event of encountering badly formatted utf-16 codes
35197: (8)                  (mismatched surrogates) no error/exception is given, but
35198: (8)                  undefined values may be returned.
35199: (4)              """
35200: (4)              return _mupdf.fz_read_utf16_le(stm)
35201: (0)          def fz_realloc(p, size):
35202: (4)              r"""
35203: (4)              Class-aware wrapper for `::fz_realloc()`.
35204: (8)                  Reallocates a block of memory to given size. Existing contents
35205: (8)                  up to min(old_size,new_size) are maintained. The rest of the
35206: (8)                  block is uninitialised.
35207: (8)                  fz_realloc(ctx, NULL, size) behaves like fz_malloc(ctx, size).
35208: (8)                  fz_realloc(ctx, p, 0); behaves like fz_free(ctx, p).
35209: (8)                  Throws exception in the event of failure to allocate.
35210: (4)              """
35211: (4)              return _mupdf.fz_realloc(p, size)
35212: (0)          def fz_realloc_no_throw(p, size):
35213: (4)              r"""
35214: (4)              Class-aware wrapper for `::fz_realloc_no_throw()`.
35215: (8)                  fz_realloc equivalent that returns NULL rather than throwing
35216: (8)                  exceptions.
35217: (4)              """
35218: (4)              return _mupdf.fz_realloc_no_throw(p, size)
35219: (0)          def fz_realpath(path, resolved_path):
35220: (4)              r"""
35221: (4)              Class-aware wrapper for `::fz_realpath()`.
35222: (8)                  Resolve a path to an absolute file name.
35223: (8)                  The resolved path buffer must be of at least PATH_MAX size.
35224: (4)              """
35225: (4)              return _mupdf.fz_realpath(path, resolved_path)
35226: (0)          def fz_recognize_image_format(p):
35227: (4)              r"""
35228: (4)              Class-aware wrapper for `::fz_recognize_image_format()`.
35229: (8)                  Recognise image format strings in the first 8 bytes from image
35230: (8)                  data.
35231: (4)              """
35232: (4)              return _mupdf.fz_recognize_image_format(p)
35233: (0)          def fz_rect_from_irect(bbox):
35234: (4)              r"""
35235: (4)              Class-aware wrapper for `::fz_rect_from_irect()`.
35236: (8)                  Convert a bbox into a rect.
35237: (8)                  For our purposes, a rect can represent all the values we meet in
35238: (8)                  a bbox, so nothing can go wrong.
35239: (8)                  rect: A place to store the generated rectangle.
35240: (8)                  bbox: The bbox to convert.
35241: (8)                  Returns rect (updated).
35242: (4)              """
35243: (4)              return _mupdf.fz_rect_from_irect(bbox)
35244: (0)          def fz_rect_from_quad(q):
35245: (4)              r"""
35246: (4)              Class-aware wrapper for `::fz_rect_from_quad()`.
35247: (8)                  Convert a quad to the smallest rect that covers it.
35248: (4)              """
35249: (4)              return _mupdf.fz_rect_from_quad(q)
35250: (0)          def fz_rectto(path, x0, y0, x1, y1):
35251: (4)              r"""
35252: (4)              Class-aware wrapper for `::fz_rectto()`.
35253: (8)                  Append a 'rectto' command to an open path.
35254: (8)                  The rectangle is equivalent to:
35255: (12)                     moveto x0 y0
```

```
35256: (12)                     lineto x1 y0
35257: (12)                     lineto x1 y1
35258: (12)                     lineto x0 y1
35259: (12)                     closepath
35260: (8)                  path: The path to modify.
35261: (8)                  x0, y0: First corner of the rectangle.
35262: (8)                  x1, y1: Second corner of the rectangle.
35263: (8)                  Throws exceptions on failure to allocate, or attempting to
35264: (8)                  modify a packed path.
35265: (4)              """
35266: (4)              return _mupdf.fz_rectto(path, x0, y0, x1, y1)
35267: (0)          def fz_register_archive_handler(handler):
35268: (4)              r""" Class-aware wrapper for `::fz_register_archive_handler()`."""
35269: (4)              return _mupdf.fz_register_archive_handler(handler)
35270: (0)          def fz_register_document_handler(handler):
35271: (4)              r"""
35272: (4)              Class-aware wrapper for `::fz_register_document_handler()`.
35273: (8)                  Register a handler for a document type.
35274: (8)                  handler: The handler to register. This must live on for the duration
of the
35275: (8)                  use of this handler. It will be passed back to the handler for calls
so
35276: (8)                  the caller can use it to retrieve state.
35277: (4)              """
35278: (4)              return _mupdf.fz_register_document_handler(handler)
35279: (0)          def fz_register_document_handlers():
35280: (4)              r"""
35281: (4)              Class-aware wrapper for `::fz_register_document_handlers()`.
35282: (8)                  Register handlers for all the standard document types supported in
35283: (8)                  this build.
35284: (4)              """
35285: (4)              return _mupdf.fz_register_document_handlers()
35286: (0)          def fz_remove_item(drop, key, type):
35287: (4)              r"""
35288: (4)              Class-aware wrapper for `::fz_remove_item()`.
35289: (8)                  Remove an item from the store.
35290: (8)                  If an item indexed by the given key exists in the store, remove
35291: (8)                  it.
35292: (8)                  drop: The function used to free the value (to ensure we get a
35293: (8)                  value of the correct type).
35294: (8)                  key: The key used to find the item to remove.
35295: (8)                  type: Functions used to manipulate the key.
35296: (4)              """
35297: (4)              return _mupdf.fz_remove_item(drop, key, type)
35298: (0)          def fz_render_flags(dev, set, clear):
35299: (4)              r""" Class-aware wrapper for `::fz_render_flags()`."""
35300: (4)              return _mupdf.fz_render_flags(dev, set, clear)
35301: (0)          def fz_render_glyph_pixmap(font, gid, ctm, scissor, aa):
35302: (4)              r"""
35303: (4)              Class-aware wrapper for `::fz_render_glyph_pixmap()`.
35304: (8)                  Create a pixmap containing a rendered glyph.
35305: (8)                  Lookup gid from font, clip it with scissor, and rendering it
35306: (8)                  with aa bits of antialiasing into a new pixmap.
35307: (8)                  The caller takes ownership of the pixmap and so must free it.
35308: (8)                  Note: This function is no longer used for normal rendering
35309: (8)                  operations, and is kept around just because we use it in the
35310: (8)                  app. It should be considered "at risk" of removal from the API.
35311: (4)              """
35312: (4)              return _mupdf.fz_render_glyph_pixmap(font, gid, ctm, scissor, aa)
35313: (0)          def fz_render_t3_glyph_direct(dev, font, gid, trm, gstate, def_cs):
35314: (4)              r"""
35315: (4)              Class-aware wrapper for `::fz_render_t3_glyph_direct()`.
35316: (8)                  Nasty PDF interpreter specific hernia, required to allow the
35317: (8)                  interpreter to replay glyphs from a type3 font directly into
35318: (8)                  the target device.
35319: (8)                  This is only used in exceptional circumstances (such as type3
35320: (8)                  glyphs that inherit current graphics state, or nested type3
35321: (8)                  glyphs).
35322: (4)              """
```

```
35323: (4)             return _mupdf.fz_render_t3_glyph_direct(dev, font, gid, trm, gstate,
def_cs)
35324: (0)         def fz_rendering_intent_name(ri):
35325: (4)             r"""
35326: (4)             Class-aware wrapper for `::fz_rendering_intent_name()`.
35327: (8)                 Map from enumerated rendering intent to string.
35328: (8)                 The returned string is static and therefore must not be freed.
35329: (4)             """
35330: (4)             return _mupdf.fz_rendering_intent_name(ri)
35331: (0)         def fz_report_error():
35332: (4)             r""" Class-aware wrapper for `::fz_report_error()`."""
35333: (4)             return _mupdf.fz_report_error()
35334: (0)         def fz_reset_output(arg_0):
35335: (4)             r"""
35336: (4)             Class-aware wrapper for `::fz_reset_output()`.
35337: (8)                 Reset a closed output stream. Returns state to
35338: (8)                 (broadly) that which it was in when opened. Not
35339: (8)                 all outputs can be reset, so this may throw an
35340: (8)                 exception.
35341: (4)             """
35342: (4)             return _mupdf.fz_reset_output(arg_0)
35343: (0)         def fz_reset_story(story):
35344: (4)             r""" Class-aware wrapper for `::fz_reset_story()`."""
35345: (4)             return _mupdf.fz_reset_story(story)
35346: (0)         def fz_resize_buffer(buf, capacity):
35347: (4)             r"""
35348: (4)             Class-aware wrapper for `::fz_resize_buffer()`.
35349: (8)                 Ensure that a buffer has a given capacity,
35350: (8)                 truncating data if required.
35351: (8)                 capacity: The desired capacity for the buffer. If the current
35352: (8)                 size of the buffer contents is smaller than capacity, it is
35353: (8)                 truncated.
35354: (4)             """
35355: (4)             return _mupdf.fz_resize_buffer(buf, capacity)
35356: (0)         def fz_resolve_link(doc, uri, xp, yp):
35357: (4)             r"""
35358: (4)             Class-aware wrapper for `::fz_resolve_link()`.
35359: (4)             This function has out-params. Python/C# wrappers look like:
35360: (8)                 `fz_resolve_link(::fz_document *doc, const char *uri)` =>
`(fz_location, float xp, float yp)`
35361: (8)                 Resolve an internal link to a page number.
35362: (8)                 xp, yp: Pointer to store coordinate of destination on the page.
35363: (8)                 Returns (-1,-1) if the URI cannot be resolved.
35364: (4)             """
35365: (4)             return _mupdf.fz_resolve_link(doc, uri, xp, yp)
35366: (0)         def fz_rethrow():
35367: (4)             r""" Class-aware wrapper for `::fz_rethrow()`."""
35368: (4)             return _mupdf.fz_rethrow()
35369: (0)         def fz_rethrow_if(errcode):
35370: (4)             r"""
35371: (4)             Class-aware wrapper for `::fz_rethrow_if()`.
35372: (8)                 Within an fz_catch() block, rethrow the current exception
35373: (8)                 if the errcode of the current exception matches.
35374: (8)                 This assumes no intervening use of fz_try/fz_catch.
35375: (4)             """
35376: (4)             return _mupdf.fz_rethrow_if(errcode)
35377: (0)         def fz_rethrow_unless(errcode):
35378: (4)             r""" Class-aware wrapper for `::fz_rethrow_unless()`."""
35379: (4)             return _mupdf.fz_rethrow_unless(errcode)
35380: (0)         def fz_rotate(degrees):
35381: (4)             r"""
35382: (4)             Class-aware wrapper for `::fz_rotate()`.
35383: (8)                 Create a rotation matrix.
35384: (8)                 The returned matrix is of the form
35385: (8)                 [ cos(deg) sin(deg) -sin(deg) cos(deg) 0 0 ].
35386: (8)                 m: Pointer to place to store matrix
35387: (8)                 degrees: Degrees of counter clockwise rotation. Values less
35388: (8)                 than zero and greater than 360 are handled as expected.
35389: (8)                 Returns m.
```

```
35390: (4)             """
35391: (4)             return _mupdf.fz_rotate(degrees)
35392: (0)         def fz_round_rect(rect):
35393: (4)             r"""
35394: (4)             Class-aware wrapper for `::fz_round_rect()`.
35395: (8)                 Round rectangle coordinates.
35396: (8)                 Coordinates in a bounding box are integers, so rounding of the
35397: (8)                 rects coordinates takes place. The top left corner is rounded
35398: (8)                 upwards and left while the bottom right corner is rounded
35399: (8)                 downwards and to the right.
35400: (8)                 This differs from fz_irect_from_rect, in that fz_irect_from_rect
35401: (8)                 slavishly follows the numbers (i.e any slight over/under
35402: (8)                 calculations can cause whole extra pixels to be added).
35403: (8)                 fz_round_rect allows for a small amount of rounding error when
35404: (8)                 calculating the bbox.
35405: (4)             """
35406: (4)             return _mupdf.fz_round_rect(rect)
35407: (0)         def fz_run_display_list(list, dev, ctm, scissor, cookie):
35408: (4)             r"""
35409: (4)             Class-aware wrapper for `::fz_run_display_list()`.
35410: (8)                 (Re)-run a display list through a device.
35411: (8)                 list: A display list, created by fz_new_display_list and
35412: (8)                 populated with objects from a page by running fz_run_page on a
35413: (8)                 device obtained from fz_new_list_device.
35414: (8)                 ctm: Transform to apply to display list contents. May include
35415: (8)                 for example scaling and rotation, see fz_scale, fz_rotate and
35416: (8)                 fz_concat. Set to fz_identity if no transformation is desired.
35417: (8)                 scissor: Only the part of the contents of the display list
35418: (8)                 visible within this area will be considered when the list is
35419: (8)                 run through the device. This does not imply for tile objects
35420: (8)                 contained in the display list.
35421: (8)                 cookie: Communication mechanism between caller and library
35422: (8)                 running the page. Intended for multi-threaded applications,
35423: (8)                 while single-threaded applications set cookie to NULL. The
35424: (8)                 caller may abort an ongoing page run. Cookie also communicates
35425: (8)                 progress information back to the caller. The fields inside
35426: (8)                 cookie are continually updated while the page is being run.
35427: (4)             """
35428: (4)             return _mupdf.fz_run_display_list(list, dev, ctm, scissor, cookie)
35429: (0)         def fz_run_document_structure(doc, dev, cookie):
35430: (4)             r"""
35431: (4)             Class-aware wrapper for `::fz_run_document_structure()`.
35432: (8)                 Run the document structure through a device.
35433: (8)                 doc: Document in question.
35434: (8)                 dev: Device obtained from fz_new_*_device.
35435: (8)                 cookie: Communication mechanism between caller and library.
35436: (8)                 Intended for multi-threaded applications, while
35437: (8)                 single-threaded applications set cookie to NULL. The
35438: (8)                 caller may abort an ongoing rendering of a page. Cookie also
35439: (8)                 communicates progress information back to the caller. The
35440: (8)                 fields inside cookie are continually updated while the page is
35441: (8)                 rendering.
35442: (4)             """
35443: (4)             return _mupdf.fz_run_document_structure(doc, dev, cookie)
35444: (0)         def fz_run_page(page, dev, transform, cookie):
35445: (4)             r"""
35446: (4)             Class-aware wrapper for `::fz_run_page()`.
35447: (8)                 Run a page through a device.
35448: (8)                 page: Page obtained from fz_load_page.
35449: (8)                 dev: Device obtained from fz_new_*_device.
35450: (8)                 transform: Transform to apply to page. May include for example
35451: (8)                 scaling and rotation, see fz_scale, fz_rotate and fz_concat.
35452: (8)                 Set to fz_identity if no transformation is desired.
35453: (8)                 cookie: Communication mechanism between caller and library
35454: (8)                 rendering the page. Intended for multi-threaded applications,
35455: (8)                 while single-threaded applications set cookie to NULL. The
35456: (8)                 caller may abort an ongoing rendering of a page. Cookie also
35457: (8)                 communicates progress information back to the caller. The
35458: (8)                 fields inside cookie are continually updated while the page is
```

```
35459: (8)                rendering.
35460: (4)            """
35461: (4)            return _mupdf.fz_run_page(page, dev, transform, cookie)
35462: (0)        def fz_run_page_annots(page, dev, transform, cookie):
35463: (4)            r"""
35464: (4)            Class-aware wrapper for `::fz_run_page_annots()`.
35465: (8)                Run the annotations on a page through a device.
35466: (4)            """
35467: (4)            return _mupdf.fz_run_page_annots(page, dev, transform, cookie)
35468: (0)        def fz_run_page_contents(page, dev, transform, cookie):
35469: (4)            r"""
35470: (4)            Class-aware wrapper for `::fz_run_page_contents()`.
35471: (8)                Run a page through a device. Just the main
35472: (8)                page content, without the annotations, if any.
35473: (8)                page: Page obtained from fz_load_page.
35474: (8)                dev: Device obtained from fz_new_*_device.
35475: (8)                transform: Transform to apply to page. May include for example
35476: (8)                scaling and rotation, see fz_scale, fz_rotate and fz_concat.
35477: (8)                Set to fz_identity if no transformation is desired.
35478: (8)                cookie: Communication mechanism between caller and library
35479: (8)                rendering the page. Intended for multi-threaded applications,
35480: (8)                while single-threaded applications set cookie to NULL. The
35481: (8)                caller may abort an ongoing rendering of a page. Cookie also
35482: (8)                communicates progress information back to the caller. The
35483: (8)                fields inside cookie are continually updated while the page is
35484: (8)                rendering.
35485: (4)            """
35486: (4)            return _mupdf.fz_run_page_contents(page, dev, transform, cookie)
35487: (0)        def fz_run_page_widgets(page, dev, transform, cookie):
35488: (4)            r"""
35489: (4)            Class-aware wrapper for `::fz_run_page_widgets()`.
35490: (8)                Run the widgets on a page through a device.
35491: (4)            """
35492: (4)            return _mupdf.fz_run_page_widgets(page, dev, transform, cookie)
35493: (0)        def fz_run_t3_glyph(font, gid, trm, dev):
35494: (4)            r"""
35495: (4)            Class-aware wrapper for `::fz_run_t3_glyph()`.
35496: (8)                Run a glyph from a Type3 font to
35497: (8)                a given device.
35498: (8)                font: The font to find the glyph in.
35499: (8)                gid: The glyph to run.
35500: (8)                trm: The transform to apply.
35501: (8)                dev: The device to render onto.
35502: (4)            """
35503: (4)            return _mupdf.fz_run_t3_glyph(font, gid, trm, dev)
35504: (0)        def fz_runeidx(str, p):
35505: (4)            r"""
35506: (4)            Class-aware wrapper for `::fz_runeidx()`.
35507: (8)                Compute the index of a rune in a string.
35508: (8)                str: Pointer to beginning of a string.
35509: (8)                p: Pointer to a char in str.
35510: (8)                Returns the index of the rune pointed to by p in str.
35511: (4)            """
35512: (4)            return _mupdf.fz_runeidx(str, p)
35513: (0)        def fz_runelen(rune):
35514: (4)            r"""
35515: (4)            Class-aware wrapper for `::fz_runelen()`.
35516: (8)                Count how many chars are required to represent a rune.
35517: (8)                rune: The rune to encode.
35518: (8)                Returns the number of bytes required to represent this run in
35519: (8)                UTF8.
35520: (4)            """
35521: (4)            return _mupdf.fz_runelen(rune)
35522: (0)        def fz_runeptr(str, idx):
35523: (4)            r"""
35524: (4)            Class-aware wrapper for `::fz_runeptr()`.
35525: (8)                Obtain a pointer to the char representing the rune
35526: (8)                at a given index.
35527: (8)                str: Pointer to beginning of a string.
```

```
35528: (8)                  idx: Index of a rune to return a char pointer to.
35529: (8)                  Returns a pointer to the char where the desired rune starts,
35530: (8)                  or NULL if the string ends before the index is reached.
35531: (4)              """
35532: (4)              return _mupdf.fz_runeptr(str, idx)
35533: (0)          def fz_runetochar(str, rune):
35534: (4)              r"""
35535: (4)              Class-aware wrapper for `::fz_runetochar()`.
35536: (8)                  UTF8 encode a rune to a sequence of chars.
35537: (8)                  str: Pointer to a place to put the UTF8 encoded character.
35538: (8)                  rune: Pointer to a 'rune'.
35539: (8)                  Returns the number of bytes the rune took to output.
35540: (4)              """
35541: (4)              return _mupdf.fz_runetochar(str, rune)
35542: (0)          def fz_samples_get(pixmap, offset):
35543: (4)              r"""
35544: (4)              Class-aware wrapper for `::fz_samples_get()`.
35545: (4)              Provides simple (but slow) access to pixmap data from Python and C#.
35546: (4)              """
35547: (4)              return _mupdf.fz_samples_get(pixmap, offset)
35548: (0)          def fz_samples_set(pixmap, offset, value):
35549: (4)              r"""
35550: (4)              Class-aware wrapper for `::fz_samples_set()`.
35551: (4)              Provides simple (but slow) write access to pixmap data from Python and
35552: (4)              C#.
35553: (4)              """
35554: (4)              return _mupdf.fz_samples_set(pixmap, offset, value)
35555: (0)          def fz_save_accelerator(doc, accel):
35556: (4)              r"""
35557: (4)              Class-aware wrapper for `::fz_save_accelerator()`.
35558: (8)                  Save accelerator data for the document to a given file.
35559: (4)              """
35560: (4)              return _mupdf.fz_save_accelerator(doc, accel)
35561: (0)          def fz_save_bitmap_as_pbm(bitmap, filename):
35562: (4)              r"""
35563: (4)              Class-aware wrapper for `::fz_save_bitmap_as_pbm()`.
35564: (8)                  Save a bitmap as a pbm.
35565: (4)              """
35566: (4)              return _mupdf.fz_save_bitmap_as_pbm(bitmap, filename)
35567: (0)          def fz_save_bitmap_as_pcl(bitmap, filename, append, pcl):
35568: (4)              r"""
35569: (4)              Class-aware wrapper for `::fz_save_bitmap_as_pcl()`.
35570: (8)                  Save a bitmap as mono PCL.
35571: (4)              """
35572: (4)              return _mupdf.fz_save_bitmap_as_pcl(bitmap, filename, append, pcl)
35573: (0)          def fz_save_bitmap_as_pkm(bitmap, filename):
35574: (4)              r"""
35575: (4)              Class-aware wrapper for `::fz_save_bitmap_as_pkm()`.
35576: (8)                  Save a CMYK bitmap as a pkm.
35577: (4)              """
35578: (4)              return _mupdf.fz_save_bitmap_as_pkm(bitmap, filename)
35579: (0)          def fz_save_bitmap_as_pwg(bitmap, filename, append, pwg):
35580: (4)              r"""
35581: (4)              Class-aware wrapper for `::fz_save_bitmap_as_pwg()`.
35582: (8)                  Save a bitmap as a PWG.
35583: (4)              """
35584: (4)              return _mupdf.fz_save_bitmap_as_pwg(bitmap, filename, append, pwg)
35585: (0)          def fz_save_buffer(buf, filename):
35586: (4)              r"""
35587: (4)              Class-aware wrapper for `::fz_save_buffer()`.
35588: (8)                  Save the contents of a buffer to a file.
35589: (4)              """
35590: (4)              return _mupdf.fz_save_buffer(buf, filename)
35591: (0)          def fz_save_pixmap_as_jpeg(pixmap, filename, quality):
35592: (4)              r"""
35593: (4)              Class-aware wrapper for `::fz_save_pixmap_as_jpeg()`.
35594: (8)                  Save a pixmap as a JPEG.
35595: (4)              """
35596: (4)              return _mupdf.fz_save_pixmap_as_jpeg(pixmap, filename, quality)
```

```
35597: (0)          def fz_save_pixmap_as_jpx(pixmap, filename, q):
35598: (4)              r"""
35599: (4)              Class-aware wrapper for `::fz_save_pixmap_as_jpx()`.
35600: (8)                  Save pixmap data as JP2K with no subsampling.
35601: (8)                  quality = 100 = lossless
35602: (8)                  otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
35603: (4)              """
35604: (4)              return _mupdf.fz_save_pixmap_as_jpx(pixmap, filename, q)
35605: (0)          def fz_save_pixmap_as_pam(pixmap, filename):
35606: (4)              r"""
35607: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pam()`.
35608: (8)                  Save a pixmap as a pnm (greyscale, rgb or cmyk, with or without
35609: (8)                  alpha).
35610: (4)              """
35611: (4)              return _mupdf.fz_save_pixmap_as_pam(pixmap, filename)
35612: (0)          def fz_save_pixmap_as_pbm(pixmap, filename):
35613: (4)              r"""
35614: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pbm()`.
35615: (8)                  Save a pixmap as a pbm. (Performing halftoning).
35616: (4)              """
35617: (4)              return _mupdf.fz_save_pixmap_as_pbm(pixmap, filename)
35618: (0)          def fz_save_pixmap_as_pcl(pixmap, filename, append, pcl):
35619: (4)              r"""
35620: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pcl()`.
35621: (8)                  Save an (RGB) pixmap as color PCL.
35622: (4)              """
35623: (4)              return _mupdf.fz_save_pixmap_as_pcl(pixmap, filename, append, pcl)
35624: (0)          def fz_save_pixmap_as_pclm(pixmap, filename, append, options):
35625: (4)              r"""
35626: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pclm()`.
35627: (8)                  Save a (Greyscale or RGB) pixmap as pclm.
35628: (4)              """
35629: (4)              return _mupdf.fz_save_pixmap_as_pclm(pixmap, filename, append, options)
35630: (0)          def fz_save_pixmap_as_pdfocr(pixmap, filename, append, options):
35631: (4)              r"""
35632: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pdfocr()`.
35633: (8)                  Save a (Greyscale or RGB) pixmap as pdfocr.
35634: (4)              """
35635: (4)              return _mupdf.fz_save_pixmap_as_pdfocr(pixmap, filename, append, options)
35636: (0)          def fz_save_pixmap_as_pkm(pixmap, filename):
35637: (4)              r"""
35638: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pkm()`.
35639: (8)                  Save a CMYK pixmap as a pkm. (Performing halftoning).
35640: (4)              """
35641: (4)              return _mupdf.fz_save_pixmap_as_pkm(pixmap, filename)
35642: (0)          def fz_save_pixmap_as_png(pixmap, filename):
35643: (4)              r"""
35644: (4)              Class-aware wrapper for `::fz_save_pixmap_as_png()`.
35645: (8)                  Save a (Greyscale or RGB) pixmap as a png.
35646: (4)              """
35647: (4)              return _mupdf.fz_save_pixmap_as_png(pixmap, filename)
35648: (0)          def fz_save_pixmap_as_pnm(pixmap, filename):
35649: (4)              r"""
35650: (4)              Class-aware wrapper for `::fz_save_pixmap_as_pnm()`.
35651: (8)                  Save a pixmap as a pnm (greyscale or rgb, no alpha).
35652: (4)              """
35653: (4)              return _mupdf.fz_save_pixmap_as_pnm(pixmap, filename)
35654: (0)          def fz_save_pixmap_as_ps(pixmap, filename, append):
35655: (4)              r"""
35656: (4)              Class-aware wrapper for `::fz_save_pixmap_as_ps()`.
35657: (8)                  Save a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
35658: (4)              """
35659: (4)              return _mupdf.fz_save_pixmap_as_ps(pixmap, filename, append)
35660: (0)          def fz_save_pixmap_as_psd(pixmap, filename):
35661: (4)              r"""
35662: (4)              Class-aware wrapper for `::fz_save_pixmap_as_psd()`.
35663: (8)                  Save a pixmap as a PSD file.
35664: (4)              """
```

```
35665: (4)             return _mupdf.fz_save_pixmap_as_psd(pixmap, filename)
35666: (0)         def fz_save_pixmap_as_pwg(pixmap, filename, append, pwg):
35667: (4)             r"""
35668: (4)             Class-aware wrapper for `::fz_save_pixmap_as_pwg()`.
35669: (8)                 Save a pixmap as a PWG.
35670: (4)             """
35671: (4)             return _mupdf.fz_save_pixmap_as_pwg(pixmap, filename, append, pwg)
35672: (0)         def fz_scale(sx, sy):
35673: (4)             r"""
35674: (4)             Class-aware wrapper for `::fz_scale()`.
35675: (8)                 Create a scaling matrix.
35676: (8)                 The returned matrix is of the form [ sx 0 0 sy 0 0 ].
35677: (8)                 m: Pointer to the matrix to populate
35678: (8)                 sx, sy: Scaling factors along the X- and Y-axes. A scaling
35679: (8)                 factor of 1.0 will not cause any scaling along the relevant
35680: (8)                 axis.
35681: (8)                 Returns m.
35682: (4)             """
35683: (4)             return _mupdf.fz_scale(sx, sy)
35684: (0)         def fz_scale_pixmap(src, x, y, w, h, clip):
35685: (4)             r""" Class-aware wrapper for `::fz_scale_pixmap()`."""
35686: (4)             return _mupdf.fz_scale_pixmap(src, x, y, w, h, clip)
35687: (0)         def fz_search_chapter_page_number(doc, chapter, page, needle, hit_mark,
hit_bbox, hit_max):
35688: (4)             r"""
35689: (4)             Class-aware wrapper for `::fz_search_chapter_page_number()`.
35690: (4)             This function has out-params. Python/C# wrappers look like:
35691: (8)                 `fz_search_chapter_page_number(::fz_document *doc, int chapter, int
page, const char *needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
35692: (4)             """
35693: (4)             return _mupdf.fz_search_chapter_page_number(doc, chapter, page, needle,
hit_mark, hit_bbox, hit_max)
35694: (0)         def fz_search_display_list(list, needle, hit_mark, hit_bbox, hit_max):
35695: (4)             r"""
35696: (4)             Class-aware wrapper for `::fz_search_display_list()`.
35697: (4)             This function has out-params. Python/C# wrappers look like:
35698: (8)                 `fz_search_display_list(::fz_display_list *list, const char *needle,
::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
35699: (4)             """
35700: (4)             return _mupdf.fz_search_display_list(list, needle, hit_mark, hit_bbox,
hit_max)
35701: (0)         def fz_search_page(page, needle, hit_mark, hit_bbox, hit_max):
35702: (4)             r"""
35703: (4)             Class-aware wrapper for `::fz_search_page()`.
35704: (4)             This function has out-params. Python/C# wrappers look like:
35705: (8)                 `fz_search_page(::fz_page *page, const char *needle, ::fz_quad
*hit_bbox, int hit_max)` => `(int, int hit_mark)`
35706: (8)                 Search for the 'needle' text on the page.
35707: (8)                 Record the hits in the hit_bbox array and return the number of
35708: (8)                 hits. Will stop looking once it has filled hit_max rectangles.
35709: (4)             """
35710: (4)             return _mupdf.fz_search_page(page, needle, hit_mark, hit_bbox, hit_max)
35711: (0)         def fz_search_page2(doc, number, needle, hit_max):
35712: (4)             r"""
35713: (4)             Class-aware wrapper for `::fz_search_page2()`.
35714: (4)             C++ alternative to fz_search_page() that returns information in a
std::vector.
35715: (4)             """
35716: (4)             return _mupdf.fz_search_page2(doc, number, needle, hit_max)
35717: (0)         def fz_search_page_number(doc, number, needle, hit_mark, hit_bbox, hit_max):
35718: (4)             r"""
35719: (4)             Class-aware wrapper for `::fz_search_page_number()`.
35720: (4)             This function has out-params. Python/C# wrappers look like:
35721: (8)                 `fz_search_page_number(::fz_document *doc, int number, const char
*needle, ::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
35722: (4)             """
35723: (4)             return _mupdf.fz_search_page_number(doc, number, needle, hit_mark,
hit_bbox, hit_max)
35724: (0)         def fz_search_stext_page(text, needle, hit_mark, hit_bbox, hit_max):
```

```
35725: (4)              r"""
35726: (4)              Class-aware wrapper for `::fz_search_stext_page()`.
35727: (4)              This function has out-params. Python/C# wrappers look like:
35728: (8)                  `fz_search_stext_page(::fz_stext_page *text, const char *needle,
::fz_quad *hit_bbox, int hit_max)` => `(int, int hit_mark)`
35729: (8)                  Search for occurrence of 'needle' in text page.
35730: (8)                  Return the number of hits and store hit quads in the passed in
35731: (8)                  array.
35732: (8)                  NOTE: This is an experimental interface and subject to change
35733: (8)                  without notice.
35734: (4)              """
35735: (4)              return _mupdf.fz_search_stext_page(text, needle, hit_mark, hit_bbox,
hit_max)
35736: (0)          def fz_seek(stm, offset, whence):
35737: (4)              r"""
35738: (4)              Class-aware wrapper for `::fz_seek()`.
35739: (8)                  Seek within a stream.
35740: (8)                  stm: The stream to seek within.
35741: (8)                  offset: The offset to seek to.
35742: (8)                  whence: From where the offset is measured (see fseek).
35743: (8)                  SEEK_SET - start of stream.
35744: (8)                  SEEK_CUR - current position.
35745: (8)                  SEEK_END - end of stream.
35746: (4)              """
35747: (4)              return _mupdf.fz_seek(stm, offset, whence)
35748: (0)          def fz_seek_output(out, off, whence):
35749: (4)              r"""
35750: (4)              Class-aware wrapper for `::fz_seek_output()`.
35751: (8)                  Seek to the specified file position.
35752: (8)                  See fseek for arguments.
35753: (8)                  Throw an error on unseekable outputs.
35754: (4)              """
35755: (4)              return _mupdf.fz_seek_output(out, off, whence)
35756: (0)          def fz_separation_current_behavior(sep, separation):
35757: (4)              r"""
35758: (4)              Class-aware wrapper for `::fz_separation_current_behavior()`.
35759: (8)                  Test for the current behavior of a separation.
35760: (4)              """
35761: (4)              return _mupdf.fz_separation_current_behavior(sep, separation)
35762: (0)          def fz_separation_equivalent(seps, idx, dst_cs, dst_color, prf, color_params):
35763: (4)              r"""
35764: (4)              Class-aware wrapper for `::fz_separation_equivalent()`.
35765: (4)              This function has out-params. Python/C# wrappers look like:
35766: (8)                  `fz_separation_equivalent(const ::fz_separations *seps, int idx,
::fz_colorspace *dst_cs, ::fz_colorspace *prf, ::fz_color_params color_params)` => float dst_color
35767: (8)                  Get the equivalent separation color in a given colorspace.
35768: (4)              """
35769: (4)              return _mupdf.fz_separation_equivalent(seps, idx, dst_cs, dst_color, prf,
color_params)
35770: (0)          def fz_separation_name(sep, separation):
35771: (4)              r""" Class-aware wrapper for `::fz_separation_name()`."""
35772: (4)              return _mupdf.fz_separation_name(sep, separation)
35773: (0)          def fz_set_aa_level(bits):
35774: (4)              r"""
35775: (4)              Class-aware wrapper for `::fz_set_aa_level()`.
35776: (8)                  Set the number of bits of antialiasing we should
35777: (8)                  use (for both text and graphics).
35778: (8)                  bits: The number of bits of antialiasing to use (values are
35779: (8)                  clamped to within the 0 to 8 range).
35780: (4)              """
35781: (4)              return _mupdf.fz_set_aa_level(bits)
35782: (0)          def fz_set_compressed_image_buffer(cimg, buf):
35783: (4)              r""" Class-aware wrapper for `::fz_set_compressed_image_buffer()`."""
35784: (4)              return _mupdf.fz_set_compressed_image_buffer(cimg, buf)
35785: (0)          def fz_set_default_cmyk(default_cs, cs):
35786: (4)              r""" Class-aware wrapper for `::fz_set_default_cmyk()`."""
35787: (4)              return _mupdf.fz_set_default_cmyk(default_cs, cs)
35788: (0)          def fz_set_default_colorspaces(dev, default_cs):
35789: (4)              r""" Class-aware wrapper for `::fz_set_default_colorspaces()`."""
```

```
35790: (4)             return _mupdf.fz_set_default_colorspaces(dev, default_cs)
35791: (0)         def fz_set_default_gray(default_cs, cs):
35792: (4)             r"""
35793: (4)             Class-aware wrapper for `::fz_set_default_gray()`.
35794: (8)                 Set new defaults within the default colorspace structure.
35795: (8)                 New references are taken to the new default, and references to
35796: (8)                 the old defaults dropped.
35797: (8)                 Never throws exceptions.
35798: (4)             """
35799: (4)             return _mupdf.fz_set_default_gray(default_cs, cs)
35800: (0)         def fz_set_default_output_intent(default_cs, cs):
35801: (4)             r""" Class-aware wrapper for `::fz_set_default_output_intent()`."""
35802: (4)             return _mupdf.fz_set_default_output_intent(default_cs, cs)
35803: (0)         def fz_set_default_rgb(default_cs, cs):
35804: (4)             r""" Class-aware wrapper for `::fz_set_default_rgb()`."""
35805: (4)             return _mupdf.fz_set_default_rgb(default_cs, cs)
35806: (0)         def fz_set_error_callback(error_cb, user):
35807: (4)             r"""
35808: (4)             Class-aware wrapper for `::fz_set_error_callback()`.
35809: (8)                 Set the error callback. This will be called as part of the
35810: (8)                 exception handling.
35811: (8)                 The callback must not throw exceptions!
35812: (4)             """
35813: (4)             return _mupdf.fz_set_error_callback(error_cb, user)
35814: (0)         def fz_set_font_bbox(font, xmin, ymin, xmax, ymax):
35815: (4)             r"""
35816: (4)             Class-aware wrapper for `::fz_set_font_bbox()`.
35817: (8)                 Set the font bbox.
35818: (8)                 font: The font to set the bbox for.
35819: (8)                 xmin, ymin, xmax, ymax: The bounding box.
35820: (4)             """
35821: (4)             return _mupdf.fz_set_font_bbox(font, xmin, ymin, xmax, ymax)
35822: (0)         def fz_set_font_embedding(font, embed):
35823: (4)             r"""
35824: (4)             Class-aware wrapper for `::fz_set_font_embedding()`.
35825: (8)                 Control whether a given font should be embedded or not when writing.
35826: (4)             """
35827: (4)             return _mupdf.fz_set_font_embedding(font, embed)
35828: (0)         def fz_set_graphics_aa_level(bits):
35829: (4)             r"""
35830: (4)             Class-aware wrapper for `::fz_set_graphics_aa_level()`.
35831: (8)                 Set the number of bits of antialiasing we
35832: (8)                 should use for graphics.
35833: (8)                 bits: The number of bits of antialiasing to use (values are
35834: (8)                 clamped to within the 0 to 8 range).
35835: (4)             """
35836: (4)             return _mupdf.fz_set_graphics_aa_level(bits)
35837: (0)         def fz_set_graphics_min_line_width(min_line_width):
35838: (4)             r"""
35839: (4)             Class-aware wrapper for `::fz_set_graphics_min_line_width()`.
35840: (8)                 Set the minimum line width to be
35841: (8)                 used for stroked lines.
35842: (8)                 min_line_width: The minimum line width to use (in pixels).
35843: (4)             """
35844: (4)             return _mupdf.fz_set_graphics_min_line_width(min_line_width)
35845: (0)         def fz_set_link_rect(link, rect):
35846: (4)             r""" Class-aware wrapper for `::fz_set_link_rect()`."""
35847: (4)             return _mupdf.fz_set_link_rect(link, rect)
35848: (0)         def fz_set_link_uri(link, uri):
35849: (4)             r""" Class-aware wrapper for `::fz_set_link_uri()`."""
35850: (4)             return _mupdf.fz_set_link_uri(link, uri)
35851: (0)         def fz_set_metadata(doc, key, value):
35852: (4)             r""" Class-aware wrapper for `::fz_set_metadata()`."""
35853: (4)             return _mupdf.fz_set_metadata(doc, key, value)
35854: (0)         def fz_set_pixmap_image_tile(cimg, pix):
35855: (4)             r""" Class-aware wrapper for `::fz_set_pixmap_image_tile()`."""
35856: (4)             return _mupdf.fz_set_pixmap_image_tile(cimg, pix)
35857: (0)         def fz_set_pixmap_resolution(pix, xres, yres):
35858: (4)             r"""
```

```
(4)             Class-aware wrapper for `::fz_set_pixmap_resolution()`.
(8)                 Set the pixels per inch resolution of the pixmap.
(4)             """
(4)             return _mupdf.fz_set_pixmap_resolution(pix, xres, yres)
(0)         def fz_set_separation_behavior(sep, separation, behavior):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_separation_behavior()`.
(8)                 Control the rendering of a given separation.
(4)             """
(4)             return _mupdf.fz_set_separation_behavior(sep, separation, behavior)
(0)         def fz_set_stddbg(out):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_stddbg()`.
(8)                 Set the output stream to be used for fz_stddbg. Set to NULL to
(8)                 reset to default (stderr).
(4)             """
(4)             return _mupdf.fz_set_stddbg(out)
(0)         def fz_set_text_aa_level(bits):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_text_aa_level()`.
(8)                 Set the number of bits of antialiasing we
(8)                 should use for text.
(8)                 bits: The number of bits of antialiasing to use (values are
(8)                 clamped to within the 0 to 8 range).
(4)             """
(4)             return _mupdf.fz_set_text_aa_level(bits)
(0)         def fz_set_use_document_css(use):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_use_document_css()`.
(8)                 Toggle whether to respect document styles in HTML and EPUB.
(4)             """
(4)             return _mupdf.fz_set_use_document_css(use)
(0)         def fz_set_user_context(user):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_user_context()`.
(8)                 Set the user field in the context.
(8)                 NULL initially, this field can be set to any opaque value
(8)                 required by the user. It is copied on clones.
(4)             """
(4)             return _mupdf.fz_set_user_context(user)
(0)         def fz_set_user_css(text):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_user_css()`.
(8)                 Set the user stylesheet source text for use with HTML and EPUB.
(4)             """
(4)             return _mupdf.fz_set_user_css(text)
(0)         def fz_set_warning_callback(warning_cb, user):
(4)             r"""
(4)             Class-aware wrapper for `::fz_set_warning_callback()`.
(8)                 Set the warning callback. This will be called as part of the
(8)                 exception handling.
(8)                 The callback must not throw exceptions!
(4)             """
(4)             return _mupdf.fz_set_warning_callback(warning_cb, user)
(0)         def fz_sha256_final(state, digest):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha256_final()`.
(8)                 MD5 finalization. Ends an MD5 message-digest operation, writing
(8)                 the message digest and zeroizing the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha256_final(state, digest)
(0)         def fz_sha256_init(state):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha256_init()`.
(8)                 SHA256 initialization. Begins an SHA256 operation, initialising
(8)                 the supplied context.
(8)                 Never throws an exception.
(4)             """
```

```
(4)             return _mupdf.fz_sha256_init(state)
(0)         def fz_sha256_update(state, input, inlen):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha256_update()`.
(8)                 SHA256 block update operation. Continues an SHA256 message-
(8)                 digest operation, processing another message block, and updating
(8)                 the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha256_update(state, input, inlen)
(0)         def fz_sha384_final(state, digest):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha384_final()`.
(8)                 SHA384 finalization. Ends an SHA384 message-digest operation,
(8)                 writing the message digest and zeroizing the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha384_final(state, digest)
(0)         def fz_sha384_init(state):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha384_init()`.
(8)                 SHA384 initialization. Begins an SHA384 operation, initialising
(8)                 the supplied context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha384_init(state)
(0)         def fz_sha384_update(state, input, inlen):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha384_update()`.
(8)                 SHA384 block update operation. Continues an SHA384 message-
(8)                 digest operation, processing another message block, and updating
(8)                 the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha384_update(state, input, inlen)
(0)         def fz_sha512_final(state, digest):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha512_final()`.
(8)                 SHA512 finalization. Ends an SHA512 message-digest operation,
(8)                 writing the message digest and zeroizing the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha512_final(state, digest)
(0)         def fz_sha512_init(state):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha512_init()`.
(8)                 SHA512 initialization. Begins an SHA512 operation, initialising
(8)                 the supplied context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha512_init(state)
(0)         def fz_sha512_update(state, input, inlen):
(4)             r"""
(4)             Class-aware wrapper for `::fz_sha512_update()`.
(8)                 SHA512 block update operation. Continues an SHA512 message-
(8)                 digest operation, processing another message block, and updating
(8)                 the context.
(8)                 Never throws an exception.
(4)             """
(4)             return _mupdf.fz_sha512_update(state, input, inlen)
(0)         def fz_shear(sx, sy):
(4)             r"""
(4)             Class-aware wrapper for `::fz_shear()`.
(8)                 Create a shearing matrix.
(8)                 The returned matrix is of the form [ 1 sy sx 1 0 0 ].
(8)                 m: pointer to place to store returned matrix
(8)                 sx, sy: Shearing factors. A shearing factor of 0.0 will not
(8)                 cause any shearing along the relevant axis.
(8)                 Returns m.
```

```
35997: (4)              """
35998: (4)              return _mupdf.fz_shear(sx, sy)
35999: (0)          def fz_show_glyph(text, font, trm, glyph, unicode, wmode, bidi_level,
markup_dir, language):
36000: (4)              r"""
36001: (4)              Class-aware wrapper for `::fz_show_glyph()`.
36002: (8)                  Add a glyph/unicode value to a text object.
36003: (8)                  text: Text object to add to.
36004: (8)                  font: The font the glyph should be added in.
36005: (8)                  trm: The transform to use for the glyph.
36006: (8)                  glyph: The glyph id to add.
36007: (8)                  unicode: The unicode character for the glyph.
36008: (8)                  cid: The CJK CID value or raw character code.
36009: (8)                  wmode: 1 for vertical mode, 0 for horizontal.
36010: (8)                  bidi_level: The bidirectional level for this glyph.
36011: (8)                  markup_dir: The direction of the text as specified in the
36012: (8)                  markup.
36013: (8)                  language: The language in use (if known, 0 otherwise)
36014: (8)                  (e.g. FZ_LANG_zh_Hans).
36015: (8)                  Throws exception on failure to allocate.
36016: (4)              """
36017: (4)              return _mupdf.fz_show_glyph(text, font, trm, glyph, unicode, wmode,
bidi_level, markup_dir, language)
36018: (0)          def fz_show_glyph_aux(text, font, trm, glyph, unicode, cid, wmode, bidi_level,
markup_dir, lang):
36019: (4)              r""" Class-aware wrapper for `::fz_show_glyph_aux()`."""
36020: (4)              return _mupdf.fz_show_glyph_aux(text, font, trm, glyph, unicode, cid,
wmode, bidi_level, markup_dir, lang)
36021: (0)          def fz_show_string(text, font, trm, s, wmode, bidi_level, markup_dir,
language):
36022: (4)              r"""
36023: (4)              Class-aware wrapper for `::fz_show_string()`.
36024: (8)                  Add a UTF8 string to a text object.
36025: (8)                  text: Text object to add to.
36026: (8)                  font: The font the string should be added in.
36027: (8)                  trm: The transform to use.
36028: (8)                  s: The utf-8 string to add.
36029: (8)                  wmode: 1 for vertical mode, 0 for horizontal.
36030: (8)                  bidi_level: The bidirectional level for this glyph.
36031: (8)                  markup_dir: The direction of the text as specified in the markup.
36032: (8)                  language: The language in use (if known, 0 otherwise)
36033: (12)                     (e.g. FZ_LANG_zh_Hans).
36034: (8)                  Returns the transform updated with the advance width of the
36035: (8)                  string.
36036: (4)              """
36037: (4)              return _mupdf.fz_show_string(text, font, trm, s, wmode, bidi_level,
markup_dir, language)
36038: (0)          def fz_shrink_store(percent):
36039: (4)              r"""
36040: (4)              Class-aware wrapper for `::fz_shrink_store()`.
36041: (8)                  Evict items from the store until the total size of
36042: (8)                  the objects in the store is reduced to a given percentage of its
36043: (8)                  current size.
36044: (8)                  percent: %age of current size to reduce the store to.
36045: (8)                  Returns non zero if we managed to free enough memory, zero
36046: (8)                  otherwise.
36047: (4)              """
36048: (4)              return _mupdf.fz_shrink_store(percent)
36049: (0)          def fz_skip(stm, len):
36050: (4)              r"""
36051: (4)              Class-aware wrapper for `::fz_skip()`.
36052: (8)                  Read from a stream discarding data.
36053: (8)                  stm: The stream to read from.
36054: (8)                  len: The number of bytes to read.
36055: (8)                  Returns the number of bytes read. May throw exceptions.
36056: (4)              """
36057: (4)              return _mupdf.fz_skip(stm, len)
36058: (0)          def fz_skip_space(stm):
36059: (4)              r"""
```

```
36060: (4)             Class-aware wrapper for `::fz_skip_space()`.
36061: (8)                 Skip over whitespace (bytes <= 32) in a stream.
36062: (4)             """
36063: (4)             return _mupdf.fz_skip_space(stm)
36064: (0)         def fz_skip_string(stm, str):
36065: (4)             r"""
36066: (4)             Class-aware wrapper for `::fz_skip_string()`.
36067: (8)                 Skip over a given string in a stream. Return 0 if successfully
36068: (8)                 skipped, non-zero otherwise. As many characters will be skipped
36069: (8)                 over as matched in the string.
36070: (4)             """
36071: (4)             return _mupdf.fz_skip_string(stm, str)
36072: (0)         def fz_slice_buffer(buf, start, end):
36073: (4)             r"""
36074: (4)             Class-aware wrapper for `::fz_slice_buffer()`.
36075: (8)                 Create a new buffer with a (subset of) the data from the buffer.
36076: (8)                 start: if >= 0, offset from start of buffer, if < 0 offset from end of
buffer.
36077: (8)                 end: if >= 0, offset from start of buffer, if < 0 offset from end of
buffer.
36078: (4)             """
36079: (4)             return _mupdf.fz_slice_buffer(buf, start, end)
36080: (0)         def fz_snap_selection(page, ap, bp, mode):
36081: (4)             r""" Class-aware wrapper for `::fz_snap_selection()`."""
36082: (4)             return _mupdf.fz_snap_selection(page, ap, bp, mode)
36083: (0)         def fz_start_throw_on_repair():
36084: (4)             r""" Class-aware wrapper for `::fz_start_throw_on_repair()`."""
36085: (4)             return _mupdf.fz_start_throw_on_repair()
36086: (0)         def fz_stat_ctime(path):
36087: (4)             r""" Class-aware wrapper for `::fz_stat_ctime()`."""
36088: (4)             return _mupdf.fz_stat_ctime(path)
36089: (0)         def fz_stat_mtime(path):
36090: (4)             r""" Class-aware wrapper for `::fz_stat_mtime()`."""
36091: (4)             return _mupdf.fz_stat_mtime(path)
36092: (0)         def fz_store_item(key, val, itemsize, type):
36093: (4)             r"""
36094: (4)             Class-aware wrapper for `::fz_store_item()`.
36095: (8)                 Add an item to the store.
36096: (8)                 Add an item into the store, returning NULL for success. If an
36097: (8)                 item with the same key is found in the store, then our item will
36098: (8)                 not be inserted, and the function will return a pointer to that
36099: (8)                 value instead. This function takes its own reference to val, as
36100: (8)                 required (i.e. the caller maintains ownership of its own
36101: (8)                 reference).
36102: (8)                 key: The key used to index the item.
36103: (8)                 val: The value to store.
36104: (8)                 itemsize: The size in bytes of the value (as counted towards the
36105: (8)                 store size).
36106: (8)                 type: Functions used to manipulate the key.
36107: (4)             """
36108: (4)             return _mupdf.fz_store_item(key, val, itemsize, type)
36109: (0)         def fz_store_scavenge(size, phase):
36110: (4)             r"""
36111: (4)             Class-aware wrapper for `::fz_store_scavenge()`.
36112: (4)             This function has out-params. Python/C# wrappers look like:
36113: (8)                 `fz_store_scavenge(size_t size)` => `(int, int phase)`
36114: (8)                 Internal function used as part of the scavenging
36115: (8)                 allocator; when we fail to allocate memory, before returning a
36116: (8)                 failure to the caller, we try to scavenge space within the store
36117: (8)                 by evicting at least 'size' bytes. The allocator then retries.
36118: (8)                 size: The number of bytes we are trying to have free.
36119: (8)                 phase: What phase of the scavenge we are in. Updated on exit.
36120: (8)                 Returns non zero if we managed to free any memory.
36121: (4)             """
36122: (4)             return _mupdf.fz_store_scavenge(size, phase)
36123: (0)         def fz_store_scavenge_external(size, phase):
36124: (4)             r"""
36125: (4)             Class-aware wrapper for `::fz_store_scavenge_external()`.
36126: (4)             This function has out-params. Python/C# wrappers look like:
```

```
36127: (8)                `fz_store_scavenge_external(size_t size)` => `(int, int phase)`
36128: (8)                External function for callers to use
36129: (8)                to scavenge while trying allocations.
36130: (8)                size: The number of bytes we are trying to have free.
36131: (8)                phase: What phase of the scavenge we are in. Updated on exit.
36132: (8)                Returns non zero if we managed to free any memory.
36133: (4)            """
36134: (4)            return _mupdf.fz_store_scavenge_external(size, phase)
36135: (0)        def fz_story_document(story):
36136: (4)            r""" Class-aware wrapper for `::fz_story_document()`."""
36137: (4)            return _mupdf.fz_story_document(story)
36138: (0)        def fz_story_positions(story, cb, arg):
36139: (4)            r""" Class-aware wrapper for `::fz_story_positions()`."""
36140: (4)            return _mupdf.fz_story_positions(story, cb, arg)
36141: (0)        def fz_story_warnings(story):
36142: (4)            r""" Class-aware wrapper for `::fz_story_warnings()`."""
36143: (4)            return _mupdf.fz_story_warnings(story)
36144: (0)        def fz_strcasecmp(a, b):
36145: (4)            r"""
36146: (4)            Class-aware wrapper for `::fz_strcasecmp()`.
36147: (8)                Case insensitive (ASCII only) string comparison.
36148: (4)            """
36149: (4)            return _mupdf.fz_strcasecmp(a, b)
36150: (0)        def fz_strdup(s):
36151: (4)            r"""
36152: (4)            Class-aware wrapper for `::fz_strdup()`.
36153: (8)                Portable strdup implementation, using fz allocators.
36154: (4)            """
36155: (4)            return _mupdf.fz_strdup(s)
36156: (0)        def fz_stream_filename(stm):
36157: (4)            r"""
36158: (4)            Class-aware wrapper for `::fz_stream_filename()`.
36159: (8)                Return the filename (UTF-8 encoded) from which a stream was opened.
36160: (8)                Returns NULL if the filename is not available (or the stream was
36161: (8)                opened from a source other than a file).
36162: (4)            """
36163: (4)            return _mupdf.fz_stream_filename(stm)
36164: (0)        def fz_stream_from_output(arg_0):
36165: (4)            r"""
36166: (4)            Class-aware wrapper for `::fz_stream_from_output()`.
36167: (8)                Obtain the fz_output in the form of a fz_stream.
36168: (8)                This allows data to be read back from some forms of fz_output
36169: (8)                object. When finished reading, the fz_stream should be released
36170: (8)                by calling fz_drop_stream. Until the fz_stream is dropped, no
36171: (8)                further operations should be performed on the fz_output object.
36172: (4)            """
36173: (4)            return _mupdf.fz_stream_from_output(arg_0)
36174: (0)        def fz_string_from_box_type(box):
36175: (4)            r""" Class-aware wrapper for `::fz_string_from_box_type()`."""
36176: (4)            return _mupdf.fz_string_from_box_type(box)
36177: (0)        def fz_string_from_buffer(buf):
36178: (4)            r"""
36179: (4)            Class-aware wrapper for `::fz_string_from_buffer()`.
36180: (8)                Ensure that a buffer's data ends in a
36181: (8)                0 byte, and return a pointer to it.
36182: (4)            """
36183: (4)            return _mupdf.fz_string_from_buffer(buf)
36184: (0)        def fz_string_from_text_language(str, lang):
36185: (4)            r"""
36186: (4)            Class-aware wrapper for `::fz_string_from_text_language()`.
36187: (8)                Recover ISO 639 (639-{1,2,3,5}) language specification
36188: (8)                strings losslessly from a 15 bit fz_text_language code.
36189: (8)                No validation is carried out. See note above.
36190: (4)            """
36191: (4)            return _mupdf.fz_string_from_text_language(str, lang)
36192: (0)        def fz_string_from_text_language2(lang):
36193: (4)            r"""
36194: (4)            Class-aware wrapper for `::fz_string_from_text_language2()`.
36195: (4)            C++ alternative to fz_string_from_text_language() that returns information
```

```
in a std::string.
36196: (4)              """
36197: (4)              return _mupdf.fz_string_from_text_language2(lang)
36198: (0)          def fz_strlcat(dst, src, n):
36199: (4)              r"""
36200: (4)              Class-aware wrapper for `::fz_strlcat()`.
36201: (8)                  Concatenate 2 strings, with a maximum length.
36202: (8)                  dst: pointer to first string in a buffer of n bytes.
36203: (8)                  src: pointer to string to concatenate.
36204: (8)                  n: Size (in bytes) of buffer that dst is in.
36205: (8)                  Returns the real length that a concatenated dst + src would have
36206: (8)                  been (not including terminator).
36207: (4)              """
36208: (4)              return _mupdf.fz_strlcat(dst, src, n)
36209: (0)          def fz_strlcpy(dst, src, n):
36210: (4)              r"""
36211: (4)              Class-aware wrapper for `::fz_strlcpy()`.
36212: (8)                  Copy at most n-1 chars of a string into a destination
36213: (8)                  buffer with null termination, returning the real length of the
36214: (8)                  initial string (excluding terminator).
36215: (8)                  dst: Destination buffer, at least n bytes long.
36216: (8)                  src: C string (non-NULL).
36217: (8)                  n: Size of dst buffer in bytes.
36218: (8)                  Returns the length (excluding terminator) of src.
36219: (4)              """
36220: (4)              return _mupdf.fz_strlcpy(dst, src, n)
36221: (0)          def fz_strncasecmp(a, b, n):
36222: (4)              r""" Class-aware wrapper for `::fz_strncasecmp()`."""
36223: (4)              return _mupdf.fz_strncasecmp(a, b, n)
36224: (0)          def fz_strnlen(s, maxlen):
36225: (4)              r"""
36226: (4)              Class-aware wrapper for `::fz_strnlen()`.
36227: (8)                  Return strlen(s), if that is less than maxlen, or maxlen if
36228: (8)                  there is no null byte ('') among the first maxlen bytes.
36229: (4)              """
36230: (4)              return _mupdf.fz_strnlen(s, maxlen)
36231: (0)          def fz_stroke_path(dev, path, stroke, ctm, colorspace, color, alpha,
color_params):
36232: (4)              r""" Class-aware wrapper for `::fz_stroke_path()`."""
36233: (4)              return _mupdf.fz_stroke_path(dev, path, stroke, ctm, colorspace, color,
alpha, color_params)
36234: (0)          def fz_stroke_text(dev, text, stroke, ctm, colorspace, color, alpha,
color_params):
36235: (4)              r""" Class-aware wrapper for `::fz_stroke_text()`."""
36236: (4)              return _mupdf.fz_stroke_text(dev, text, stroke, ctm, colorspace, color,
alpha, color_params)
36237: (0)          def fz_strsep(stringp, delim):
36238: (4)              r"""
36239: (4)              Class-aware wrapper for `::fz_strsep()`.
36240: (4)              This function has out-params. Python/C# wrappers look like:
36241: (8)                  `fz_strsep(const char *delim)` => `(char *, char *stringp)`
36242: (8)                  Given a pointer to a C string (or a pointer to NULL) break
36243: (8)                  it at the first occurrence of a delimiter char (from a given
36244: (8)                  set).
36245: (8)                  stringp: Pointer to a C string pointer (or NULL). Updated on
36246: (8)                  exit to point to the first char of the string after the
36247: (8)                  delimiter that was found. The string pointed to by stringp will
36248: (8)                  be corrupted by this call (as the found delimiter will be
36249: (8)                  overwritten by 0).
36250: (8)                  delim: A C string of acceptable delimiter characters.
36251: (8)                  Returns a pointer to a C string containing the chars of stringp
36252: (8)                  up to the first delimiter char (or the end of the string), or
36253: (8)                  NULL.
36254: (4)              """
36255: (4)              return _mupdf.fz_strsep(stringp, delim)
36256: (0)          def fz_strtof(s, es):
36257: (4)              r"""
36258: (4)              Class-aware wrapper for `::fz_strtof()`.
36259: (4)              This function has out-params. Python/C# wrappers look like:
```

```
36260: (8)                  `fz_strtof(const char *s)` => `(float, char *es)`
36261: (8)                  Locale-independent decimal to binary conversion. On overflow
36262: (8)                  return (-)INFINITY and set errno to ERANGE. On underflow return
36263: (8)                  0 and set errno to ERANGE. Special inputs (case insensitive):
36264: (8)                  "NAN", "INF" or "INFINITY".
36265: (4)              """
36266: (4)              return _mupdf.fz_strtof(s, es)
36267: (0)          def fz_structure_from_string(str):
36268: (4)              r""" Class-aware wrapper for `::fz_structure_from_string()`."""
36269: (4)              return _mupdf.fz_structure_from_string(str)
36270: (0)          def fz_structure_to_string(type):
36271: (4)              r""" Class-aware wrapper for `::fz_structure_to_string()`."""
36272: (4)              return _mupdf.fz_structure_to_string(type)
36273: (0)          def fz_subpixel_adjust(ctm, subpix_ctm, qe, qf):
36274: (4)              r"""
36275: (4)              Class-aware wrapper for `::fz_subpixel_adjust()`.
36276: (8)                  Perform subpixel quantisation and adjustment on a glyph matrix.
36277: (8)                  ctm: On entry, the desired 'ideal' transformation for a glyph.
36278: (8)                  On exit, adjusted to a (very similar) transformation quantised
36279: (8)                  for subpixel caching.
36280: (8)                  subpix_ctm: Initialised by the routine to the transform that
36281: (8)                  should be used to render the glyph.
36282: (8)                  qe, qf: which subpixel position we quantised to.
36283: (8)                  Returns: the size of the glyph.
36284: (8)                  Note: This is currently only exposed for use in our app. It
36285: (8)                  should be considered "at risk" of removal from the API.
36286: (4)              """
36287: (4)              return _mupdf.fz_subpixel_adjust(ctm, subpix_ctm, qe, qf)
36288: (0)          def fz_subsample_pixmap(tile, factor):
36289: (4)              r""" Class-aware wrapper for `::fz_subsample_pixmap()`."""
36290: (4)              return _mupdf.fz_subsample_pixmap(tile, factor)
36291: (0)          def fz_subset_cff_for_gids(orig, gids, num_gids, symbolic, cidfont):
36292: (4)              r"""
36293: (4)              Class-aware wrapper for `::fz_subset_cff_for_gids()`.
36294: (4)              This function has out-params. Python/C# wrappers look like:
36295: (8)                  `fz_subset_cff_for_gids(::fz_buffer *orig, int num_gids, int symbolic,
int cidfont)` => `(fz_buffer *, int gids)`
36296: (4)              """
36297: (4)              return _mupdf.fz_subset_cff_for_gids(orig, gids, num_gids, symbolic,
cidfont)
36298: (0)          def fz_subset_ttf_for_gids(orig, gids, num_gids, symbolic, cidfont):
36299: (4)              r"""
36300: (4)              Class-aware wrapper for `::fz_subset_ttf_for_gids()`.
36301: (4)              This function has out-params. Python/C# wrappers look like:
36302: (8)                  `fz_subset_ttf_for_gids(::fz_buffer *orig, int num_gids, int symbolic,
int cidfont)` => `(fz_buffer *, int gids)`
36303: (4)              """
36304: (4)              return _mupdf.fz_subset_ttf_for_gids(orig, gids, num_gids, symbolic,
cidfont)
36305: (0)          def fz_sync_bits(stm):
36306: (4)              r"""
36307: (4)              Class-aware wrapper for `::fz_sync_bits()`.
36308: (8)                  Called after reading bits to tell the stream
36309: (8)                  that we are about to return to reading bytewise. Resyncs
36310: (8)                  the stream to whole byte boundaries.
36311: (4)              """
36312: (4)              return _mupdf.fz_sync_bits(stm)
36313: (0)          def fz_tell(stm):
36314: (4)              r"""
36315: (4)              Class-aware wrapper for `::fz_tell()`.
36316: (8)                  return the current reading position within a stream
36317: (4)              """
36318: (4)              return _mupdf.fz_tell(stm)
36319: (0)          def fz_tell_output(out):
36320: (4)              r"""
36321: (4)              Class-aware wrapper for `::fz_tell_output()`.
36322: (8)                  Return the current file position.
36323: (8)                  Throw an error on untellable outputs.
36324: (4)              """
```

```
36325: (4)             return _mupdf.fz_tell_output(out)
36326: (0)         def fz_terminate_buffer(buf):
36327: (4)             r"""
36328: (4)             Class-aware wrapper for `::fz_terminate_buffer()`.
36329: (8)                 Zero-terminate buffer in order to use as a C string.
36330: (8)                 This byte is invisible and does not affect the length of the
36331: (8)                 buffer as returned by fz_buffer_storage. The zero byte is
36332: (8)                 written *after* the data, and subsequent writes will overwrite
36333: (8)                 the terminating byte.
36334: (8)                 Subsequent changes to the size of the buffer (such as by
36335: (8)                 fz_buffer_trim, fz_buffer_grow, fz_resize_buffer, etc) may
36336: (8)                 invalidate this.
36337: (4)             """
36338: (4)             return _mupdf.fz_terminate_buffer(buf)
36339: (0)         def fz_text_aa_level():
36340: (4)             r"""
36341: (4)             Class-aware wrapper for `::fz_text_aa_level()`.
36342: (8)                 Get the number of bits of antialiasing we are
36343: (8)                 using for text. Between 0 and 8.
36344: (4)             """
36345: (4)             return _mupdf.fz_text_aa_level()
36346: (0)         def fz_text_language_from_string(str):
36347: (4)             r"""
36348: (4)             Class-aware wrapper for `::fz_text_language_from_string()`.
36349: (8)                 Convert ISO 639 (639-{1,2,3,5}) language specification
36350: (8)                 strings losslessly to a 15 bit fz_text_language code.
36351: (8)                 No validation is carried out. Obviously invalid (out
36352: (8)                 of spec) codes will be mapped to FZ_LANG_UNSET, but
36353: (8)                 well-formed (but undefined) codes will be blithely
36354: (8)                 accepted.
36355: (4)             """
36356: (4)             return _mupdf.fz_text_language_from_string(str)
36357: (0)         def fz_tint_pixmap(pix, black, white):
36358: (4)             r"""
36359: (4)             Class-aware wrapper for `::fz_tint_pixmap()`.
36360: (8)                 Tint all the pixels in an RGB, BGR, or Gray pixmap.
36361: (8)                 black: Map black to this hexadecimal RGB color.
36362: (8)                 white: Map white to this hexadecimal RGB color.
36363: (4)             """
36364: (4)             return _mupdf.fz_tint_pixmap(pix, black, white)
36365: (0)         def fz_tolower(c):
36366: (4)             r"""
36367: (4)             Class-aware wrapper for `::fz_tolower()`.
36368: (8)                 Unicode aware tolower and toupper functions.
36369: (4)             """
36370: (4)             return _mupdf.fz_tolower(c)
36371: (0)         def fz_toupper(c):
36372: (4)             r""" Class-aware wrapper for `::fz_toupper()`."""
36373: (4)             return _mupdf.fz_toupper(c)
36374: (0)         def fz_transform_page(mediabox, resolution, rotate):
36375: (4)             r"""
36376: (4)             Class-aware wrapper for `::fz_transform_page()`.
36377: (8)                 Create transform matrix to draw page
36378: (8)                 at a given resolution and rotation. Adjusts the scaling
36379: (8)                 factors so that the page covers whole number of
36380: (8)                 pixels and adjust the page origin to be at 0,0.
36381: (4)             """
36382: (4)             return _mupdf.fz_transform_page(mediabox, resolution, rotate)
36383: (0)         def fz_transform_path(path, transform):
36384: (4)             r"""
36385: (4)             Class-aware wrapper for `::fz_transform_path()`.
36386: (8)                 Transform a path by a given
36387: (8)                 matrix.
36388: (8)                 path: The path to modify (must not be a packed path).
36389: (8)                 transform: The transform to apply.
36390: (8)                 Throws exceptions if the path is packed, or on failure
36391: (8)                 to allocate.
36392: (4)             """
36393: (4)             return _mupdf.fz_transform_path(path, transform)
```

```
(0)     def fz_transform_point(point, m):
(4)         r"""
(4)         Class-aware wrapper for `::fz_transform_point()`.
(8)             Apply a transformation to a point.
(8)             transform: Transformation matrix to apply. See fz_concat,
(8)             fz_scale, fz_rotate and fz_translate for how to create a
(8)             matrix.
(8)             point: Pointer to point to update.
(8)             Returns transform (unchanged).
(4)         """
(4)         return _mupdf.fz_transform_point(point, m)
(0)     def fz_transform_point_xy(x, y, m):
(4)         r""" Class-aware wrapper for `::fz_transform_point_xy()`."""
(4)         return _mupdf.fz_transform_point_xy(x, y, m)
(0)     def fz_transform_quad(q, m):
(4)         r"""
(4)         Class-aware wrapper for `::fz_transform_quad()`.
(8)             Transform a quad by a matrix.
(4)         """
(4)         return _mupdf.fz_transform_quad(q, m)
(0)     def fz_transform_rect(rect, m):
(4)         r"""
(4)         Class-aware wrapper for `::fz_transform_rect()`.
(8)             Apply a transform to a rectangle.
(8)             After the four corner points of the axis-aligned rectangle
(8)             have been transformed it may not longer be axis-aligned. So a
(8)             new axis-aligned rectangle is created covering at least the
(8)             area of the transformed rectangle.
(8)             transform: Transformation matrix to apply. See fz_concat,
(8)             fz_scale and fz_rotate for how to create a matrix.
(8)             rect: Rectangle to be transformed. The two special cases
(8)             fz_empty_rect and fz_infinite_rect, may be used but are
(8)             returned unchanged as expected.
(4)         """
(4)         return _mupdf.fz_transform_rect(rect, m)
(0)     def fz_transform_vector(vector, m):
(4)         r"""
(4)         Class-aware wrapper for `::fz_transform_vector()`.
(8)             Apply a transformation to a vector.
(8)             transform: Transformation matrix to apply. See fz_concat,
(8)             fz_scale and fz_rotate for how to create a matrix. Any
(8)             translation will be ignored.
(8)             vector: Pointer to vector to update.
(4)         """
(4)         return _mupdf.fz_transform_vector(vector, m)
(0)     def fz_translate(tx, ty):
(4)         r"""
(4)         Class-aware wrapper for `::fz_translate()`.
(8)             Create a translation matrix.
(8)             The returned matrix is of the form [ 1 0 0 1 tx ty ].
(8)             m: A place to store the created matrix.
(8)             tx, ty: Translation distances along the X- and Y-axes. A
(8)             translation of 0 will not cause any translation along the
(8)             relevant axis.
(8)             Returns m.
(4)         """
(4)         return _mupdf.fz_translate(tx, ty)
(0)     def fz_translate_irect(a, xoff, yoff):
(4)         r""" Class-aware wrapper for `::fz_translate_irect()`."""
(4)         return _mupdf.fz_translate_irect(a, xoff, yoff)
(0)     def fz_translate_rect(a, xoff, yoff):
(4)         r"""
(4)         Class-aware wrapper for `::fz_translate_rect()`.
(8)             Translate bounding box.
(8)             Translate a bbox by a given x and y offset. Allows for overflow.
(4)         """
(4)         return _mupdf.fz_translate_rect(a, xoff, yoff)
(0)     def fz_tree_archive_add_buffer(arch_, name, buf):
(4)         r"""
```

```
36463: (4)             Class-aware wrapper for `::fz_tree_archive_add_buffer()`.
36464: (8)                 Add a named buffer to an existing tree archive.
36465: (8)                 The tree will take a new reference to the buffer. Ownership
36466: (8)                 is not transferred.
36467: (4)             """
36468: (4)             return _mupdf.fz_tree_archive_add_buffer(arch_, name, buf)
36469: (0)         def fz_tree_archive_add_data(arch_, name, data, size):
36470: (4)             r"""
36471: (4)             Class-aware wrapper for `::fz_tree_archive_add_data()`.
36472: (8)                 Add a named block of data to an existing tree archive.
36473: (8)                 The data will be copied into a buffer, and so the caller
36474: (8)                 may free it as soon as this returns.
36475: (4)             """
36476: (4)             return _mupdf.fz_tree_archive_add_data(arch_, name, data, size)
36477: (0)         def fz_tree_lookup(node, key):
36478: (4)             r"""
36479: (4)             Class-aware wrapper for `::fz_tree_lookup()`.
36480: (8)                 Look for the value of a node in the tree with the given key.
36481: (8)                 Simple pointer equivalence is used for key.
36482: (8)                 Returns NULL for no match.
36483: (4)             """
36484: (4)             return _mupdf.fz_tree_lookup(node, key)
36485: (0)         def fz_trim_buffer(buf):
36486: (4)             r"""
36487: (4)             Class-aware wrapper for `::fz_trim_buffer()`.
36488: (8)                 Trim wasted capacity from a buffer by resizing internal memory.
36489: (4)             """
36490: (4)             return _mupdf.fz_trim_buffer(buf)
36491: (0)         def fz_trim_path(path):
36492: (4)             r"""
36493: (4)             Class-aware wrapper for `::fz_trim_path()`.
36494: (8)                 Minimise the internal storage used by a path.
36495: (8)                 As paths are constructed, the internal buffers
36496: (8)                 grow. To avoid repeated reallocations they
36497: (8)                 grow with some spare space. Once a path has
36498: (8)                 been fully constructed, this call allows the
36499: (8)                 excess space to be trimmed.
36500: (4)             """
36501: (4)             return _mupdf.fz_trim_path(path)
36502: (0)         def fz_truncate_output(arg_0):
36503: (4)             r"""
36504: (4)             Class-aware wrapper for `::fz_truncate_output()`.
36505: (8)                 Truncate the output at the current position.
36506: (8)                 This allows output streams which have seeked back from the end
36507: (8)                 of their storage to be truncated at the current point.
36508: (4)             """
36509: (4)             return _mupdf.fz_truncate_output(arg_0)
36510: (0)         def fz_try_invert_matrix(inv, src):
36511: (4)             r"""
36512: (4)             Class-aware wrapper for `::fz_try_invert_matrix()`.
36513: (8)                 Attempt to create an inverse matrix.
36514: (8)                 inverse: Place to store inverse matrix.
36515: (8)                 matrix: Matrix to invert. A degenerate matrix, where the
36516: (8)                 determinant is equal to zero, can not be inverted.
36517: (8)                 Returns 1 if matrix is degenerate (singular), or 0 otherwise.
36518: (4)             """
36519: (4)             return _mupdf.fz_try_invert_matrix(inv, src)
36520: (0)         def fz_try_open_archive_entry(arch, name):
36521: (4)             r"""
36522: (4)             Class-aware wrapper for `::fz_try_open_archive_entry()`.
36523: (8)                 Opens an archive entry as a stream.
36524: (8)                 Returns NULL if a matching entry cannot be found, otherwise
36525: (8)                 behaves exactly as fz_open_archive_entry.
36526: (4)             """
36527: (4)             return _mupdf.fz_try_open_archive_entry(arch, name)
36528: (0)         def fz_try_open_archive_with_stream(file):
36529: (4)             r"""
36530: (4)             Class-aware wrapper for `::fz_try_open_archive_with_stream()`.
36531: (8)                 Open zip or tar archive stream.
```

```
36532: (8)                  Does the same as fz_open_archive_with_stream, but will not throw
36533: (8)                  an error in the event of failing to recognise the format. Will
36534: (8)                  still throw errors in other cases though!
36535: (4)              """
36536: (4)              return _mupdf.fz_try_open_archive_with_stream(file)
36537: (0)          def fz_try_open_file(name):
36538: (4)              r"""
36539: (4)              Class-aware wrapper for `::fz_try_open_file()`.
36540: (8)                  Open the named file and wrap it in a stream.
36541: (8)                  Does the same as fz_open_file, but in the event the file
36542: (8)                  does not open, it will return NULL rather than throw an
36543: (8)                  exception.
36544: (4)              """
36545: (4)              return _mupdf.fz_try_open_file(name)
36546: (0)          def fz_try_parse_xml_archive_entry(dir, filename, preserve_white):
36547: (4)              r"""
36548: (4)              Class-aware wrapper for `::fz_try_parse_xml_archive_entry()`.
36549: (8)                  Try and parse the contents of an archive entry into a tree of xml
nodes.
36550: (8)                  preserve_white: whether to keep or delete all-whitespace nodes.
36551: (8)                  Will return NULL if the archive entry can't be found. Otherwise
behaves
36552: (8)                  the same as fz_parse_xml_archive_entry. May throw exceptions.
36553: (4)              """
36554: (4)              return _mupdf.fz_try_parse_xml_archive_entry(dir, filename,
preserve_white)
36555: (0)          def fz_try_read_archive_entry(arch, name):
36556: (4)              r"""
36557: (4)              Class-aware wrapper for `::fz_try_read_archive_entry()`.
36558: (8)                  Reads all bytes in an archive entry
36559: (8)                  into a buffer.
36560: (8)                  name: Entry name to look for, this must be an exact match to
36561: (8)                  the entry name in the archive.
36562: (8)                  Returns NULL if a matching entry cannot be found. Otherwise behaves
36563: (8)                  the same as fz_read_archive_entry. Exceptions may be thrown.
36564: (4)              """
36565: (4)              return _mupdf.fz_try_read_archive_entry(arch, name)
36566: (0)          def fz_try_read_file(filename):
36567: (4)              r"""
36568: (4)              Class-aware wrapper for `::fz_try_read_file()`.
36569: (8)                  Read all the contents of a file into a buffer.
36570: (8)                  Returns NULL if the file does not exist, otherwise
36571: (8)                  behaves exactly as fz_read_file.
36572: (4)              """
36573: (4)              return _mupdf.fz_try_read_file(filename)
36574: (0)          def fz_tune_image_decode(image_decode, arg):
36575: (4)              r"""
36576: (4)              Class-aware wrapper for `::fz_tune_image_decode()`.
36577: (8)                  Set the tuning function to use for
36578: (8)                  image decode.
36579: (8)                  image_decode: Function to use.
36580: (8)                  arg: Opaque argument to be passed to tuning function.
36581: (4)              """
36582: (4)              return _mupdf.fz_tune_image_decode(image_decode, arg)
36583: (0)          def fz_tune_image_scale(image_scale, arg):
36584: (4)              r"""
36585: (4)              Class-aware wrapper for `::fz_tune_image_scale()`.
36586: (8)                  Set the tuning function to use for
36587: (8)                  image scaling.
36588: (8)                  image_scale: Function to use.
36589: (8)                  arg: Opaque argument to be passed to tuning function.
36590: (4)              """
36591: (4)              return _mupdf.fz_tune_image_scale(image_scale, arg)
36592: (0)          def fz_unicode_from_glyph_name(name):
36593: (4)              r""" Class-aware wrapper for `::fz_unicode_from_glyph_name()`."""
36594: (4)              return _mupdf.fz_unicode_from_glyph_name(name)
36595: (0)          def fz_unicode_from_glyph_name_strict(name):
36596: (4)              r""" Class-aware wrapper for `::fz_unicode_from_glyph_name_strict()`."""
36597: (4)              return _mupdf.fz_unicode_from_glyph_name_strict(name)
```

```
36598: (0)          def fz_union_rect(a, b):
36599: (4)              r"""
36600: (4)              Class-aware wrapper for `::fz_union_rect()`.
36601: (8)                  Compute union of two rectangles.
36602: (8)                  Given two rectangles, update the first to be the smallest
36603: (8)                  axis-aligned rectangle that encompasses both given rectangles.
36604: (8)                  If either rectangle is infinite then the union is also infinite.
36605: (8)                  If either rectangle is empty then the union is simply the
36606: (8)                  non-empty rectangle. Should both rectangles be empty, then the
36607: (8)                  union is also empty.
36608: (4)              """
36609: (4)              return _mupdf.fz_union_rect(a, b)
36610: (0)          def fz_unlock(lock):
36611: (4)              r"""
36612: (4)              Class-aware wrapper for `::fz_unlock()`.
36613: (8)                  Unlock one of the user supplied mutexes.
36614: (4)              """
36615: (4)              return _mupdf.fz_unlock(lock)
36616: (0)          def fz_unpack_stream(src, depth, w, h, n, indexed, pad, skip):
36617: (4)              r""" Class-aware wrapper for `::fz_unpack_stream()`."""
36618: (4)              return _mupdf.fz_unpack_stream(src, depth, w, h, n, indexed, pad, skip)
36619: (0)          def fz_unread_byte(stm):
36620: (4)              r"""
36621: (4)              Class-aware wrapper for `::fz_unread_byte()`.
36622: (8)                  Unread the single last byte successfully
36623: (8)                  read from a stream. Do not call this without having
36624: (8)                  successfully read a byte.
36625: (8)                  stm: The stream to operate upon.
36626: (4)              """
36627: (4)              return _mupdf.fz_unread_byte(stm)
36628: (0)          def fz_unshare_stroke_state(shared):
36629: (4)              r"""
36630: (4)              Class-aware wrapper for `::fz_unshare_stroke_state()`.
36631: (8)                  Given a reference to a (possibly) shared stroke_state structure,
36632: (8)                  return a reference to an equivalent stroke_state structure
36633: (8)                  that is guaranteed to be unshared (i.e. one that can
36634: (8)                  safely be modified).
36635: (8)                  shared: The reference to a (possibly) shared structure
36636: (8)                  to unshare. Ownership of this reference is passed in
36637: (8)                  to this function, even in the case of exceptions being
36638: (8)                  thrown.
36639: (8)                  Exceptions may be thrown in the event of failure to
36640: (8)                  allocate if required.
36641: (4)              """
36642: (4)              return _mupdf.fz_unshare_stroke_state(shared)
36643: (0)          def fz_unshare_stroke_state_with_dash_len(shared, len):
36644: (4)              r"""
36645: (4)              Class-aware wrapper for `::fz_unshare_stroke_state_with_dash_len()`.
36646: (8)                  Given a reference to a (possibly) shared stroke_state structure,
36647: (8)                  return a reference to a stroke_state structure (with room for a
36648: (8)                  given amount of dash data) that is guaranteed to be unshared
36649: (8)                  (i.e. one that can safely be modified).
36650: (8)                  shared: The reference to a (possibly) shared structure
36651: (8)                  to unshare. Ownership of this reference is passed in
36652: (8)                  to this function, even in the case of exceptions being
36653: (8)                  thrown.
36654: (8)                  Exceptions may be thrown in the event of failure to
36655: (8)                  allocate if required.
36656: (4)              """
36657: (4)              return _mupdf.fz_unshare_stroke_state_with_dash_len(shared, len)
36658: (0)          def fz_urldecode(url):
36659: (4)              r"""
36660: (4)              Class-aware wrapper for `::fz_urldecode()`.
36661: (8)                  Like fz_decode_uri_component but in-place.
36662: (4)              """
36663: (4)              return _mupdf.fz_urldecode(url)
36664: (0)          def fz_use_document_css():
36665: (4)              r"""
36666: (4)              Class-aware wrapper for `::fz_use_document_css()`.
```

```
36667: (8)                Return whether to respect document styles in HTML and EPUB.
36668: (4)            """
36669: (4)            return _mupdf.fz_use_document_css()
36670: (0)        def fz_user_context():
36671: (4)            r"""
36672: (4)            Class-aware wrapper for `::fz_user_context()`.
36673: (8)                Read the user field from the context.
36674: (4)            """
36675: (4)            return _mupdf.fz_user_context()
36676: (0)        def fz_user_css():
36677: (4)            r"""
36678: (4)            Class-aware wrapper for `::fz_user_css()`.
36679: (8)                Get the user stylesheet source text.
36680: (4)            """
36681: (4)            return _mupdf.fz_user_css()
36682: (0)        def fz_utflen(s):
36683: (4)            r"""
36684: (4)            Class-aware wrapper for `::fz_utflen()`.
36685: (8)                Count how many runes the UTF-8 encoded string
36686: (8)                consists of.
36687: (8)                s: The UTF-8 encoded, NUL-terminated text string.
36688: (8)                Returns the number of runes in the string.
36689: (4)            """
36690: (4)            return _mupdf.fz_utflen(s)
36691: (0)        def fz_var_imp(arg_0):
36692: (4)            r""" Class-aware wrapper for `::fz_var_imp()`."""
36693: (4)            return _mupdf.fz_var_imp(arg_0)
36694: (0)        def fz_walk_path(path, walker, arg):
36695: (4)            r"""
36696: (4)            Class-aware wrapper for `::fz_walk_path()`.
36697: (8)                Walk the segments of a path, calling the
36698: (8)                appropriate callback function from a given set for each
36699: (8)                segment of the path.
36700: (8)                path: The path to walk.
36701: (8)                walker: The set of callback functions to use. The first
36702: (8)                4 callback pointers in the set must be non-NULL. The
36703: (8)                subsequent ones can either be supplied, or can be left
36704: (8)                as NULL, in which case the top 4 functions will be
36705: (8)                called as appropriate to simulate them.
36706: (8)                arg: An opaque argument passed in to each callback.
36707: (8)                Exceptions will only be thrown if the underlying callback
36708: (8)                functions throw them.
36709: (4)            """
36710: (4)            return _mupdf.fz_walk_path(path, walker, arg)
36711: (0)        def fz_warn(*args):
36712: (4)            r""" Class-aware wrapper for `::fz_warn()`."""
36713: (4)            return _mupdf.fz_warn(*args)
36714: (0)        def fz_warning_callback(user):
36715: (4)            r"""
36716: (4)            Class-aware wrapper for `::fz_warning_callback()`.
36717: (4)            This function has out-params. Python/C# wrappers look like:
36718: (8)                `fz_warning_callback()` => `(fz_warning_cb *, void *user)`
36719: (8)                Retrieve the currently set warning callback, or NULL if none
36720: (8)                has been set. Optionally, if user is non-NULL, the user pointer
36721: (8)                given when the warning callback was set is also passed back to
36722: (8)                the caller.
36723: (4)            """
36724: (4)            return _mupdf.fz_warning_callback(user)
36725: (0)        def fz_warp_pixmap(src, points, width, height):
36726: (4)            r""" Class-aware wrapper for `::fz_warp_pixmap()`."""
36727: (4)            return _mupdf.fz_warp_pixmap(src, points, width, height)
36728: (0)        def fz_windows_1250_from_unicode(u):
36729: (4)            r""" Class-aware wrapper for `::fz_windows_1250_from_unicode()`."""
36730: (4)            return _mupdf.fz_windows_1250_from_unicode(u)
36731: (0)        def fz_windows_1251_from_unicode(u):
36732: (4)            r""" Class-aware wrapper for `::fz_windows_1251_from_unicode()`."""
36733: (4)            return _mupdf.fz_windows_1251_from_unicode(u)
36734: (0)        def fz_windows_1252_from_unicode(u):
36735: (4)            r""" Class-aware wrapper for `::fz_windows_1252_from_unicode()`."""
```

```
36736: (4)             return _mupdf.fz_windows_1252_from_unicode(u)
36737: (0)         def fz_write_band(writer, stride, band_height, samples):
36738: (4)             r"""
36739: (4)             Class-aware wrapper for `::fz_write_band()`.
36740: (8)                 Cause a band writer to write the next band
36741: (8)                 of data for an image.
36742: (8)                 stride: The byte offset from the first byte of the data
36743: (8)                 for a pixel to the first byte of the data for the same pixel
36744: (8)                 on the row below.
36745: (8)                 band_height: The number of lines in this band.
36746: (8)                 samples: Pointer to first byte of the data.
36747: (4)             """
36748: (4)             return _mupdf.fz_write_band(writer, stride, band_height, samples)
36749: (0)         def fz_write_base64(out, data, size, newline):
36750: (4)             r"""
36751: (4)             Class-aware wrapper for `::fz_write_base64()`.
36752: (8)                 Write a base64 encoded data block, optionally with periodic
36753: (8)                 newlines.
36754: (4)             """
36755: (4)             return _mupdf.fz_write_base64(out, data, size, newline)
36756: (0)         def fz_write_base64_buffer(out, data, newline):
36757: (4)             r"""
36758: (4)             Class-aware wrapper for `::fz_write_base64_buffer()`.
36759: (8)                 Write a base64 encoded fz_buffer, optionally with periodic
36760: (8)                 newlines.
36761: (4)             """
36762: (4)             return _mupdf.fz_write_base64_buffer(out, data, newline)
36763: (0)         def fz_write_bitmap_as_pbm(out, bitmap):
36764: (4)             r"""
36765: (4)             Class-aware wrapper for `::fz_write_bitmap_as_pbm()`.
36766: (8)                 Write a bitmap as a pbm.
36767: (4)             """
36768: (4)             return _mupdf.fz_write_bitmap_as_pbm(out, bitmap)
36769: (0)         def fz_write_bitmap_as_pcl(out, bitmap, pcl):
36770: (4)             r"""
36771: (4)             Class-aware wrapper for `::fz_write_bitmap_as_pcl()`.
36772: (8)                 Write a bitmap as mono PCL.
36773: (4)             """
36774: (4)             return _mupdf.fz_write_bitmap_as_pcl(out, bitmap, pcl)
36775: (0)         def fz_write_bitmap_as_pkm(out, bitmap):
36776: (4)             r"""
36777: (4)             Class-aware wrapper for `::fz_write_bitmap_as_pkm()`.
36778: (8)                 Write a CMYK bitmap as a pkm.
36779: (4)             """
36780: (4)             return _mupdf.fz_write_bitmap_as_pkm(out, bitmap)
36781: (0)         def fz_write_bitmap_as_pwg(out, bitmap, pwg):
36782: (4)             r"""
36783: (4)             Class-aware wrapper for `::fz_write_bitmap_as_pwg()`.
36784: (8)                 Write a bitmap as a PWG.
36785: (4)             """
36786: (4)             return _mupdf.fz_write_bitmap_as_pwg(out, bitmap, pwg)
36787: (0)         def fz_write_bitmap_as_pwg_page(out, bitmap, pwg):
36788: (4)             r"""
36789: (4)             Class-aware wrapper for `::fz_write_bitmap_as_pwg_page()`.
36790: (8)                 Write a bitmap as a PWG page.
36791: (8)                 Caller should provide a file header by calling
36792: (8)                 fz_write_pwg_file_header, but can then write several pages to
36793: (8)                 the same file.
36794: (4)             """
36795: (4)             return _mupdf.fz_write_bitmap_as_pwg_page(out, bitmap, pwg)
36796: (0)         def fz_write_bits(out, data, num_bits):
36797: (4)             r"""
36798: (4)             Class-aware wrapper for `::fz_write_bits()`.
36799: (8)                 Write num_bits of data to the end of the output stream, assumed to be
packed
36800: (8)                 most significant bits first.
36801: (4)             """
36802: (4)             return _mupdf.fz_write_bits(out, data, num_bits)
36803: (0)         def fz_write_bits_sync(out):
```

```
36804: (4)             r"""
36805: (4)             Class-aware wrapper for `::fz_write_bits_sync()`.
36806: (8)                 Sync to byte boundary after writing bits.
36807: (4)             """
36808: (4)             return _mupdf.fz_write_bits_sync(out)
36809: (0)         def fz_write_buffer(out, data):
36810: (4)             r""" Class-aware wrapper for `::fz_write_buffer()`."""
36811: (4)             return _mupdf.fz_write_buffer(out, data)
36812: (0)         def fz_write_byte(out, x):
36813: (4)             r""" Class-aware wrapper for `::fz_write_byte()`."""
36814: (4)             return _mupdf.fz_write_byte(out, x)
36815: (0)         def fz_write_char(out, x):
36816: (4)             r""" Class-aware wrapper for `::fz_write_char()`."""
36817: (4)             return _mupdf.fz_write_char(out, x)
36818: (0)         def fz_write_data(out, data, size):
36819: (4)             r"""
36820: (4)             Class-aware wrapper for `::fz_write_data()`.
36821: (8)                 Write data to output.
36822: (8)                 data: Pointer to data to write.
36823: (8)                 size: Size of data to write in bytes.
36824: (4)             """
36825: (4)             return _mupdf.fz_write_data(out, data, size)
36826: (0)         def fz_write_document(wri, doc):
36827: (4)             r"""
36828: (4)             Class-aware wrapper for `::fz_write_document()`.
36829: (8)                 Convenience function to feed all the pages of a document to
36830: (8)                 fz_begin_page/fz_run_page/fz_end_page.
36831: (4)             """
36832: (4)             return _mupdf.fz_write_document(wri, doc)
36833: (0)         def fz_write_float_be(out, f):
36834: (4)             r""" Class-aware wrapper for `::fz_write_float_be()`."""
36835: (4)             return _mupdf.fz_write_float_be(out, f)
36836: (0)         def fz_write_float_le(out, f):
36837: (4)             r""" Class-aware wrapper for `::fz_write_float_le()`."""
36838: (4)             return _mupdf.fz_write_float_le(out, f)
36839: (0)         def fz_write_header(writer, w, h, n, alpha, xres, yres, pagenum, cs, seps):
36840: (4)             r"""
36841: (4)             Class-aware wrapper for `::fz_write_header()`.
36842: (8)                 Cause a band writer to write the header for
36843: (8)                 a banded image with the given properties/dimensions etc. This
36844: (8)                 also configures the bandwriter for the format of the data to be
36845: (8)                 passed in future calls.
36846: (8)                 w, h: Width and Height of the entire page.
36847: (8)                 n: Number of components (including spots and alphas).
36848: (8)                 alpha: Number of alpha components.
36849: (8)                 xres, yres: X and Y resolutions in dpi.
36850: (8)                 cs: Colorspace (NULL for bitmaps)
36851: (8)                 seps: Separation details (or NULL).
36852: (4)             """
36853: (4)             return _mupdf.fz_write_header(writer, w, h, n, alpha, xres, yres, pagenum,
cs, seps)
36854: (0)         def fz_write_image_as_data_uri(out, image):
36855: (4)             r"""
36856: (4)             Class-aware wrapper for `::fz_write_image_as_data_uri()`.
36857: (8)                 Write image as a data URI (for HTML and SVG output).
36858: (4)             """
36859: (4)             return _mupdf.fz_write_image_as_data_uri(out, image)
36860: (0)         def fz_write_int16_be(out, x):
36861: (4)             r""" Class-aware wrapper for `::fz_write_int16_be()`."""
36862: (4)             return _mupdf.fz_write_int16_be(out, x)
36863: (0)         def fz_write_int16_le(out, x):
36864: (4)             r""" Class-aware wrapper for `::fz_write_int16_le()`."""
36865: (4)             return _mupdf.fz_write_int16_le(out, x)
36866: (0)         def fz_write_int32_be(out, x):
36867: (4)             r"""
36868: (4)             Class-aware wrapper for `::fz_write_int32_be()`.
36869: (8)                 Write different sized data to an output stream.
36870: (4)             """
36871: (4)             return _mupdf.fz_write_int32_be(out, x)
```

```
36872: (0)          def fz_write_int32_le(out, x):
36873: (4)              r""" Class-aware wrapper for `::fz_write_int32_le()`."""
36874: (4)              return _mupdf.fz_write_int32_le(out, x)
36875: (0)          def fz_write_pixmap_as_data_uri(out, pixmap):
36876: (4)              r""" Class-aware wrapper for `::fz_write_pixmap_as_data_uri()`."""
36877: (4)              return _mupdf.fz_write_pixmap_as_data_uri(out, pixmap)
36878: (0)          def fz_write_pixmap_as_jpeg(out, pix, quality, invert_cmyk):
36879: (4)              r"""
36880: (4)              Class-aware wrapper for `::fz_write_pixmap_as_jpeg()`.
36881: (8)                  Write a pixmap as a JPEG.
36882: (4)              """
36883: (4)              return _mupdf.fz_write_pixmap_as_jpeg(out, pix, quality, invert_cmyk)
36884: (0)          def fz_write_pixmap_as_jpx(out, pix, quality):
36885: (4)              r"""
36886: (4)              Class-aware wrapper for `::fz_write_pixmap_as_jpx()`.
36887: (8)                  Pixmap data as JP2K with no subsampling.
36888: (8)                  quality = 100 = lossless
36889: (8)                  otherwise for a factor of x compression use 100-x. (so 80 is 1:20
compression)
36890: (4)              """
36891: (4)              return _mupdf.fz_write_pixmap_as_jpx(out, pix, quality)
36892: (0)          def fz_write_pixmap_as_pam(out, pixmap):
36893: (4)              r"""
36894: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pam()`.
36895: (8)                  Write a pixmap as a pnm (greyscale, rgb or cmyk, with or without
36896: (8)                  alpha).
36897: (4)              """
36898: (4)              return _mupdf.fz_write_pixmap_as_pam(out, pixmap)
36899: (0)          def fz_write_pixmap_as_pcl(out, pixmap, pcl):
36900: (4)              r"""
36901: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pcl()`.
36902: (8)                  Write an (RGB) pixmap as color PCL.
36903: (4)              """
36904: (4)              return _mupdf.fz_write_pixmap_as_pcl(out, pixmap, pcl)
36905: (0)          def fz_write_pixmap_as_pclm(out, pixmap, options):
36906: (4)              r"""
36907: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pclm()`.
36908: (8)                  Write a (Greyscale or RGB) pixmap as pclm.
36909: (4)              """
36910: (4)              return _mupdf.fz_write_pixmap_as_pclm(out, pixmap, options)
36911: (0)          def fz_write_pixmap_as_pdfocr(out, pixmap, options):
36912: (4)              r"""
36913: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pdfocr()`.
36914: (8)                  Write a (Greyscale or RGB) pixmap as pdfocr.
36915: (4)              """
36916: (4)              return _mupdf.fz_write_pixmap_as_pdfocr(out, pixmap, options)
36917: (0)          def fz_write_pixmap_as_png(out, pixmap):
36918: (4)              r"""
36919: (4)              Class-aware wrapper for `::fz_write_pixmap_as_png()`.
36920: (8)                  Write a (Greyscale or RGB) pixmap as a png.
36921: (4)              """
36922: (4)              return _mupdf.fz_write_pixmap_as_png(out, pixmap)
36923: (0)          def fz_write_pixmap_as_pnm(out, pixmap):
36924: (4)              r"""
36925: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pnm()`.
36926: (8)                  Write a pixmap as a pnm (greyscale or rgb, no alpha).
36927: (4)              """
36928: (4)              return _mupdf.fz_write_pixmap_as_pnm(out, pixmap)
36929: (0)          def fz_write_pixmap_as_ps(out, pixmap):
36930: (4)              r"""
36931: (4)              Class-aware wrapper for `::fz_write_pixmap_as_ps()`.
36932: (8)                  Write a (gray, rgb, or cmyk, no alpha) pixmap out as postscript.
36933: (4)              """
36934: (4)              return _mupdf.fz_write_pixmap_as_ps(out, pixmap)
36935: (0)          def fz_write_pixmap_as_psd(out, pixmap):
36936: (4)              r"""
36937: (4)              Class-aware wrapper for `::fz_write_pixmap_as_psd()`.
36938: (8)                  Write a pixmap as a PSD file.
36939: (4)              """
```

```
36940: (4)              return _mupdf.fz_write_pixmap_as_psd(out, pixmap)
36941: (0)          def fz_write_pixmap_as_pwg(out, pixmap, pwg):
36942: (4)              r"""
36943: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pwg()`.
36944: (8)                  Write a pixmap as a PWG.
36945: (4)              """
36946: (4)              return _mupdf.fz_write_pixmap_as_pwg(out, pixmap, pwg)
36947: (0)          def fz_write_pixmap_as_pwg_page(out, pixmap, pwg):
36948: (4)              r"""
36949: (4)              Class-aware wrapper for `::fz_write_pixmap_as_pwg_page()`.
36950: (8)                  Write a pixmap as a PWG page.
36951: (8)                  Caller should provide a file header by calling
36952: (8)                  fz_write_pwg_file_header, but can then write several pages to
36953: (8)                  the same file.
36954: (4)              """
36955: (4)              return _mupdf.fz_write_pixmap_as_pwg_page(out, pixmap, pwg)
36956: (0)          def fz_write_ps_file_header(out):
36957: (4)              r"""
36958: (4)              Class-aware wrapper for `::fz_write_ps_file_header()`.
36959: (8)                  Write the file level header for ps band writer output.
36960: (4)              """
36961: (4)              return _mupdf.fz_write_ps_file_header(out)
36962: (0)          def fz_write_ps_file_trailer(out, pages):
36963: (4)              r"""
36964: (4)              Class-aware wrapper for `::fz_write_ps_file_trailer()`.
36965: (8)                  Write the file level trailer for ps band writer output.
36966: (4)              """
36967: (4)              return _mupdf.fz_write_ps_file_trailer(out, pages)
36968: (0)          def fz_write_pwg_file_header(out):
36969: (4)              r"""
36970: (4)              Class-aware wrapper for `::fz_write_pwg_file_header()`.
36971: (8)                  Output the file header to a pwg stream, ready for pages to follow it.
36972: (4)              """
36973: (4)              return _mupdf.fz_write_pwg_file_header(out)
36974: (0)          def fz_write_rune(out, rune):
36975: (4)              r"""
36976: (4)              Class-aware wrapper for `::fz_write_rune()`.
36977: (8)                  Write a UTF-8 encoded unicode character.
36978: (4)              """
36979: (4)              return _mupdf.fz_write_rune(out, rune)
36980: (0)          def fz_write_stabilized_story(writer, user_css, em, contentfn, contentfn_ref,
rectfn, rectfn_ref, pagefn, pagefn_ref, dir):
36981: (4)              r""" Class-aware wrapper for `::fz_write_stabilized_story()`."""
36982: (4)              return _mupdf.fz_write_stabilized_story(writer, user_css, em, contentfn,
contentfn_ref, rectfn, rectfn_ref, pagefn, pagefn_ref, dir)
36983: (0)          def fz_write_story(writer, story, rectfn, rectfn_ref, positionfn,
positionfn_ref, pagefn, pagefn_ref):
36984: (4)              r""" Class-aware wrapper for `::fz_write_story()`."""
36985: (4)              return _mupdf.fz_write_story(writer, story, rectfn, rectfn_ref,
positionfn, positionfn_ref, pagefn, pagefn_ref)
36986: (0)          def fz_write_stream(out, _in):
36987: (4)              r"""
36988: (4)              Class-aware wrapper for `::fz_write_stream()`.
36989: (8)                  Copy the stream contents to the output.
36990: (4)              """
36991: (4)              return _mupdf.fz_write_stream(out, _in)
36992: (0)          def fz_write_string(out, s):
36993: (4)              r"""
36994: (4)              Class-aware wrapper for `::fz_write_string()`.
36995: (8)                  Write a string. Does not write zero terminator.
36996: (4)              """
36997: (4)              return _mupdf.fz_write_string(out, s)
36998: (0)          def fz_write_uint16_be(out, x):
36999: (4)              r""" Class-aware wrapper for `::fz_write_uint16_be()`."""
37000: (4)              return _mupdf.fz_write_uint16_be(out, x)
37001: (0)          def fz_write_uint16_le(out, x):
37002: (4)              r""" Class-aware wrapper for `::fz_write_uint16_le()`."""
37003: (4)              return _mupdf.fz_write_uint16_le(out, x)
37004: (0)          def fz_write_uint32_be(out, x):
```

```
(4)             r""" Class-aware wrapper for `::fz_write_uint32_be()`."""
(4)             return _mupdf.fz_write_uint32_be(out, x)
(0)         def fz_write_uint32_le(out, x):
(4)             r""" Class-aware wrapper for `::fz_write_uint32_le()`."""
(4)             return _mupdf.fz_write_uint32_le(out, x)
(0)         def fz_write_zip_entry(zip, name, buf, compress):
(4)             r"""
(4)             Class-aware wrapper for `::fz_write_zip_entry()`.
(8)                 Given a buffer of data, (optionally) compress it, and add it to
(8)                 the zip file with the given name.
(4)             """
(4)             return _mupdf.fz_write_zip_entry(zip, name, buf, compress)
(0)         def fz_xml_add_att(pool, node, key, val):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_add_att()`.
(8)                 Add an attribute to an XML node.
(4)             """
(4)             return _mupdf.fz_xml_add_att(pool, node, key, val)
(0)         def fz_xml_att(item, att):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_att()`.
(8)                 Return the value of an attribute of an XML node.
(8)                 NULL if the attribute doesn't exist.
(4)             """
(4)             return _mupdf.fz_xml_att(item, att)
(0)         def fz_xml_att_alt(item, one, two):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_att_alt()`.
(8)                 Return the value of an attribute of an XML node.
(8)                 If the first attribute doesn't exist, try the second.
(8)                 NULL if neither attribute exists.
(4)             """
(4)             return _mupdf.fz_xml_att_alt(item, one, two)
(0)         def fz_xml_att_eq(item, name, match):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_att_eq()`.
(8)                 Check for a matching attribute on an XML node.
(8)                 If the node has the requested attribute (name), and the value
(8)                 matches (match) then return 1. Otherwise, 0.
(4)             """
(4)             return _mupdf.fz_xml_att_eq(item, name, match)
(0)         def fz_xml_down(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_down()`.
(8)                 Return first child of XML node.
(4)             """
(4)             return _mupdf.fz_xml_down(item)
(0)         def fz_xml_find(item, tag):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_find()`.
(8)                 Search the siblings of XML nodes starting with item looking for
(8)                 the first with the given tag.
(8)                 Return NULL if none found.
(4)             """
(4)             return _mupdf.fz_xml_find(item, tag)
(0)         def fz_xml_find_dfs(item, tag, att, match):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_find_dfs()`.
(8)                 Perform a depth first search from item, returning the first
(8)                 child that matches the given tag (or any tag if tag is NULL),
(8)                 with the given attribute (if att is non NULL), that matches
(8)                 match (if match is non NULL).
(4)             """
(4)             return _mupdf.fz_xml_find_dfs(item, tag, att, match)
(0)         def fz_xml_find_dfs_top(item, tag, att, match, top):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_find_dfs_top()`.
(8)                 Perform a depth first search from item, returning the first
(8)                 child that matches the given tag (or any tag if tag is NULL),
```

```
37074: (8)                 with the given attribute (if att is non NULL), that matches
37075: (8)                 match (if match is non NULL). The search stops if it ever
37076: (8)                 reaches the top of the tree, or the declared 'top' item.
37077: (4)             """
37078: (4)             return _mupdf.fz_xml_find_dfs_top(item, tag, att, match, top)
37079: (0)         def fz_xml_find_down(item, tag):
37080: (4)             r"""
37081: (4)             Class-aware wrapper for `::fz_xml_find_down()`.
37082: (8)                 Search the siblings of XML nodes starting with the first child
37083: (8)                 of item looking for the first with the given tag.
37084: (8)                 Return NULL if none found.
37085: (4)             """
37086: (4)             return _mupdf.fz_xml_find_down(item, tag)
37087: (0)         def fz_xml_find_down_match(item, tag, att, match):
37088: (4)             r"""
37089: (4)             Class-aware wrapper for `::fz_xml_find_down_match()`.
37090: (8)                 Search the siblings of XML nodes starting with the first child
37091: (8)                 of item looking for the first with the given tag (or any tag if
37092: (8)                 tag is NULL), and with a matching attribute.
37093: (8)                 Return NULL if none found.
37094: (4)             """
37095: (4)             return _mupdf.fz_xml_find_down_match(item, tag, att, match)
37096: (0)         def fz_xml_find_match(item, tag, att, match):
37097: (4)             r"""
37098: (4)             Class-aware wrapper for `::fz_xml_find_match()`.
37099: (8)                 Search the siblings of XML nodes starting with item looking for
37100: (8)                 the first with the given tag (or any tag if tag is NULL), and
37101: (8)                 with a matching attribute.
37102: (8)                 Return NULL if none found.
37103: (4)             """
37104: (4)             return _mupdf.fz_xml_find_match(item, tag, att, match)
37105: (0)         def fz_xml_find_next(item, tag):
37106: (4)             r"""
37107: (4)             Class-aware wrapper for `::fz_xml_find_next()`.
37108: (8)                 Search the siblings of XML nodes starting with the first sibling
37109: (8)                 of item looking for the first with the given tag.
37110: (8)                 Return NULL if none found.
37111: (4)             """
37112: (4)             return _mupdf.fz_xml_find_next(item, tag)
37113: (0)         def fz_xml_find_next_dfs(item, tag, att, match):
37114: (4)             r"""
37115: (4)             Class-aware wrapper for `::fz_xml_find_next_dfs()`.
37116: (8)                 Perform a depth first search onwards from item, returning the first
37117: (8)                 child that matches the given tag (or any tag if tag is NULL),
37118: (8)                 with the given attribute (if att is non NULL), that matches
37119: (8)                 match (if match is non NULL).
37120: (4)             """
37121: (4)             return _mupdf.fz_xml_find_next_dfs(item, tag, att, match)
37122: (0)         def fz_xml_find_next_dfs_top(item, tag, att, match, top):
37123: (4)             r"""
37124: (4)             Class-aware wrapper for `::fz_xml_find_next_dfs_top()`.
37125: (8)                 Perform a depth first search onwards from item, returning the first
37126: (8)                 child that matches the given tag (or any tag if tag is NULL),
37127: (8)                 with the given attribute (if att is non NULL), that matches
37128: (8)                 match (if match is non NULL). The search stops if it ever reaches
37129: (8)                 the top of the tree, or the declared 'top' item.
37130: (4)             """
37131: (4)             return _mupdf.fz_xml_find_next_dfs_top(item, tag, att, match, top)
37132: (0)         def fz_xml_find_next_match(item, tag, att, match):
37133: (4)             r"""
37134: (4)             Class-aware wrapper for `::fz_xml_find_next_match()`.
37135: (8)                 Search the siblings of XML nodes starting with the first sibling
37136: (8)                 of item looking for the first with the given tag (or any tag if tag
37137: (8)                 is NULL), and with a matching attribute.
37138: (8)                 Return NULL if none found.
37139: (4)             """
37140: (4)             return _mupdf.fz_xml_find_next_match(item, tag, att, match)
37141: (0)         def fz_xml_is_tag(item, name):
37142: (4)             r"""
```

```
(4)             Class-aware wrapper for `::fz_xml_is_tag()`.
(8)                 Return true if the tag name matches.
(4)             """
(4)             return _mupdf.fz_xml_is_tag(item, name)
(0)         def fz_xml_next(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_next()`.
(8)                 Return next sibling of XML node.
(4)             """
(4)             return _mupdf.fz_xml_next(item)
(0)         def fz_xml_prev(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_prev()`.
(8)                 Return previous sibling of XML node.
(4)             """
(4)             return _mupdf.fz_xml_prev(item)
(0)         def fz_xml_root(xml):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_root()`.
(8)                 Return the topmost XML node of a document.
(4)             """
(4)             return _mupdf.fz_xml_root(xml)
(0)         def fz_xml_tag(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_tag()`.
(8)                 Return tag of XML node. Return NULL for text nodes.
(4)             """
(4)             return _mupdf.fz_xml_tag(item)
(0)         def fz_xml_text(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_text()`.
(8)                 Return the text content of an XML node.
(8)                 Return NULL if the node is a tag.
(4)             """
(4)             return _mupdf.fz_xml_text(item)
(0)         def fz_xml_up(item):
(4)             r"""
(4)             Class-aware wrapper for `::fz_xml_up()`.
(8)                 Return parent of XML node.
(4)             """
(4)             return _mupdf.fz_xml_up(item)
(0)         def pdf_abandon_operation(doc):
(4)             r""" Class-aware wrapper for `::pdf_abandon_operation()`."""
(4)             return _mupdf.pdf_abandon_operation(doc)
(0)         def pdf_access_exec_menu_item_event(evt):
(4)             r""" Class-aware wrapper for `::pdf_access_exec_menu_item_event()`."""
(4)             return _mupdf.pdf_access_exec_menu_item_event(evt)
(0)         def pdf_add_annot_border_dash_item(annot, length):
(4)             r""" Class-aware wrapper for `::pdf_add_annot_border_dash_item()`."""
(4)             return _mupdf.pdf_add_annot_border_dash_item(annot, length)
(0)         def pdf_add_annot_ink_list(annot, n, stroke):
(4)             r""" Class-aware wrapper for `::pdf_add_annot_ink_list()`."""
(4)             return _mupdf.pdf_add_annot_ink_list(annot, n, stroke)
(0)         def pdf_add_annot_ink_list_stroke(annot):
(4)             r""" Class-aware wrapper for `::pdf_add_annot_ink_list_stroke()`."""
(4)             return _mupdf.pdf_add_annot_ink_list_stroke(annot)
(0)         def pdf_add_annot_ink_list_stroke_vertex(annot, p):
(4)             r""" Class-aware wrapper for
`::pdf_add_annot_ink_list_stroke_vertex()`."""
(4)             return _mupdf.pdf_add_annot_ink_list_stroke_vertex(annot, p)
(0)         def pdf_add_annot_quad_point(annot, quad):
(4)             r""" Class-aware wrapper for `::pdf_add_annot_quad_point()`."""
(4)             return _mupdf.pdf_add_annot_quad_point(annot, quad)
(0)         def pdf_add_annot_vertex(annot, p):
(4)             r""" Class-aware wrapper for `::pdf_add_annot_vertex()`."""
(4)             return _mupdf.pdf_add_annot_vertex(annot, p)
(0)         def pdf_add_cid_font(doc, font):
(4)             r""" Class-aware wrapper for `::pdf_add_cid_font()`."""
(4)             return _mupdf.pdf_add_cid_font(doc, font)
```

```
37211: (0)          def pdf_add_cjk_font(doc, font, script, wmode, serif):
37212: (4)              r""" Class-aware wrapper for `::pdf_add_cjk_font()`."""
37213: (4)              return _mupdf.pdf_add_cjk_font(doc, font, script, wmode, serif)
37214: (0)          def pdf_add_codespace(cmap, low, high, n):
37215: (4)              r""" Class-aware wrapper for `::pdf_add_codespace()`."""
37216: (4)              return _mupdf.pdf_add_codespace(cmap, low, high, n)
37217: (0)          def pdf_add_embedded_file(doc, filename, mimetype, contents, created, modifed,
add_checksum):
37218: (4)              r""" Class-aware wrapper for `::pdf_add_embedded_file()`."""
37219: (4)              return _mupdf.pdf_add_embedded_file(doc, filename, mimetype, contents,
created, modifed, add_checksum)
37220: (0)          def pdf_add_hmtx(font, lo, hi, w):
37221: (4)              r""" Class-aware wrapper for `::pdf_add_hmtx()`."""
37222: (4)              return _mupdf.pdf_add_hmtx(font, lo, hi, w)
37223: (0)          def pdf_add_image(doc, image):
37224: (4)              r""" Class-aware wrapper for `::pdf_add_image()`."""
37225: (4)              return _mupdf.pdf_add_image(doc, image)
37226: (0)          def pdf_add_journal_fragment(doc, parent, copy, copy_stream, newobj):
37227: (4)              r""" Class-aware wrapper for `::pdf_add_journal_fragment()`."""
37228: (4)              return _mupdf.pdf_add_journal_fragment(doc, parent, copy, copy_stream,
newobj)
37229: (0)          def pdf_add_new_array(doc, initial):
37230: (4)              r""" Class-aware wrapper for `::pdf_add_new_array()`."""
37231: (4)              return _mupdf.pdf_add_new_array(doc, initial)
37232: (0)          def pdf_add_new_dict(doc, initial):
37233: (4)              r""" Class-aware wrapper for `::pdf_add_new_dict()`."""
37234: (4)              return _mupdf.pdf_add_new_dict(doc, initial)
37235: (0)          def pdf_add_object(doc, obj):
37236: (4)              r""" Class-aware wrapper for `::pdf_add_object()`."""
37237: (4)              return _mupdf.pdf_add_object(doc, obj)
37238: (0)          def pdf_add_page(doc, mediabox, rotate, resources, contents):
37239: (4)              r""" Class-aware wrapper for `::pdf_add_page()`."""
37240: (4)              return _mupdf.pdf_add_page(doc, mediabox, rotate, resources, contents)
37241: (0)          def pdf_add_simple_font(doc, font, encoding):
37242: (4)              r""" Class-aware wrapper for `::pdf_add_simple_font()`."""
37243: (4)              return _mupdf.pdf_add_simple_font(doc, font, encoding)
37244: (0)          def pdf_add_stream(doc, buf, obj, compressed):
37245: (4)              r""" Class-aware wrapper for `::pdf_add_stream()`."""
37246: (4)              return _mupdf.pdf_add_stream(doc, buf, obj, compressed)
37247: (0)          def pdf_add_substitute_font(doc, font):
37248: (4)              r""" Class-aware wrapper for `::pdf_add_substitute_font()`."""
37249: (4)              return _mupdf.pdf_add_substitute_font(doc, font)
37250: (0)          def pdf_add_vmtx(font, lo, hi, x, y, w):
37251: (4)              r""" Class-aware wrapper for `::pdf_add_vmtx()`."""
37252: (4)              return _mupdf.pdf_add_vmtx(font, lo, hi, x, y, w)
37253: (0)          def pdf_annot_MK_BC(annot, n, color):
37254: (4)              r"""
37255: (4)              Class-aware wrapper for `::pdf_annot_MK_BC()`.
37256: (4)              This function has out-params. Python/C# wrappers look like:
37257: (8)                  `pdf_annot_MK_BC(::pdf_annot *annot, float color[4])` => int n
37258: (4)              """
37259: (4)              return _mupdf.pdf_annot_MK_BC(annot, n, color)
37260: (0)          def pdf_annot_MK_BC_rgb(annot, rgb):
37261: (4)              r""" Class-aware wrapper for `::pdf_annot_MK_BC_rgb()`."""
37262: (4)              return _mupdf.pdf_annot_MK_BC_rgb(annot, rgb)
37263: (0)          def pdf_annot_MK_BG(annot, n, color):
37264: (4)              r"""
37265: (4)              Class-aware wrapper for `::pdf_annot_MK_BG()`.
37266: (4)              This function has out-params. Python/C# wrappers look like:
37267: (8)                  `pdf_annot_MK_BG(::pdf_annot *annot, float color[4])` => int n
37268: (4)              """
37269: (4)              return _mupdf.pdf_annot_MK_BG(annot, n, color)
37270: (0)          def pdf_annot_MK_BG_rgb(annot, rgb):
37271: (4)              r""" Class-aware wrapper for `::pdf_annot_MK_BG_rgb()`."""
37272: (4)              return _mupdf.pdf_annot_MK_BG_rgb(annot, rgb)
37273: (0)          def pdf_annot_active(annot):
37274: (4)              r""" Class-aware wrapper for `::pdf_annot_active()`."""
37275: (4)              return _mupdf.pdf_annot_active(annot)
37276: (0)          def pdf_annot_ap(annot):
```

```
37277: (4)             r""" Class-aware wrapper for `::pdf_annot_ap()`."""
37278: (4)             return _mupdf.pdf_annot_ap(annot)
37279: (0)         def pdf_annot_author(annot):
37280: (4)             r""" Class-aware wrapper for `::pdf_annot_author()`."""
37281: (4)             return _mupdf.pdf_annot_author(annot)
37282: (0)         def pdf_annot_border(annot):
37283: (4)             r""" Class-aware wrapper for `::pdf_annot_border()`."""
37284: (4)             return _mupdf.pdf_annot_border(annot)
37285: (0)         def pdf_annot_border_dash_count(annot):
37286: (4)             r""" Class-aware wrapper for `::pdf_annot_border_dash_count()`."""
37287: (4)             return _mupdf.pdf_annot_border_dash_count(annot)
37288: (0)         def pdf_annot_border_dash_item(annot, i):
37289: (4)             r""" Class-aware wrapper for `::pdf_annot_border_dash_item()`."""
37290: (4)             return _mupdf.pdf_annot_border_dash_item(annot, i)
37291: (0)         def pdf_annot_border_effect(annot):
37292: (4)             r""" Class-aware wrapper for `::pdf_annot_border_effect()`."""
37293: (4)             return _mupdf.pdf_annot_border_effect(annot)
37294: (0)         def pdf_annot_border_effect_intensity(annot):
37295: (4)             r""" Class-aware wrapper for `::pdf_annot_border_effect_intensity()`."""
37296: (4)             return _mupdf.pdf_annot_border_effect_intensity(annot)
37297: (0)         def pdf_annot_border_style(annot):
37298: (4)             r""" Class-aware wrapper for `::pdf_annot_border_style()`."""
37299: (4)             return _mupdf.pdf_annot_border_style(annot)
37300: (0)         def pdf_annot_border_width(annot):
37301: (4)             r""" Class-aware wrapper for `::pdf_annot_border_width()`."""
37302: (4)             return _mupdf.pdf_annot_border_width(annot)
37303: (0)         def pdf_annot_color(annot, n, color):
37304: (4)             r"""
37305: (4)             Class-aware wrapper for `::pdf_annot_color()`.
37306: (4)             This function has out-params. Python/C# wrappers look like:
37307: (8)                 `pdf_annot_color(::pdf_annot *annot, float color[4])` => int n
37308: (4)             """
37309: (4)             return _mupdf.pdf_annot_color(annot, n, color)
37310: (0)         def pdf_annot_contents(annot):
37311: (4)             r""" Class-aware wrapper for `::pdf_annot_contents()`."""
37312: (4)             return _mupdf.pdf_annot_contents(annot)
37313: (0)         def pdf_annot_creation_date(annot):
37314: (4)             r""" Class-aware wrapper for `::pdf_annot_creation_date()`."""
37315: (4)             return _mupdf.pdf_annot_creation_date(annot)
37316: (0)         def pdf_annot_default_appearance(annot, font, size, n, color):
37317: (4)             r"""
37318: (4)             Class-aware wrapper for `::pdf_annot_default_appearance()`.
37319: (4)             This function has out-params. Python/C# wrappers look like:
37320: (8)                 `pdf_annot_default_appearance(::pdf_annot *annot, float color[4])` =>
`(const char *font, float size, int n)`
37321: (4)             """
37322: (4)             return _mupdf.pdf_annot_default_appearance(annot, font, size, n, color)
37323: (0)         def pdf_annot_ensure_local_xref(annot):
37324: (4)             r""" Class-aware wrapper for `::pdf_annot_ensure_local_xref()`."""
37325: (4)             return _mupdf.pdf_annot_ensure_local_xref(annot)
37326: (0)         def pdf_annot_event_blur(annot):
37327: (4)             r""" Class-aware wrapper for `::pdf_annot_event_blur()`."""
37328: (4)             return _mupdf.pdf_annot_event_blur(annot)
37329: (0)         def pdf_annot_event_down(annot):
37330: (4)             r""" Class-aware wrapper for `::pdf_annot_event_down()`."""
37331: (4)             return _mupdf.pdf_annot_event_down(annot)
37332: (0)         def pdf_annot_event_enter(annot):
37333: (4)             r""" Class-aware wrapper for `::pdf_annot_event_enter()`."""
37334: (4)             return _mupdf.pdf_annot_event_enter(annot)
37335: (0)         def pdf_annot_event_exit(annot):
37336: (4)             r""" Class-aware wrapper for `::pdf_annot_event_exit()`."""
37337: (4)             return _mupdf.pdf_annot_event_exit(annot)
37338: (0)         def pdf_annot_event_focus(annot):
37339: (4)             r""" Class-aware wrapper for `::pdf_annot_event_focus()`."""
37340: (4)             return _mupdf.pdf_annot_event_focus(annot)
37341: (0)         def pdf_annot_event_page_close(annot):
37342: (4)             r""" Class-aware wrapper for `::pdf_annot_event_page_close()`."""
37343: (4)             return _mupdf.pdf_annot_event_page_close(annot)
37344: (0)         def pdf_annot_event_page_invisible(annot):
```

```
(4)             r""" Class-aware wrapper for `::pdf_annot_event_page_invisible()`."""
(4)             return _mupdf.pdf_annot_event_page_invisible(annot)
(0)         def pdf_annot_event_page_open(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_event_page_open()`."""
(4)             return _mupdf.pdf_annot_event_page_open(annot)
(0)         def pdf_annot_event_page_visible(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_event_page_visible()`."""
(4)             return _mupdf.pdf_annot_event_page_visible(annot)
(0)         def pdf_annot_event_up(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_event_up()`."""
(4)             return _mupdf.pdf_annot_event_up(annot)
(0)         def pdf_annot_field_event_keystroke(doc, annot, evt):
(4)             r""" Class-aware wrapper for `::pdf_annot_field_event_keystroke()`."""
(4)             return _mupdf.pdf_annot_field_event_keystroke(doc, annot, evt)
(0)         def pdf_annot_field_flags(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_field_flags()`."""
(4)             return _mupdf.pdf_annot_field_flags(annot)
(0)         def pdf_annot_field_label(widget):
(4)             r""" Class-aware wrapper for `::pdf_annot_field_label()`."""
(4)             return _mupdf.pdf_annot_field_label(widget)
(0)         def pdf_annot_field_value(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_field_value()`."""
(4)             return _mupdf.pdf_annot_field_value(annot)
(0)         def pdf_annot_filespec(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_filespec()`."""
(4)             return _mupdf.pdf_annot_filespec(annot)
(0)         def pdf_annot_flags(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_flags()`."""
(4)             return _mupdf.pdf_annot_flags(annot)
(0)         def pdf_annot_has_author(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_author()`."""
(4)             return _mupdf.pdf_annot_has_author(annot)
(0)         def pdf_annot_has_border(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_border()`."""
(4)             return _mupdf.pdf_annot_has_border(annot)
(0)         def pdf_annot_has_border_effect(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_border_effect()`."""
(4)             return _mupdf.pdf_annot_has_border_effect(annot)
(0)         def pdf_annot_has_filespec(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_filespec()`."""
(4)             return _mupdf.pdf_annot_has_filespec(annot)
(0)         def pdf_annot_has_icon_name(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_icon_name()`."""
(4)             return _mupdf.pdf_annot_has_icon_name(annot)
(0)         def pdf_annot_has_ink_list(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_ink_list()`."""
(4)             return _mupdf.pdf_annot_has_ink_list(annot)
(0)         def pdf_annot_has_intent(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_intent()`."""
(4)             return _mupdf.pdf_annot_has_intent(annot)
(0)         def pdf_annot_has_interior_color(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_interior_color()`."""
(4)             return _mupdf.pdf_annot_has_interior_color(annot)
(0)         def pdf_annot_has_line(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_line()`."""
(4)             return _mupdf.pdf_annot_has_line(annot)
(0)         def pdf_annot_has_line_ending_styles(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_line_ending_styles()`."""
(4)             return _mupdf.pdf_annot_has_line_ending_styles(annot)
(0)         def pdf_annot_has_open(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_open()`."""
(4)             return _mupdf.pdf_annot_has_open(annot)
(0)         def pdf_annot_has_quad_points(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_quad_points()`."""
(4)             return _mupdf.pdf_annot_has_quad_points(annot)
(0)         def pdf_annot_has_quadding(annot):
(4)             r""" Class-aware wrapper for `::pdf_annot_has_quadding()`."""
(4)             return _mupdf.pdf_annot_has_quadding(annot)
(0)         def pdf_annot_has_rect(annot):
```

```
37414: (4)             r""" Class-aware wrapper for `::pdf_annot_has_rect()`."""
37415: (4)             return _mupdf.pdf_annot_has_rect(annot)
37416: (0)         def pdf_annot_has_vertices(annot):
37417: (4)             r""" Class-aware wrapper for `::pdf_annot_has_vertices()`."""
37418: (4)             return _mupdf.pdf_annot_has_vertices(annot)
37419: (0)         def pdf_annot_hidden_for_editing(annot):
37420: (4)             r""" Class-aware wrapper for `::pdf_annot_hidden_for_editing()`."""
37421: (4)             return _mupdf.pdf_annot_hidden_for_editing(annot)
37422: (0)         def pdf_annot_hot(annot):
37423: (4)             r""" Class-aware wrapper for `::pdf_annot_hot()`."""
37424: (4)             return _mupdf.pdf_annot_hot(annot)
37425: (0)         def pdf_annot_icon_name(annot):
37426: (4)             r""" Class-aware wrapper for `::pdf_annot_icon_name()`."""
37427: (4)             return _mupdf.pdf_annot_icon_name(annot)
37428: (0)         def pdf_annot_ink_list_count(annot):
37429: (4)             r""" Class-aware wrapper for `::pdf_annot_ink_list_count()`."""
37430: (4)             return _mupdf.pdf_annot_ink_list_count(annot)
37431: (0)         def pdf_annot_ink_list_stroke_count(annot, i):
37432: (4)             r""" Class-aware wrapper for `::pdf_annot_ink_list_stroke_count()`."""
37433: (4)             return _mupdf.pdf_annot_ink_list_stroke_count(annot, i)
37434: (0)         def pdf_annot_ink_list_stroke_vertex(annot, i, k):
37435: (4)             r""" Class-aware wrapper for `::pdf_annot_ink_list_stroke_vertex()`."""
37436: (4)             return _mupdf.pdf_annot_ink_list_stroke_vertex(annot, i, k)
37437: (0)         def pdf_annot_intent(annot):
37438: (4)             r""" Class-aware wrapper for `::pdf_annot_intent()`."""
37439: (4)             return _mupdf.pdf_annot_intent(annot)
37440: (0)         def pdf_annot_interior_color(annot, n, color):
37441: (4)             r"""
37442: (4)             Class-aware wrapper for `::pdf_annot_interior_color()`.
37443: (4)             This function has out-params. Python/C# wrappers look like:
37444: (8)                 `pdf_annot_interior_color(::pdf_annot *annot, float color[4])` => int
n
37445: (4)             """
37446: (4)             return _mupdf.pdf_annot_interior_color(annot, n, color)
37447: (0)         def pdf_annot_is_open(annot):
37448: (4)             r""" Class-aware wrapper for `::pdf_annot_is_open()`."""
37449: (4)             return _mupdf.pdf_annot_is_open(annot)
37450: (0)         def pdf_annot_is_standard_stamp(annot):
37451: (4)             r""" Class-aware wrapper for `::pdf_annot_is_standard_stamp()`."""
37452: (4)             return _mupdf.pdf_annot_is_standard_stamp(annot)
37453: (0)         def pdf_annot_language(annot):
37454: (4)             r""" Class-aware wrapper for `::pdf_annot_language()`."""
37455: (4)             return _mupdf.pdf_annot_language(annot)
37456: (0)         def pdf_annot_line(annot, a, b):
37457: (4)             r""" Class-aware wrapper for `::pdf_annot_line()`."""
37458: (4)             return _mupdf.pdf_annot_line(annot, a, b)
37459: (0)         def pdf_annot_line_end_style(annot):
37460: (4)             r""" Class-aware wrapper for `::pdf_annot_line_end_style()`."""
37461: (4)             return _mupdf.pdf_annot_line_end_style(annot)
37462: (0)         def pdf_annot_line_ending_styles(annot, start_style, end_style):
37463: (4)             r"""
37464: (4)             Class-aware wrapper for `::pdf_annot_line_ending_styles()`.
37465: (4)             This function has out-params. Python/C# wrappers look like:
37466: (8)                 `pdf_annot_line_ending_styles(::pdf_annot *annot)` => `(enum
pdf_line_ending start_style, enum pdf_line_ending end_style)`
37467: (4)             """
37468: (4)             return _mupdf.pdf_annot_line_ending_styles(annot, start_style, end_style)
37469: (0)         def pdf_annot_line_start_style(annot):
37470: (4)             r""" Class-aware wrapper for `::pdf_annot_line_start_style()`."""
37471: (4)             return _mupdf.pdf_annot_line_start_style(annot)
37472: (0)         def pdf_annot_modification_date(annot):
37473: (4)             r""" Class-aware wrapper for `::pdf_annot_modification_date()`."""
37474: (4)             return _mupdf.pdf_annot_modification_date(annot)
37475: (0)         def pdf_annot_needs_resynthesis(annot):
37476: (4)             r""" Class-aware wrapper for `::pdf_annot_needs_resynthesis()`."""
37477: (4)             return _mupdf.pdf_annot_needs_resynthesis(annot)
37478: (0)         def pdf_annot_obj(annot):
37479: (4)             r""" Class-aware wrapper for `::pdf_annot_obj()`."""
37480: (4)             return _mupdf.pdf_annot_obj(annot)
```

```
37481: (0)          def pdf_annot_opacity(annot):
37482: (4)              r""" Class-aware wrapper for `::pdf_annot_opacity()`."""
37483: (4)              return _mupdf.pdf_annot_opacity(annot)
37484: (0)          def pdf_annot_page(annot):
37485: (4)              r""" Class-aware wrapper for `::pdf_annot_page()`."""
37486: (4)              return _mupdf.pdf_annot_page(annot)
37487: (0)          def pdf_annot_pop_and_discard_local_xref(annot):
37488: (4)              r""" Class-aware wrapper for
`::pdf_annot_pop_and_discard_local_xref()`."""
37489: (4)              return _mupdf.pdf_annot_pop_and_discard_local_xref(annot)
37490: (0)          def pdf_annot_pop_local_xref(annot):
37491: (4)              r""" Class-aware wrapper for `::pdf_annot_pop_local_xref()`."""
37492: (4)              return _mupdf.pdf_annot_pop_local_xref(annot)
37493: (0)          def pdf_annot_popup(annot):
37494: (4)              r""" Class-aware wrapper for `::pdf_annot_popup()`."""
37495: (4)              return _mupdf.pdf_annot_popup(annot)
37496: (0)          def pdf_annot_push_local_xref(annot):
37497: (4)              r""" Class-aware wrapper for `::pdf_annot_push_local_xref()`."""
37498: (4)              return _mupdf.pdf_annot_push_local_xref(annot)
37499: (0)          def pdf_annot_quad_point(annot, i):
37500: (4)              r""" Class-aware wrapper for `::pdf_annot_quad_point()`."""
37501: (4)              return _mupdf.pdf_annot_quad_point(annot, i)
37502: (0)          def pdf_annot_quad_point_count(annot):
37503: (4)              r""" Class-aware wrapper for `::pdf_annot_quad_point_count()`."""
37504: (4)              return _mupdf.pdf_annot_quad_point_count(annot)
37505: (0)          def pdf_annot_quadding(annot):
37506: (4)              r""" Class-aware wrapper for `::pdf_annot_quadding()`."""
37507: (4)              return _mupdf.pdf_annot_quadding(annot)
37508: (0)          def pdf_annot_rect(annot):
37509: (4)              r""" Class-aware wrapper for `::pdf_annot_rect()`."""
37510: (4)              return _mupdf.pdf_annot_rect(annot)
37511: (0)          def pdf_annot_request_resynthesis(annot):
37512: (4)              r""" Class-aware wrapper for `::pdf_annot_request_resynthesis()`."""
37513: (4)              return _mupdf.pdf_annot_request_resynthesis(annot)
37514: (0)          def pdf_annot_request_synthesis(annot):
37515: (4)              r""" Class-aware wrapper for `::pdf_annot_request_synthesis()`."""
37516: (4)              return _mupdf.pdf_annot_request_synthesis(annot)
37517: (0)          def pdf_annot_transform(annot):
37518: (4)              r""" Class-aware wrapper for `::pdf_annot_transform()`."""
37519: (4)              return _mupdf.pdf_annot_transform(annot)
37520: (0)          def pdf_annot_type(annot):
37521: (4)              r""" Class-aware wrapper for `::pdf_annot_type()`."""
37522: (4)              return _mupdf.pdf_annot_type(annot)
37523: (0)          def pdf_annot_type_from_string(subtype):
37524: (4)              r""" Class-aware wrapper for `::pdf_annot_type_from_string()`."""
37525: (4)              return _mupdf.pdf_annot_type_from_string(subtype)
37526: (0)          def pdf_annot_vertex(annot, i):
37527: (4)              r""" Class-aware wrapper for `::pdf_annot_vertex()`."""
37528: (4)              return _mupdf.pdf_annot_vertex(annot, i)
37529: (0)          def pdf_annot_vertex_count(annot):
37530: (4)              r""" Class-aware wrapper for `::pdf_annot_vertex_count()`."""
37531: (4)              return _mupdf.pdf_annot_vertex_count(annot)
37532: (0)          def pdf_append_explicit_dest_to_uri(url, dest):
37533: (4)              r""" Class-aware wrapper for `::pdf_append_explicit_dest_to_uri()`."""
37534: (4)              return _mupdf.pdf_append_explicit_dest_to_uri(url, dest)
37535: (0)          def pdf_append_named_dest_to_uri(url, name):
37536: (4)              r""" Class-aware wrapper for `::pdf_append_named_dest_to_uri()`."""
37537: (4)              return _mupdf.pdf_append_named_dest_to_uri(url, name)
37538: (0)          def pdf_append_token(buf, tok, lex):
37539: (4)              r""" Class-aware wrapper for `::pdf_append_token()`."""
37540: (4)              return _mupdf.pdf_append_token(buf, tok, lex)
37541: (0)          def pdf_apply_redaction(annot, opts):
37542: (4)              r""" Class-aware wrapper for `::pdf_apply_redaction()`."""
37543: (4)              return _mupdf.pdf_apply_redaction(annot, opts)
37544: (0)          def pdf_array_contains(array, obj):
37545: (4)              r""" Class-aware wrapper for `::pdf_array_contains()`."""
37546: (4)              return _mupdf.pdf_array_contains(array, obj)
37547: (0)          def pdf_array_delete(array, index):
37548: (4)              r""" Class-aware wrapper for `::pdf_array_delete()`."""
```

```
37549: (4)             return _mupdf.pdf_array_delete(array, index)
37550: (0)         def pdf_array_find(array, obj):
37551: (4)             r""" Class-aware wrapper for `::pdf_array_find()`."""
37552: (4)             return _mupdf.pdf_array_find(array, obj)
37553: (0)         def pdf_array_get(array, i):
37554: (4)             r""" Class-aware wrapper for `::pdf_array_get()`."""
37555: (4)             return _mupdf.pdf_array_get(array, i)
37556: (0)         def pdf_array_get_bool(array, index):
37557: (4)             r""" Class-aware wrapper for `::pdf_array_get_bool()`."""
37558: (4)             return _mupdf.pdf_array_get_bool(array, index)
37559: (0)         def pdf_array_get_int(array, index):
37560: (4)             r""" Class-aware wrapper for `::pdf_array_get_int()`."""
37561: (4)             return _mupdf.pdf_array_get_int(array, index)
37562: (0)         def pdf_array_get_matrix(array, index):
37563: (4)             r""" Class-aware wrapper for `::pdf_array_get_matrix()`."""
37564: (4)             return _mupdf.pdf_array_get_matrix(array, index)
37565: (0)         def pdf_array_get_name(array, index):
37566: (4)             r""" Class-aware wrapper for `::pdf_array_get_name()`."""
37567: (4)             return _mupdf.pdf_array_get_name(array, index)
37568: (0)         def pdf_array_get_real(array, index):
37569: (4)             r""" Class-aware wrapper for `::pdf_array_get_real()`."""
37570: (4)             return _mupdf.pdf_array_get_real(array, index)
37571: (0)         def pdf_array_get_rect(array, index):
37572: (4)             r""" Class-aware wrapper for `::pdf_array_get_rect()`."""
37573: (4)             return _mupdf.pdf_array_get_rect(array, index)
37574: (0)         def pdf_array_get_string(array, index, sizep):
37575: (4)             r"""
37576: (4)             Class-aware wrapper for `::pdf_array_get_string()`.
37577: (4)             This function has out-params. Python/C# wrappers look like:
37578: (8)                 `pdf_array_get_string(::pdf_obj *array, int index)` => `(const char *,
size_t sizep)`
37579: (4)             """
37580: (4)             return _mupdf.pdf_array_get_string(array, index, sizep)
37581: (0)         def pdf_array_get_text_string(array, index):
37582: (4)             r""" Class-aware wrapper for `::pdf_array_get_text_string()`."""
37583: (4)             return _mupdf.pdf_array_get_text_string(array, index)
37584: (0)         def pdf_array_insert(array, obj, index):
37585: (4)             r""" Class-aware wrapper for `::pdf_array_insert()`."""
37586: (4)             return _mupdf.pdf_array_insert(array, obj, index)
37587: (0)         def pdf_array_len(array):
37588: (4)             r""" Class-aware wrapper for `::pdf_array_len()`."""
37589: (4)             return _mupdf.pdf_array_len(array)
37590: (0)         def pdf_array_push(array, obj):
37591: (4)             r""" Class-aware wrapper for `::pdf_array_push()`."""
37592: (4)             return _mupdf.pdf_array_push(array, obj)
37593: (0)         def pdf_array_push_array(array, initial):
37594: (4)             r""" Class-aware wrapper for `::pdf_array_push_array()`."""
37595: (4)             return _mupdf.pdf_array_push_array(array, initial)
37596: (0)         def pdf_array_push_bool(array, x):
37597: (4)             r""" Class-aware wrapper for `::pdf_array_push_bool()`."""
37598: (4)             return _mupdf.pdf_array_push_bool(array, x)
37599: (0)         def pdf_array_push_dict(array, initial):
37600: (4)             r""" Class-aware wrapper for `::pdf_array_push_dict()`."""
37601: (4)             return _mupdf.pdf_array_push_dict(array, initial)
37602: (0)         def pdf_array_push_int(array, x):
37603: (4)             r""" Class-aware wrapper for `::pdf_array_push_int()`."""
37604: (4)             return _mupdf.pdf_array_push_int(array, x)
37605: (0)         def pdf_array_push_name(array, x):
37606: (4)             r""" Class-aware wrapper for `::pdf_array_push_name()`."""
37607: (4)             return _mupdf.pdf_array_push_name(array, x)
37608: (0)         def pdf_array_push_real(array, x):
37609: (4)             r""" Class-aware wrapper for `::pdf_array_push_real()`."""
37610: (4)             return _mupdf.pdf_array_push_real(array, x)
37611: (0)         def pdf_array_push_string(array, x, n):
37612: (4)             r""" Class-aware wrapper for `::pdf_array_push_string()`."""
37613: (4)             return _mupdf.pdf_array_push_string(array, x, n)
37614: (0)         def pdf_array_push_text_string(array, x):
37615: (4)             r""" Class-aware wrapper for `::pdf_array_push_text_string()`."""
37616: (4)             return _mupdf.pdf_array_push_text_string(array, x)
```

```
(0)         def pdf_array_put(array, i, obj):
(4)             r""" Class-aware wrapper for `::pdf_array_put()`."""
(4)             return _mupdf.pdf_array_put(array, i, obj)
(0)         def pdf_array_put_array(array, i, initial):
(4)             r""" Class-aware wrapper for `::pdf_array_put_array()`."""
(4)             return _mupdf.pdf_array_put_array(array, i, initial)
(0)         def pdf_array_put_bool(array, i, x):
(4)             r""" Class-aware wrapper for `::pdf_array_put_bool()`."""
(4)             return _mupdf.pdf_array_put_bool(array, i, x)
(0)         def pdf_array_put_dict(array, i, initial):
(4)             r""" Class-aware wrapper for `::pdf_array_put_dict()`."""
(4)             return _mupdf.pdf_array_put_dict(array, i, initial)
(0)         def pdf_array_put_int(array, i, x):
(4)             r""" Class-aware wrapper for `::pdf_array_put_int()`."""
(4)             return _mupdf.pdf_array_put_int(array, i, x)
(0)         def pdf_array_put_name(array, i, x):
(4)             r""" Class-aware wrapper for `::pdf_array_put_name()`."""
(4)             return _mupdf.pdf_array_put_name(array, i, x)
(0)         def pdf_array_put_real(array, i, x):
(4)             r""" Class-aware wrapper for `::pdf_array_put_real()`."""
(4)             return _mupdf.pdf_array_put_real(array, i, x)
(0)         def pdf_array_put_string(array, i, x, n):
(4)             r""" Class-aware wrapper for `::pdf_array_put_string()`."""
(4)             return _mupdf.pdf_array_put_string(array, i, x, n)
(0)         def pdf_array_put_text_string(array, i, x):
(4)             r""" Class-aware wrapper for `::pdf_array_put_text_string()`."""
(4)             return _mupdf.pdf_array_put_text_string(array, i, x)
(0)         def pdf_authenticate_password(doc, pw):
(4)             r""" Class-aware wrapper for `::pdf_authenticate_password()`."""
(4)             return _mupdf.pdf_authenticate_password(doc, pw)
(0)         def pdf_bake_document(doc, bake_annots, bake_widgets):
(4)             r""" Class-aware wrapper for `::pdf_bake_document()`."""
(4)             return _mupdf.pdf_bake_document(doc, bake_annots, bake_widgets)
(0)         def pdf_begin_implicit_operation(doc):
(4)             r""" Class-aware wrapper for `::pdf_begin_implicit_operation()`."""
(4)             return _mupdf.pdf_begin_implicit_operation(doc)
(0)         def pdf_begin_operation(doc, operation):
(4)             r""" Class-aware wrapper for `::pdf_begin_operation()`."""
(4)             return _mupdf.pdf_begin_operation(doc, operation)
(0)         def pdf_bound_annot(annot):
(4)             r""" Class-aware wrapper for `::pdf_bound_annot()`."""
(4)             return _mupdf.pdf_bound_annot(annot)
(0)         def pdf_bound_page(page, box):
(4)             r""" Class-aware wrapper for `::pdf_bound_page()`."""
(4)             return _mupdf.pdf_bound_page(page, box)
(0)         def pdf_bound_widget(widget):
(4)             r""" Class-aware wrapper for `::pdf_bound_widget()`."""
(4)             return _mupdf.pdf_bound_widget(widget)
(0)         def pdf_button_field_on_state(field):
(4)             r""" Class-aware wrapper for `::pdf_button_field_on_state()`."""
(4)             return _mupdf.pdf_button_field_on_state(field)
(0)         def pdf_calculate_form(doc):
(4)             r""" Class-aware wrapper for `::pdf_calculate_form()`."""
(4)             return _mupdf.pdf_calculate_form(doc)
(0)         def pdf_can_be_saved_incrementally(doc):
(4)             r""" Class-aware wrapper for `::pdf_can_be_saved_incrementally()`."""
(4)             return _mupdf.pdf_can_be_saved_incrementally(doc)
(0)         def pdf_can_redo(doc):
(4)             r""" Class-aware wrapper for `::pdf_can_redo()`."""
(4)             return _mupdf.pdf_can_redo(doc)
(0)         def pdf_can_undo(doc):
(4)             r""" Class-aware wrapper for `::pdf_can_undo()`."""
(4)             return _mupdf.pdf_can_undo(doc)
(0)         def pdf_check_certificate(verifier, doc, signature):
(4)             r""" Class-aware wrapper for `::pdf_check_certificate()`."""
(4)             return _mupdf.pdf_check_certificate(verifier, doc, signature)
(0)         def pdf_check_digest(verifier, doc, signature):
(4)             r""" Class-aware wrapper for `::pdf_check_digest()`."""
(4)             return _mupdf.pdf_check_digest(verifier, doc, signature)
```

```
(0)         def pdf_check_widget_certificate(verifier, widget):
(4)             r""" Class-aware wrapper for `::pdf_check_widget_certificate()`."""
(4)             return _mupdf.pdf_check_widget_certificate(verifier, widget)
(0)         def pdf_check_widget_digest(verifier, widget):
(4)             r""" Class-aware wrapper for `::pdf_check_widget_digest()`."""
(4)             return _mupdf.pdf_check_widget_digest(verifier, widget)
(0)         def pdf_choice_field_option(field, exportval, i):
(4)             r""" Class-aware wrapper for `::pdf_choice_field_option()`."""
(4)             return _mupdf.pdf_choice_field_option(field, exportval, i)
(0)         def pdf_choice_field_option_count(field):
(4)             r""" Class-aware wrapper for `::pdf_choice_field_option_count()`."""
(4)             return _mupdf.pdf_choice_field_option_count(field)
(0)         def pdf_choice_widget_is_multiselect(tw):
(4)             r""" Class-aware wrapper for `::pdf_choice_widget_is_multiselect()`."""
(4)             return _mupdf.pdf_choice_widget_is_multiselect(tw)
(0)         def pdf_choice_widget_options(tw, exportval, opts):
(4)             r""" Class-aware wrapper for `::pdf_choice_widget_options()`."""
(4)             return _mupdf.pdf_choice_widget_options(tw, exportval, opts)
(0)         def pdf_choice_widget_options2(tw, exportval):
(4)             r"""
(4)             Class-aware wrapper for `::pdf_choice_widget_options2()`.   Swig-friendly
wrapper for pdf_choice_widget_options(), returns the
(4)             options directly in a vector.
(4)             """
(4)             return _mupdf.pdf_choice_widget_options2(tw, exportval)
(0)         def pdf_choice_widget_set_value(tw, n, opts):
(4)             r""" Class-aware wrapper for `::pdf_choice_widget_set_value()`."""
(4)             return _mupdf.pdf_choice_widget_set_value(tw, n, opts)
(0)         def pdf_choice_widget_value(tw, opts):
(4)             r""" Class-aware wrapper for `::pdf_choice_widget_value()`."""
(4)             return _mupdf.pdf_choice_widget_value(tw, opts)
(0)         def pdf_clean_file(infile, outfile, password, opts, retainlen):
(4)             r""" Class-aware wrapper for `::pdf_clean_file()`."""
(4)             return _mupdf.pdf_clean_file(infile, outfile, password, opts, retainlen)
(0)         def pdf_clean_font_name(fontname):
(4)             r""" Class-aware wrapper for `::pdf_clean_font_name()`."""
(4)             return _mupdf.pdf_clean_font_name(fontname)
(0)         def pdf_clean_obj(obj):
(4)             r""" Class-aware wrapper for `::pdf_clean_obj()`."""
(4)             return _mupdf.pdf_clean_obj(obj)
(0)         def pdf_clear_annot_border_dash(annot):
(4)             r""" Class-aware wrapper for `::pdf_clear_annot_border_dash()`."""
(4)             return _mupdf.pdf_clear_annot_border_dash(annot)
(0)         def pdf_clear_annot_ink_list(annot):
(4)             r""" Class-aware wrapper for `::pdf_clear_annot_ink_list()`."""
(4)             return _mupdf.pdf_clear_annot_ink_list(annot)
(0)         def pdf_clear_annot_quad_points(annot):
(4)             r""" Class-aware wrapper for `::pdf_clear_annot_quad_points()`."""
(4)             return _mupdf.pdf_clear_annot_quad_points(annot)
(0)         def pdf_clear_annot_vertices(annot):
(4)             r""" Class-aware wrapper for `::pdf_clear_annot_vertices()`."""
(4)             return _mupdf.pdf_clear_annot_vertices(annot)
(0)         def pdf_clear_signature(widget):
(4)             r""" Class-aware wrapper for `::pdf_clear_signature()`."""
(4)             return _mupdf.pdf_clear_signature(widget)
(0)         def pdf_clear_xref(doc):
(4)             r""" Class-aware wrapper for `::pdf_clear_xref()`."""
(4)             return _mupdf.pdf_clear_xref(doc)
(0)         def pdf_clear_xref_to_mark(doc):
(4)             r""" Class-aware wrapper for `::pdf_clear_xref_to_mark()`."""
(4)             return _mupdf.pdf_clear_xref_to_mark(doc)
(0)         def pdf_close_processor(proc):
(4)             r""" Class-aware wrapper for `::pdf_close_processor()`."""
(4)             return _mupdf.pdf_close_processor(proc)
(0)         def pdf_cmap_size(cmap):
(4)             r""" Class-aware wrapper for `::pdf_cmap_size()`."""
(4)             return _mupdf.pdf_cmap_size(cmap)
(0)         def pdf_cmap_wmode(cmap):
(4)             r""" Class-aware wrapper for `::pdf_cmap_wmode()`."""
```

```
(4)             return _mupdf.pdf_cmap_wmode(cmap)
(0)         def pdf_copy_array(array):
(4)             r""" Class-aware wrapper for `::pdf_copy_array()`."""
(4)             return _mupdf.pdf_copy_array(array)
(0)         def pdf_copy_dict(dict):
(4)             r""" Class-aware wrapper for `::pdf_copy_dict()`."""
(4)             return _mupdf.pdf_copy_dict(dict)
(0)         def pdf_count_layer_config_ui(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_layer_config_ui()`."""
(4)             return _mupdf.pdf_count_layer_config_ui(doc)
(0)         def pdf_count_layer_configs(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_layer_configs()`."""
(4)             return _mupdf.pdf_count_layer_configs(doc)
(0)         def pdf_count_layers(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_layers()`."""
(4)             return _mupdf.pdf_count_layers(doc)
(0)         def pdf_count_objects(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_objects()`."""
(4)             return _mupdf.pdf_count_objects(doc)
(0)         def pdf_count_pages(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_pages()`."""
(4)             return _mupdf.pdf_count_pages(doc)
(0)         def pdf_count_pages_imp(doc, chapter):
(4)             r""" Class-aware wrapper for `::pdf_count_pages_imp()`."""
(4)             return _mupdf.pdf_count_pages_imp(doc, chapter)
(0)         def pdf_count_q_balance(doc, res, stm, prepend, append):
(4)             r"""
(4)             Class-aware wrapper for `::pdf_count_q_balance()`.
(4)             This function has out-params. Python/C# wrappers look like:
(8)                 `pdf_count_q_balance(::pdf_document *doc, ::pdf_obj *res, ::pdf_obj
*stm)` => `(int prepend, int append)`
(4)             """
(4)             return _mupdf.pdf_count_q_balance(doc, res, stm, prepend, append)
(0)         def pdf_count_signatures(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_signatures()`."""
(4)             return _mupdf.pdf_count_signatures(doc)
(0)         def pdf_count_unsaved_versions(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_unsaved_versions()`."""
(4)             return _mupdf.pdf_count_unsaved_versions(doc)
(0)         def pdf_count_versions(doc):
(4)             r""" Class-aware wrapper for `::pdf_count_versions()`."""
(4)             return _mupdf.pdf_count_versions(doc)
(0)         def pdf_create_annot(page, type):
(4)             r""" Class-aware wrapper for `::pdf_create_annot()`."""
(4)             return _mupdf.pdf_create_annot(page, type)
(0)         def pdf_create_annot_raw(page, type):
(4)             r""" Class-aware wrapper for `::pdf_create_annot_raw()`."""
(4)             return _mupdf.pdf_create_annot_raw(page, type)
(0)         def pdf_create_document():
(4)             r""" Class-aware wrapper for `::pdf_create_document()`."""
(4)             return _mupdf.pdf_create_document()
(0)         def pdf_create_field_name(doc, prefix, buf, len):
(4)             r""" Class-aware wrapper for `::pdf_create_field_name()`."""
(4)             return _mupdf.pdf_create_field_name(doc, prefix, buf, len)
(0)         def pdf_create_link(page, bbox, uri):
(4)             r""" Class-aware wrapper for `::pdf_create_link()`."""
(4)             return _mupdf.pdf_create_link(page, bbox, uri)
(0)         def pdf_create_object(doc):
(4)             r""" Class-aware wrapper for `::pdf_create_object()`."""
(4)             return _mupdf.pdf_create_object(doc)
(0)         def pdf_create_signature_widget(page, name):
(4)             r""" Class-aware wrapper for `::pdf_create_signature_widget()`."""
(4)             return _mupdf.pdf_create_signature_widget(page, name)
(0)         def pdf_crypt_encrypt_metadata(crypt):
(4)             r""" Class-aware wrapper for `::pdf_crypt_encrypt_metadata()`."""
(4)             return _mupdf.pdf_crypt_encrypt_metadata(crypt)
(0)         def pdf_crypt_key(crypt):
(4)             r""" Class-aware wrapper for `::pdf_crypt_key()`."""
(4)             return _mupdf.pdf_crypt_key(crypt)
```

```
37822: (0)          def pdf_crypt_length(crypt):
37823: (4)              r""" Class-aware wrapper for `::pdf_crypt_length()`."""
37824: (4)              return _mupdf.pdf_crypt_length(crypt)
37825: (0)          def pdf_crypt_method(crypt):
37826: (4)              r""" Class-aware wrapper for `::pdf_crypt_method()`."""
37827: (4)              return _mupdf.pdf_crypt_method(crypt)
37828: (0)          def pdf_crypt_obj(crypt, obj, num, gen):
37829: (4)              r""" Class-aware wrapper for `::pdf_crypt_obj()`."""
37830: (4)              return _mupdf.pdf_crypt_obj(crypt, obj, num, gen)
37831: (0)          def pdf_crypt_owner_encryption(crypt):
37832: (4)              r""" Class-aware wrapper for `::pdf_crypt_owner_encryption()`."""
37833: (4)              return _mupdf.pdf_crypt_owner_encryption(crypt)
37834: (0)          def pdf_crypt_owner_password(crypt):
37835: (4)              r""" Class-aware wrapper for `::pdf_crypt_owner_password()`."""
37836: (4)              return _mupdf.pdf_crypt_owner_password(crypt)
37837: (0)          def pdf_crypt_permissions(crypt):
37838: (4)              r""" Class-aware wrapper for `::pdf_crypt_permissions()`."""
37839: (4)              return _mupdf.pdf_crypt_permissions(crypt)
37840: (0)          def pdf_crypt_permissions_encryption(crypt):
37841: (4)              r""" Class-aware wrapper for `::pdf_crypt_permissions_encryption()`."""
37842: (4)              return _mupdf.pdf_crypt_permissions_encryption(crypt)
37843: (0)          def pdf_crypt_revision(crypt):
37844: (4)              r""" Class-aware wrapper for `::pdf_crypt_revision()`."""
37845: (4)              return _mupdf.pdf_crypt_revision(crypt)
37846: (0)          def pdf_crypt_stream_method(crypt):
37847: (4)              r""" Class-aware wrapper for `::pdf_crypt_stream_method()`."""
37848: (4)              return _mupdf.pdf_crypt_stream_method(crypt)
37849: (0)          def pdf_crypt_string_method(crypt):
37850: (4)              r""" Class-aware wrapper for `::pdf_crypt_string_method()`."""
37851: (4)              return _mupdf.pdf_crypt_string_method(crypt)
37852: (0)          def pdf_crypt_user_encryption(crypt):
37853: (4)              r""" Class-aware wrapper for `::pdf_crypt_user_encryption()`."""
37854: (4)              return _mupdf.pdf_crypt_user_encryption(crypt)
37855: (0)          def pdf_crypt_user_password(crypt):
37856: (4)              r""" Class-aware wrapper for `::pdf_crypt_user_password()`."""
37857: (4)              return _mupdf.pdf_crypt_user_password(crypt)
37858: (0)          def pdf_crypt_version(crypt):
37859: (4)              r""" Class-aware wrapper for `::pdf_crypt_version()`."""
37860: (4)              return _mupdf.pdf_crypt_version(crypt)
37861: (0)          def pdf_cycle(here, prev, obj):
37862: (4)              r""" Class-aware wrapper for `::pdf_cycle()`."""
37863: (4)              return _mupdf.pdf_cycle(here, prev, obj)
37864: (0)          def pdf_debug_doc_changes(doc):
37865: (4)              r""" Class-aware wrapper for `::pdf_debug_doc_changes()`."""
37866: (4)              return _mupdf.pdf_debug_doc_changes(doc)
37867: (0)          def pdf_debug_obj(obj):
37868: (4)              r""" Class-aware wrapper for `::pdf_debug_obj()`."""
37869: (4)              return _mupdf.pdf_debug_obj(obj)
37870: (0)          def pdf_debug_ref(obj):
37871: (4)              r""" Class-aware wrapper for `::pdf_debug_ref()`."""
37872: (4)              return _mupdf.pdf_debug_ref(obj)
37873: (0)          def pdf_decode_cmap(cmap, s, e, cpt):
37874: (4)              r"""
37875: (4)              Class-aware wrapper for `::pdf_decode_cmap()`.
37876: (4)              This function has out-params. Python/C# wrappers look like:
37877: (8)                  `pdf_decode_cmap(::pdf_cmap *cmap, unsigned char *s, unsigned char
*e)` => `(int, unsigned int cpt)`
37878: (4)              """
37879: (4)              return _mupdf.pdf_decode_cmap(cmap, s, e, cpt)
37880: (0)          def pdf_deep_copy_obj(obj):
37881: (4)              r""" Class-aware wrapper for `::pdf_deep_copy_obj()`."""
37882: (4)              return _mupdf.pdf_deep_copy_obj(obj)
37883: (0)          def pdf_delete_annot(page, annot):
37884: (4)              r""" Class-aware wrapper for `::pdf_delete_annot()`."""
37885: (4)              return _mupdf.pdf_delete_annot(page, annot)
37886: (0)          def pdf_delete_link(page, link):
37887: (4)              r""" Class-aware wrapper for `::pdf_delete_link()`."""
37888: (4)              return _mupdf.pdf_delete_link(page, link)
37889: (0)          def pdf_delete_object(doc, num):
```

```
37890: (4)             r""" Class-aware wrapper for `::pdf_delete_object()`."""
37891: (4)             return _mupdf.pdf_delete_object(doc, num)
37892: (0)         def pdf_delete_page(doc, number):
37893: (4)             r""" Class-aware wrapper for `::pdf_delete_page()`."""
37894: (4)             return _mupdf.pdf_delete_page(doc, number)
37895: (0)         def pdf_delete_page_labels(doc, index):
37896: (4)             r""" Class-aware wrapper for `::pdf_delete_page_labels()`."""
37897: (4)             return _mupdf.pdf_delete_page_labels(doc, index)
37898: (0)         def pdf_delete_page_range(doc, start, end):
37899: (4)             r""" Class-aware wrapper for `::pdf_delete_page_range()`."""
37900: (4)             return _mupdf.pdf_delete_page_range(doc, start, end)
37901: (0)         def pdf_deselect_layer_config_ui(doc, ui):
37902: (4)             r""" Class-aware wrapper for `::pdf_deselect_layer_config_ui()`."""
37903: (4)             return _mupdf.pdf_deselect_layer_config_ui(doc, ui)
37904: (0)         def pdf_deserialise_journal(doc, stm):
37905: (4)             r""" Class-aware wrapper for `::pdf_deserialise_journal()`."""
37906: (4)             return _mupdf.pdf_deserialise_journal(doc, stm)
37907: (0)         def pdf_dict_del(dict, key):
37908: (4)             r""" Class-aware wrapper for `::pdf_dict_del()`."""
37909: (4)             return _mupdf.pdf_dict_del(dict, key)
37910: (0)         def pdf_dict_dels(dict, key):
37911: (4)             r""" Class-aware wrapper for `::pdf_dict_dels()`."""
37912: (4)             return _mupdf.pdf_dict_dels(dict, key)
37913: (0)         def pdf_dict_get(dict, key):
37914: (4)             r""" Class-aware wrapper for `::pdf_dict_get()`."""
37915: (4)             return _mupdf.pdf_dict_get(dict, key)
37916: (0)         def pdf_dict_get_bool(dict, key):
37917: (4)             r""" Class-aware wrapper for `::pdf_dict_get_bool()`."""
37918: (4)             return _mupdf.pdf_dict_get_bool(dict, key)
37919: (0)         def pdf_dict_get_bool_default(dict, key, _def):
37920: (4)             r""" Class-aware wrapper for `::pdf_dict_get_bool_default()`."""
37921: (4)             return _mupdf.pdf_dict_get_bool_default(dict, key, _def)
37922: (0)         def pdf_dict_get_date(dict, key):
37923: (4)             r""" Class-aware wrapper for `::pdf_dict_get_date()`."""
37924: (4)             return _mupdf.pdf_dict_get_date(dict, key)
37925: (0)         def pdf_dict_get_inheritable(dict, key):
37926: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable()`."""
37927: (4)             return _mupdf.pdf_dict_get_inheritable(dict, key)
37928: (0)         def pdf_dict_get_inheritable_bool(dict, key):
37929: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_bool()`."""
37930: (4)             return _mupdf.pdf_dict_get_inheritable_bool(dict, key)
37931: (0)         def pdf_dict_get_inheritable_date(dict, key):
37932: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_date()`."""
37933: (4)             return _mupdf.pdf_dict_get_inheritable_date(dict, key)
37934: (0)         def pdf_dict_get_inheritable_int(dict, key):
37935: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_int()`."""
37936: (4)             return _mupdf.pdf_dict_get_inheritable_int(dict, key)
37937: (0)         def pdf_dict_get_inheritable_int64(dict, key):
37938: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_int64()`."""
37939: (4)             return _mupdf.pdf_dict_get_inheritable_int64(dict, key)
37940: (0)         def pdf_dict_get_inheritable_matrix(dict, key):
37941: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_matrix()`."""
37942: (4)             return _mupdf.pdf_dict_get_inheritable_matrix(dict, key)
37943: (0)         def pdf_dict_get_inheritable_name(dict, key):
37944: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_name()`."""
37945: (4)             return _mupdf.pdf_dict_get_inheritable_name(dict, key)
37946: (0)         def pdf_dict_get_inheritable_real(dict, key):
37947: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_real()`."""
37948: (4)             return _mupdf.pdf_dict_get_inheritable_real(dict, key)
37949: (0)         def pdf_dict_get_inheritable_rect(dict, key):
37950: (4)             r""" Class-aware wrapper for `::pdf_dict_get_inheritable_rect()`."""
37951: (4)             return _mupdf.pdf_dict_get_inheritable_rect(dict, key)
37952: (0)         def pdf_dict_get_inheritable_string(dict, key, sizep):
37953: (4)             r"""
37954: (4)             Class-aware wrapper for `::pdf_dict_get_inheritable_string()`.
37955: (4)             This function has out-params. Python/C# wrappers look like:
37956: (8)                 `pdf_dict_get_inheritable_string(::pdf_obj *dict, ::pdf_obj *key)` =>
`(const char *, size_t sizep)`
37957: (4)             """
```

```
37958: (4)             return _mupdf.pdf_dict_get_inheritable_string(dict, key, sizep)
37959: (0)         def pdf_dict_get_inheritable_text_string(dict, key):
37960: (4)             r""" Class-aware wrapper for
`::pdf_dict_get_inheritable_text_string()`."""
37961: (4)             return _mupdf.pdf_dict_get_inheritable_text_string(dict, key)
37962: (0)         def pdf_dict_get_int(dict, key):
37963: (4)             r""" Class-aware wrapper for `::pdf_dict_get_int()`."""
37964: (4)             return _mupdf.pdf_dict_get_int(dict, key)
37965: (0)         def pdf_dict_get_int64(dict, key):
37966: (4)             r""" Class-aware wrapper for `::pdf_dict_get_int64()`."""
37967: (4)             return _mupdf.pdf_dict_get_int64(dict, key)
37968: (0)         def pdf_dict_get_int_default(dict, key, _def):
37969: (4)             r""" Class-aware wrapper for `::pdf_dict_get_int_default()`."""
37970: (4)             return _mupdf.pdf_dict_get_int_default(dict, key, _def)
37971: (0)         def pdf_dict_get_key(dict, idx):
37972: (4)             r""" Class-aware wrapper for `::pdf_dict_get_key()`."""
37973: (4)             return _mupdf.pdf_dict_get_key(dict, idx)
37974: (0)         def pdf_dict_get_matrix(dict, key):
37975: (4)             r""" Class-aware wrapper for `::pdf_dict_get_matrix()`."""
37976: (4)             return _mupdf.pdf_dict_get_matrix(dict, key)
37977: (0)         def pdf_dict_get_name(dict, key):
37978: (4)             r""" Class-aware wrapper for `::pdf_dict_get_name()`."""
37979: (4)             return _mupdf.pdf_dict_get_name(dict, key)
37980: (0)         def pdf_dict_get_real(dict, key):
37981: (4)             r""" Class-aware wrapper for `::pdf_dict_get_real()`."""
37982: (4)             return _mupdf.pdf_dict_get_real(dict, key)
37983: (0)         def pdf_dict_get_real_default(dict, key, _def):
37984: (4)             r""" Class-aware wrapper for `::pdf_dict_get_real_default()`."""
37985: (4)             return _mupdf.pdf_dict_get_real_default(dict, key, _def)
37986: (0)         def pdf_dict_get_rect(dict, key):
37987: (4)             r""" Class-aware wrapper for `::pdf_dict_get_rect()`."""
37988: (4)             return _mupdf.pdf_dict_get_rect(dict, key)
37989: (0)         def pdf_dict_get_string(dict, key, sizep):
37990: (4)             r"""
37991: (4)             Class-aware wrapper for `::pdf_dict_get_string()`.
37992: (4)             This function has out-params. Python/C# wrappers look like:
37993: (8)                 `pdf_dict_get_string(::pdf_obj *dict, ::pdf_obj *key)` => `(const char
*, size_t sizep)`
37994: (4)             """
37995: (4)             return _mupdf.pdf_dict_get_string(dict, key, sizep)
37996: (0)         def pdf_dict_get_text_string(dict, key):
37997: (4)             r""" Class-aware wrapper for `::pdf_dict_get_text_string()`."""
37998: (4)             return _mupdf.pdf_dict_get_text_string(dict, key)
37999: (0)         def pdf_dict_get_val(dict, idx):
38000: (4)             r""" Class-aware wrapper for `::pdf_dict_get_val()`."""
38001: (4)             return _mupdf.pdf_dict_get_val(dict, idx)
38002: (0)         def pdf_dict_geta(dict, key, abbrev):
38003: (4)             r""" Class-aware wrapper for `::pdf_dict_geta()`."""
38004: (4)             return _mupdf.pdf_dict_geta(dict, key, abbrev)
38005: (0)         def pdf_dict_getp(dict, path):
38006: (4)             r""" Class-aware wrapper for `::pdf_dict_getp()`."""
38007: (4)             return _mupdf.pdf_dict_getp(dict, path)
38008: (0)         def pdf_dict_getp_inheritable(dict, path):
38009: (4)             r""" Class-aware wrapper for `::pdf_dict_getp_inheritable()`."""
38010: (4)             return _mupdf.pdf_dict_getp_inheritable(dict, path)
38011: (0)         def pdf_dict_gets(dict, key):
38012: (4)             r""" Class-aware wrapper for `::pdf_dict_gets()`."""
38013: (4)             return _mupdf.pdf_dict_gets(dict, key)
38014: (0)         def pdf_dict_gets_inheritable(dict, key):
38015: (4)             r""" Class-aware wrapper for `::pdf_dict_gets_inheritable()`."""
38016: (4)             return _mupdf.pdf_dict_gets_inheritable(dict, key)
38017: (0)         def pdf_dict_getsa(dict, key, abbrev):
38018: (4)             r""" Class-aware wrapper for `::pdf_dict_getsa()`."""
38019: (4)             return _mupdf.pdf_dict_getsa(dict, key, abbrev)
38020: (0)         def pdf_dict_len(dict):
38021: (4)             r""" Class-aware wrapper for `::pdf_dict_len()`."""
38022: (4)             return _mupdf.pdf_dict_len(dict)
38023: (0)         def pdf_dict_put(dict, key, val):
38024: (4)             r""" Class-aware wrapper for `::pdf_dict_put()`."""
```

```
(4)             return _mupdf.pdf_dict_put(dict, key, val)
(0)         def pdf_dict_put_array(dict, key, initial):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_array()`."""
(4)             return _mupdf.pdf_dict_put_array(dict, key, initial)
(0)         def pdf_dict_put_bool(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_bool()`."""
(4)             return _mupdf.pdf_dict_put_bool(dict, key, x)
(0)         def pdf_dict_put_date(dict, key, time):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_date()`."""
(4)             return _mupdf.pdf_dict_put_date(dict, key, time)
(0)         def pdf_dict_put_dict(dict, key, initial):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_dict()`."""
(4)             return _mupdf.pdf_dict_put_dict(dict, key, initial)
(0)         def pdf_dict_put_int(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_int()`."""
(4)             return _mupdf.pdf_dict_put_int(dict, key, x)
(0)         def pdf_dict_put_matrix(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_matrix()`."""
(4)             return _mupdf.pdf_dict_put_matrix(dict, key, x)
(0)         def pdf_dict_put_name(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_name()`."""
(4)             return _mupdf.pdf_dict_put_name(dict, key, x)
(0)         def pdf_dict_put_real(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_real()`."""
(4)             return _mupdf.pdf_dict_put_real(dict, key, x)
(0)         def pdf_dict_put_rect(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_rect()`."""
(4)             return _mupdf.pdf_dict_put_rect(dict, key, x)
(0)         def pdf_dict_put_string(dict, key, x, n):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_string()`."""
(4)             return _mupdf.pdf_dict_put_string(dict, key, x, n)
(0)         def pdf_dict_put_text_string(dict, key, x):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_text_string()`."""
(4)             return _mupdf.pdf_dict_put_text_string(dict, key, x)
(0)         def pdf_dict_put_val_null(obj, idx):
(4)             r""" Class-aware wrapper for `::pdf_dict_put_val_null()`."""
(4)             return _mupdf.pdf_dict_put_val_null(obj, idx)
(0)         def pdf_dict_putp(dict, path, val):
(4)             r""" Class-aware wrapper for `::pdf_dict_putp()`."""
(4)             return _mupdf.pdf_dict_putp(dict, path, val)
(0)         def pdf_dict_puts(dict, key, val):
(4)             r""" Class-aware wrapper for `::pdf_dict_puts()`."""
(4)             return _mupdf.pdf_dict_puts(dict, key, val)
(0)         def pdf_dict_puts_dict(dict, key, initial):
(4)             r""" Class-aware wrapper for `::pdf_dict_puts_dict()`."""
(4)             return _mupdf.pdf_dict_puts_dict(dict, key, initial)
(0)         def pdf_dirty_annot(annot):
(4)             r""" Class-aware wrapper for `::pdf_dirty_annot()`."""
(4)             return _mupdf.pdf_dirty_annot(annot)
(0)         def pdf_dirty_obj(obj):
(4)             r""" Class-aware wrapper for `::pdf_dirty_obj()`."""
(4)             return _mupdf.pdf_dirty_obj(obj)
(0)         def pdf_disable_js(doc):
(4)             r""" Class-aware wrapper for `::pdf_disable_js()`."""
(4)             return _mupdf.pdf_disable_js(doc)
(0)         def pdf_discard_journal(journal):
(4)             r""" Class-aware wrapper for `::pdf_discard_journal()`."""
(4)             return _mupdf.pdf_discard_journal(journal)
(0)         def pdf_doc_was_linearized(doc):
(4)             r""" Class-aware wrapper for `::pdf_doc_was_linearized()`."""
(4)             return _mupdf.pdf_doc_was_linearized(doc)
(0)         def pdf_document_event_did_print(doc):
(4)             r""" Class-aware wrapper for `::pdf_document_event_did_print()`."""
(4)             return _mupdf.pdf_document_event_did_print(doc)
(0)         def pdf_document_event_did_save(doc):
(4)             r""" Class-aware wrapper for `::pdf_document_event_did_save()`."""
(4)             return _mupdf.pdf_document_event_did_save(doc)
(0)         def pdf_document_event_will_close(doc):
(4)             r""" Class-aware wrapper for `::pdf_document_event_will_close()`."""
```

```
38094: (4)             return _mupdf.pdf_document_event_will_close(doc)
38095: (0)         def pdf_document_event_will_print(doc):
38096: (4)             r""" Class-aware wrapper for `::pdf_document_event_will_print()`."""
38097: (4)             return _mupdf.pdf_document_event_will_print(doc)
38098: (0)         def pdf_document_event_will_save(doc):
38099: (4)             r""" Class-aware wrapper for `::pdf_document_event_will_save()`."""
38100: (4)             return _mupdf.pdf_document_event_will_save(doc)
38101: (0)         def pdf_document_from_fz_document(ptr):
38102: (4)             r""" Class-aware wrapper for `::pdf_document_from_fz_document()`."""
38103: (4)             return _mupdf.pdf_document_from_fz_document(ptr)
38104: (0)         def pdf_document_language(doc):
38105: (4)             r""" Class-aware wrapper for `::pdf_document_language()`."""
38106: (4)             return _mupdf.pdf_document_language(doc)
38107: (0)         def pdf_document_output_intent(doc):
38108: (4)             r""" Class-aware wrapper for `::pdf_document_output_intent()`."""
38109: (4)             return _mupdf.pdf_document_output_intent(doc)
38110: (0)         def pdf_document_permissions(doc):
38111: (4)             r""" Class-aware wrapper for `::pdf_document_permissions()`."""
38112: (4)             return _mupdf.pdf_document_permissions(doc)
38113: (0)         def pdf_edit_text_field_value(widget, value, change, selStart, selEnd,
newvalue):
38114: (4)             r"""
38115: (4)             Class-aware wrapper for `::pdf_edit_text_field_value()`.
38116: (4)             This function has out-params. Python/C# wrappers look like:
38117: (8)                 `pdf_edit_text_field_value(::pdf_annot *widget, const char *value,
const char *change)` => `(int, int selStart, int selEnd, char *newvalue)`
38118: (4)             """
38119: (4)             return _mupdf.pdf_edit_text_field_value(widget, value, change, selStart,
selEnd, newvalue)
38120: (0)         def pdf_empty_store(doc):
38121: (4)             r""" Class-aware wrapper for `::pdf_empty_store()`."""
38122: (4)             return _mupdf.pdf_empty_store(doc)
38123: (0)         def pdf_enable_journal(doc):
38124: (4)             r""" Class-aware wrapper for `::pdf_enable_journal()`."""
38125: (4)             return _mupdf.pdf_enable_journal(doc)
38126: (0)         def pdf_enable_js(doc):
38127: (4)             r""" Class-aware wrapper for `::pdf_enable_js()`."""
38128: (4)             return _mupdf.pdf_enable_js(doc)
38129: (0)         def pdf_enable_layer(doc, layer, enabled):
38130: (4)             r""" Class-aware wrapper for `::pdf_enable_layer()`."""
38131: (4)             return _mupdf.pdf_enable_layer(doc, layer, enabled)
38132: (0)         def pdf_encrypt_data(crypt, num, gen, fmt_str_out, arg, s, n):
38133: (4)             r""" Class-aware wrapper for `::pdf_encrypt_data()`."""
38134: (4)             return _mupdf.pdf_encrypt_data(crypt, num, gen, fmt_str_out, arg, s, n)
38135: (0)         def pdf_encrypted_len(crypt, num, gen, len):
38136: (4)             r""" Class-aware wrapper for `::pdf_encrypted_len()`."""
38137: (4)             return _mupdf.pdf_encrypted_len(crypt, num, gen, len)
38138: (0)         def pdf_end_hmtx(font):
38139: (4)             r""" Class-aware wrapper for `::pdf_end_hmtx()`."""
38140: (4)             return _mupdf.pdf_end_hmtx(font)
38141: (0)         def pdf_end_operation(doc):
38142: (4)             r""" Class-aware wrapper for `::pdf_end_operation()`."""
38143: (4)             return _mupdf.pdf_end_operation(doc)
38144: (0)         def pdf_end_vmtx(font):
38145: (4)             r""" Class-aware wrapper for `::pdf_end_vmtx()`."""
38146: (4)             return _mupdf.pdf_end_vmtx(font)
38147: (0)         def pdf_ensure_solid_xref(doc, num):
38148: (4)             r""" Class-aware wrapper for `::pdf_ensure_solid_xref()`."""
38149: (4)             return _mupdf.pdf_ensure_solid_xref(doc, num)
38150: (0)         def pdf_eval_function(func, _in, inlen, out, outlen):
38151: (4)             r"""
38152: (4)             Class-aware wrapper for `::pdf_eval_function()`.
38153: (4)             This function has out-params. Python/C# wrappers look like:
38154: (8)                 `pdf_eval_function(::pdf_function *func, const float *in, int inlen,
int outlen)` => float out
38155: (4)             """
38156: (4)             return _mupdf.pdf_eval_function(func, _in, inlen, out, outlen)
38157: (0)         def pdf_event_issue_alert(doc, evt):
38158: (4)             r""" Class-aware wrapper for `::pdf_event_issue_alert()`."""
```

```
38159: (4)             return _mupdf.pdf_event_issue_alert(doc, evt)
38160: (0)         def pdf_event_issue_exec_menu_item(doc, item):
38161: (4)             r""" Class-aware wrapper for `::pdf_event_issue_exec_menu_item()`."""
38162: (4)             return _mupdf.pdf_event_issue_exec_menu_item(doc, item)
38163: (0)         def pdf_event_issue_launch_url(doc, url, new_frame):
38164: (4)             r""" Class-aware wrapper for `::pdf_event_issue_launch_url()`."""
38165: (4)             return _mupdf.pdf_event_issue_launch_url(doc, url, new_frame)
38166: (0)         def pdf_event_issue_mail_doc(doc, evt):
38167: (4)             r""" Class-aware wrapper for `::pdf_event_issue_mail_doc()`."""
38168: (4)             return _mupdf.pdf_event_issue_mail_doc(doc, evt)
38169: (0)         def pdf_event_issue_print(doc):
38170: (4)             r""" Class-aware wrapper for `::pdf_event_issue_print()`."""
38171: (4)             return _mupdf.pdf_event_issue_print(doc)
38172: (0)         def pdf_field_border_style(field):
38173: (4)             r""" Class-aware wrapper for `::pdf_field_border_style()`."""
38174: (4)             return _mupdf.pdf_field_border_style(field)
38175: (0)         def pdf_field_display(field):
38176: (4)             r""" Class-aware wrapper for `::pdf_field_display()`."""
38177: (4)             return _mupdf.pdf_field_display(field)
38178: (0)         def pdf_field_event_calculate(doc, field):
38179: (4)             r""" Class-aware wrapper for `::pdf_field_event_calculate()`."""
38180: (4)             return _mupdf.pdf_field_event_calculate(doc, field)
38181: (0)         def pdf_field_event_format(doc, field):
38182: (4)             r""" Class-aware wrapper for `::pdf_field_event_format()`."""
38183: (4)             return _mupdf.pdf_field_event_format(doc, field)
38184: (0)         def pdf_field_event_keystroke(doc, field, evt):
38185: (4)             r""" Class-aware wrapper for `::pdf_field_event_keystroke()`."""
38186: (4)             return _mupdf.pdf_field_event_keystroke(doc, field, evt)
38187: (0)         def pdf_field_event_validate(doc, field, value, newvalue):
38188: (4)             r"""
38189: (4)             Class-aware wrapper for `::pdf_field_event_validate()`.
38190: (4)             This function has out-params. Python/C# wrappers look like:
38191: (8)                 `pdf_field_event_validate(::pdf_document *doc, ::pdf_obj *field, const
char *value)` => `(int, char *newvalue)`
38192: (4)             """
38193: (4)             return _mupdf.pdf_field_event_validate(doc, field, value, newvalue)
38194: (0)         def pdf_field_flags(field):
38195: (4)             r""" Class-aware wrapper for `::pdf_field_flags()`."""
38196: (4)             return _mupdf.pdf_field_flags(field)
38197: (0)         def pdf_field_label(field):
38198: (4)             r""" Class-aware wrapper for `::pdf_field_label()`."""
38199: (4)             return _mupdf.pdf_field_label(field)
38200: (0)         def pdf_field_reset(doc, field):
38201: (4)             r""" Class-aware wrapper for `::pdf_field_reset()`."""
38202: (4)             return _mupdf.pdf_field_reset(doc, field)
38203: (0)         def pdf_field_set_border_style(field, text):
38204: (4)             r""" Class-aware wrapper for `::pdf_field_set_border_style()`."""
38205: (4)             return _mupdf.pdf_field_set_border_style(field, text)
38206: (0)         def pdf_field_set_button_caption(field, text):
38207: (4)             r""" Class-aware wrapper for `::pdf_field_set_button_caption()`."""
38208: (4)             return _mupdf.pdf_field_set_button_caption(field, text)
38209: (0)         def pdf_field_set_display(field, d):
38210: (4)             r""" Class-aware wrapper for `::pdf_field_set_display()`."""
38211: (4)             return _mupdf.pdf_field_set_display(field, d)
38212: (0)         def pdf_field_set_fill_color(field, col):
38213: (4)             r""" Class-aware wrapper for `::pdf_field_set_fill_color()`."""
38214: (4)             return _mupdf.pdf_field_set_fill_color(field, col)
38215: (0)         def pdf_field_set_text_color(field, col):
38216: (4)             r""" Class-aware wrapper for `::pdf_field_set_text_color()`."""
38217: (4)             return _mupdf.pdf_field_set_text_color(field, col)
38218: (0)         def pdf_field_type(field):
38219: (4)             r""" Class-aware wrapper for `::pdf_field_type()`."""
38220: (4)             return _mupdf.pdf_field_type(field)
38221: (0)         def pdf_field_type_string(field):
38222: (4)             r""" Class-aware wrapper for `::pdf_field_type_string()`."""
38223: (4)             return _mupdf.pdf_field_type_string(field)
38224: (0)         def pdf_field_value(field):
38225: (4)             r""" Class-aware wrapper for `::pdf_field_value()`."""
38226: (4)             return _mupdf.pdf_field_value(field)
```

```
38227: (0)          def pdf_filter_annot_contents(doc, annot, options):
38228: (4)              r""" Class-aware wrapper for `::pdf_filter_annot_contents()`."""
38229: (4)              return _mupdf.pdf_filter_annot_contents(doc, annot, options)
38230: (0)          def pdf_filter_page_contents(doc, page, options):
38231: (4)              r""" Class-aware wrapper for `::pdf_filter_page_contents()`."""
38232: (4)              return _mupdf.pdf_filter_page_contents(doc, page, options)
38233: (0)          def pdf_filter_xobject_instance(old_xobj, page_res, ctm, options, cycle_up):
38234: (4)              r""" Class-aware wrapper for `::pdf_filter_xobject_instance()`."""
38235: (4)              return _mupdf.pdf_filter_xobject_instance(old_xobj, page_res, ctm,
options, cycle_up)
38236: (0)          def pdf_find_font_resource(doc, type, encoding, item, key):
38237: (4)              r""" Class-aware wrapper for `::pdf_find_font_resource()`."""
38238: (4)              return _mupdf.pdf_find_font_resource(doc, type, encoding, item, key)
38239: (0)          def pdf_find_item(drop, key):
38240: (4)              r""" Class-aware wrapper for `::pdf_find_item()`."""
38241: (4)              return _mupdf.pdf_find_item(drop, key)
38242: (0)          def pdf_find_version_for_obj(doc, obj):
38243: (4)              r""" Class-aware wrapper for `::pdf_find_version_for_obj()`."""
38244: (4)              return _mupdf.pdf_find_version_for_obj(doc, obj)
38245: (0)          def pdf_first_annot(page):
38246: (4)              r""" Class-aware wrapper for `::pdf_first_annot()`."""
38247: (4)              return _mupdf.pdf_first_annot(page)
38248: (0)          def pdf_first_widget(page):
38249: (4)              r""" Class-aware wrapper for `::pdf_first_widget()`."""
38250: (4)              return _mupdf.pdf_first_widget(page)
38251: (0)          def pdf_flatten_inheritable_page_items(page):
38252: (4)              r""" Class-aware wrapper for `::pdf_flatten_inheritable_page_items()`."""
38253: (4)              return _mupdf.pdf_flatten_inheritable_page_items(page)
38254: (0)          def pdf_font_cid_to_gid(fontdesc, cid):
38255: (4)              r""" Class-aware wrapper for `::pdf_font_cid_to_gid()`."""
38256: (4)              return _mupdf.pdf_font_cid_to_gid(fontdesc, cid)
38257: (0)          def pdf_font_writing_supported(font):
38258: (4)              r""" Class-aware wrapper for `::pdf_font_writing_supported()`."""
38259: (4)              return _mupdf.pdf_font_writing_supported(font)
38260: (0)          def pdf_forget_xref(doc):
38261: (4)              r""" Class-aware wrapper for `::pdf_forget_xref()`."""
38262: (4)              return _mupdf.pdf_forget_xref(doc)
38263: (0)          def pdf_format_date(time, s, n):
38264: (4)              r""" Class-aware wrapper for `::pdf_format_date()`."""
38265: (4)              return _mupdf.pdf_format_date(time, s, n)
38266: (0)          def pdf_format_write_options(buffer, buffer_len, opts):
38267: (4)              r""" Class-aware wrapper for `::pdf_format_write_options()`."""
38268: (4)              return _mupdf.pdf_format_write_options(buffer, buffer_len, opts)
38269: (0)          def pdf_function_size(func):
38270: (4)              r""" Class-aware wrapper for `::pdf_function_size()`."""
38271: (4)              return _mupdf.pdf_function_size(func)
38272: (0)          def pdf_get_bound_document(obj):
38273: (4)              r""" Class-aware wrapper for `::pdf_get_bound_document()`."""
38274: (4)              return _mupdf.pdf_get_bound_document(obj)
38275: (0)          def pdf_get_doc_event_callback_data(doc):
38276: (4)              r""" Class-aware wrapper for `::pdf_get_doc_event_callback_data()`."""
38277: (4)              return _mupdf.pdf_get_doc_event_callback_data(doc)
38278: (0)          def pdf_get_embedded_file_params(fs, out):
38279: (4)              r""" Class-aware wrapper for `::pdf_get_embedded_file_params()`."""
38280: (4)              return _mupdf.pdf_get_embedded_file_params(fs, out)
38281: (0)          def pdf_get_indirect_document(obj):
38282: (4)              r""" Class-aware wrapper for `::pdf_get_indirect_document()`."""
38283: (4)              return _mupdf.pdf_get_indirect_document(obj)
38284: (0)          def pdf_get_widget_editing_state(widget):
38285: (4)              r""" Class-aware wrapper for `::pdf_get_widget_editing_state()`."""
38286: (4)              return _mupdf.pdf_get_widget_editing_state(widget)
38287: (0)          def pdf_graft_mapped_object(map, obj):
38288: (4)              r""" Class-aware wrapper for `::pdf_graft_mapped_object()`."""
38289: (4)              return _mupdf.pdf_graft_mapped_object(map, obj)
38290: (0)          def pdf_graft_mapped_page(map, page_to, src, page_from):
38291: (4)              r""" Class-aware wrapper for `::pdf_graft_mapped_page()`."""
38292: (4)              return _mupdf.pdf_graft_mapped_page(map, page_to, src, page_from)
38293: (0)          def pdf_graft_object(dst, obj):
38294: (4)              r""" Class-aware wrapper for `::pdf_graft_object()`."""
```

```
(4)             return _mupdf.pdf_graft_object(dst, obj)
(0)         def pdf_graft_page(dst, page_to, src, page_from):
(4)             r""" Class-aware wrapper for `::pdf_graft_page()`."""
(4)             return _mupdf.pdf_graft_page(dst, page_to, src, page_from)
(0)         def pdf_has_permission(doc, p):
(4)             r""" Class-aware wrapper for `::pdf_has_permission()`."""
(4)             return _mupdf.pdf_has_permission(doc, p)
(0)         def pdf_has_unsaved_changes(doc):
(4)             r""" Class-aware wrapper for `::pdf_has_unsaved_changes()`."""
(4)             return _mupdf.pdf_has_unsaved_changes(doc)
(0)         def pdf_has_unsaved_sigs(doc):
(4)             r""" Class-aware wrapper for `::pdf_has_unsaved_sigs()`."""
(4)             return _mupdf.pdf_has_unsaved_sigs(doc)
(0)         def pdf_insert_font_resource(doc, key, obj):
(4)             r""" Class-aware wrapper for `::pdf_insert_font_resource()`."""
(4)             return _mupdf.pdf_insert_font_resource(doc, key, obj)
(0)         def pdf_insert_page(doc, at, page):
(4)             r""" Class-aware wrapper for `::pdf_insert_page()`."""
(4)             return _mupdf.pdf_insert_page(doc, at, page)
(0)         def pdf_intent_from_name(obj):
(4)             r""" Class-aware wrapper for `::pdf_intent_from_name()`."""
(4)             return _mupdf.pdf_intent_from_name(obj)
(0)         def pdf_intent_from_string(str):
(4)             r""" Class-aware wrapper for `::pdf_intent_from_string()`."""
(4)             return _mupdf.pdf_intent_from_string(str)
(0)         def pdf_invalidate_xfa(doc):
(4)             r""" Class-aware wrapper for `::pdf_invalidate_xfa()`."""
(4)             return _mupdf.pdf_invalidate_xfa(doc)
(0)         def pdf_is_array(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_array()`."""
(4)             return _mupdf.pdf_is_array(obj)
(0)         def pdf_is_bool(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_bool()`."""
(4)             return _mupdf.pdf_is_bool(obj)
(0)         def pdf_is_dict(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_dict()`."""
(4)             return _mupdf.pdf_is_dict(obj)
(0)         def pdf_is_embedded_file(fs):
(4)             r""" Class-aware wrapper for `::pdf_is_embedded_file()`."""
(4)             return _mupdf.pdf_is_embedded_file(fs)
(0)         def pdf_is_field_locked(locked, name):
(4)             r""" Class-aware wrapper for `::pdf_is_field_locked()`."""
(4)             return _mupdf.pdf_is_field_locked(locked, name)
(0)         def pdf_is_indirect(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_indirect()`."""
(4)             return _mupdf.pdf_is_indirect(obj)
(0)         def pdf_is_int(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_int()`."""
(4)             return _mupdf.pdf_is_int(obj)
(0)         def pdf_is_jpx_image(dict):
(4)             r""" Class-aware wrapper for `::pdf_is_jpx_image()`."""
(4)             return _mupdf.pdf_is_jpx_image(dict)
(0)         def pdf_is_local_object(doc, obj):
(4)             r""" Class-aware wrapper for `::pdf_is_local_object()`."""
(4)             return _mupdf.pdf_is_local_object(doc, obj)
(0)         def pdf_is_name(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_name()`."""
(4)             return _mupdf.pdf_is_name(obj)
(0)         def pdf_is_null(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_null()`."""
(4)             return _mupdf.pdf_is_null(obj)
(0)         def pdf_is_number(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_number()`."""
(4)             return _mupdf.pdf_is_number(obj)
(0)         def pdf_is_ocg_hidden(doc, rdb, usage, ocg):
(4)             r""" Class-aware wrapper for `::pdf_is_ocg_hidden()`."""
(4)             return _mupdf.pdf_is_ocg_hidden(doc, rdb, usage, ocg)
(0)         def pdf_is_real(obj):
(4)             r""" Class-aware wrapper for `::pdf_is_real()`."""
```

```
38364: (4)             return _mupdf.pdf_is_real(obj)
38365: (0)         def pdf_is_stream(obj):
38366: (4)             r""" Class-aware wrapper for `::pdf_is_stream()`."""
38367: (4)             return _mupdf.pdf_is_stream(obj)
38368: (0)         def pdf_is_string(obj):
38369: (4)             r""" Class-aware wrapper for `::pdf_is_string()`."""
38370: (4)             return _mupdf.pdf_is_string(obj)
38371: (0)         def pdf_is_tint_colorspace(cs):
38372: (4)             r""" Class-aware wrapper for `::pdf_is_tint_colorspace()`."""
38373: (4)             return _mupdf.pdf_is_tint_colorspace(cs)
38374: (0)         def pdf_js_event_init(js, target, value, willCommit):
38375: (4)             r""" Class-aware wrapper for `::pdf_js_event_init()`."""
38376: (4)             return _mupdf.pdf_js_event_init(js, target, value, willCommit)
38377: (0)         def pdf_js_event_init_keystroke(js, target, evt):
38378: (4)             r""" Class-aware wrapper for `::pdf_js_event_init_keystroke()`."""
38379: (4)             return _mupdf.pdf_js_event_init_keystroke(js, target, evt)
38380: (0)         def pdf_js_event_result(js):
38381: (4)             r""" Class-aware wrapper for `::pdf_js_event_result()`."""
38382: (4)             return _mupdf.pdf_js_event_result(js)
38383: (0)         def pdf_js_event_result_keystroke(js, evt):
38384: (4)             r""" Class-aware wrapper for `::pdf_js_event_result_keystroke()`."""
38385: (4)             return _mupdf.pdf_js_event_result_keystroke(js, evt)
38386: (0)         def pdf_js_event_result_validate(js, newvalue):
38387: (4)             r"""
38388: (4)             Class-aware wrapper for `::pdf_js_event_result_validate()`.
38389: (4)             This function has out-params. Python/C# wrappers look like:
38390: (8)                 `pdf_js_event_result_validate(::pdf_js *js)` => `(int, char
*newvalue)`
38391: (4)             """
38392: (4)             return _mupdf.pdf_js_event_result_validate(js, newvalue)
38393: (0)         def pdf_js_event_value(js):
38394: (4)             r""" Class-aware wrapper for `::pdf_js_event_value()`."""
38395: (4)             return _mupdf.pdf_js_event_value(js)
38396: (0)         def pdf_js_execute(js, name, code, result):
38397: (4)             r"""
38398: (4)             Class-aware wrapper for `::pdf_js_execute()`.
38399: (4)             This function has out-params. Python/C# wrappers look like:
38400: (8)                 `pdf_js_execute(::pdf_js *js, const char *name, const char *code)` =>
char *result
38401: (4)             """
38402: (4)             return _mupdf.pdf_js_execute(js, name, code, result)
38403: (0)         def pdf_js_set_console(doc, console, user):
38404: (4)             r""" Class-aware wrapper for `::pdf_js_set_console()`."""
38405: (4)             return _mupdf.pdf_js_set_console(doc, console, user)
38406: (0)         def pdf_js_supported(doc):
38407: (4)             r""" Class-aware wrapper for `::pdf_js_supported()`."""
38408: (4)             return _mupdf.pdf_js_supported(doc)
38409: (0)         def pdf_layer_config_info(doc, config_num, info):
38410: (4)             r""" Class-aware wrapper for `::pdf_layer_config_info()`."""
38411: (4)             return _mupdf.pdf_layer_config_info(doc, config_num, info)
38412: (0)         def pdf_layer_config_ui_info(doc, ui, info):
38413: (4)             r""" Class-aware wrapper for `::pdf_layer_config_ui_info()`."""
38414: (4)             return _mupdf.pdf_layer_config_ui_info(doc, ui, info)
38415: (0)         def pdf_layer_is_enabled(doc, layer):
38416: (4)             r""" Class-aware wrapper for `::pdf_layer_is_enabled()`."""
38417: (4)             return _mupdf.pdf_layer_is_enabled(doc, layer)
38418: (0)         def pdf_layer_name(doc, layer):
38419: (4)             r""" Class-aware wrapper for `::pdf_layer_name()`."""
38420: (4)             return _mupdf.pdf_layer_name(doc, layer)
38421: (0)         def pdf_layout_fit_text(font, lang, str, bounds):
38422: (4)             r""" Class-aware wrapper for `::pdf_layout_fit_text()`."""
38423: (4)             return _mupdf.pdf_layout_fit_text(font, lang, str, bounds)
38424: (0)         def pdf_lex(f, lexbuf):
38425: (4)             r""" Class-aware wrapper for `::pdf_lex()`."""
38426: (4)             return _mupdf.pdf_lex(f, lexbuf)
38427: (0)         def pdf_lex_no_string(f, lexbuf):
38428: (4)             r""" Class-aware wrapper for `::pdf_lex_no_string()`."""
38429: (4)             return _mupdf.pdf_lex_no_string(f, lexbuf)
38430: (0)         def pdf_lexbuf_fin(lexbuf):
```

```
38431: (4)             r""" Class-aware wrapper for `::pdf_lexbuf_fin()`."""
38432: (4)             return _mupdf.pdf_lexbuf_fin(lexbuf)
38433: (0)         def pdf_lexbuf_grow(lexbuf):
38434: (4)             r""" Class-aware wrapper for `::pdf_lexbuf_grow()`."""
38435: (4)             return _mupdf.pdf_lexbuf_grow(lexbuf)
38436: (0)         def pdf_lexbuf_init(lexbuf, size):
38437: (4)             r""" Class-aware wrapper for `::pdf_lexbuf_init()`."""
38438: (4)             return _mupdf.pdf_lexbuf_init(lexbuf, size)
38439: (0)         def pdf_line_ending_from_name(end):
38440: (4)             r""" Class-aware wrapper for `::pdf_line_ending_from_name()`."""
38441: (4)             return _mupdf.pdf_line_ending_from_name(end)
38442: (0)         def pdf_line_ending_from_string(end):
38443: (4)             r""" Class-aware wrapper for `::pdf_line_ending_from_string()`."""
38444: (4)             return _mupdf.pdf_line_ending_from_string(end)
38445: (0)         def pdf_load_builtin_cmap(name):
38446: (4)             r""" Class-aware wrapper for `::pdf_load_builtin_cmap()`."""
38447: (4)             return _mupdf.pdf_load_builtin_cmap(name)
38448: (0)         def pdf_load_cmap(file):
38449: (4)             r""" Class-aware wrapper for `::pdf_load_cmap()`."""
38450: (4)             return _mupdf.pdf_load_cmap(file)
38451: (0)         def pdf_load_colorspace(obj):
38452: (4)             r""" Class-aware wrapper for `::pdf_load_colorspace()`."""
38453: (4)             return _mupdf.pdf_load_colorspace(obj)
38454: (0)         def pdf_load_compressed_inline_image(doc, dict, length, cstm, indexed, image):
38455: (4)             r""" Class-aware wrapper for `::pdf_load_compressed_inline_image()`."""
38456: (4)             return _mupdf.pdf_load_compressed_inline_image(doc, dict, length, cstm,
indexed, image)
38457: (0)         def pdf_load_compressed_stream(doc, num, worst_case):
38458: (4)             r""" Class-aware wrapper for `::pdf_load_compressed_stream()`."""
38459: (4)             return _mupdf.pdf_load_compressed_stream(doc, num, worst_case)
38460: (0)         def pdf_load_default_colorspaces(doc, page):
38461: (4)             r""" Class-aware wrapper for `::pdf_load_default_colorspaces()`."""
38462: (4)             return _mupdf.pdf_load_default_colorspaces(doc, page)
38463: (0)         def pdf_load_embedded_cmap(doc, ref):
38464: (4)             r""" Class-aware wrapper for `::pdf_load_embedded_cmap()`."""
38465: (4)             return _mupdf.pdf_load_embedded_cmap(doc, ref)
38466: (0)         def pdf_load_embedded_file_contents(fs):
38467: (4)             r""" Class-aware wrapper for `::pdf_load_embedded_file_contents()`."""
38468: (4)             return _mupdf.pdf_load_embedded_file_contents(fs)
38469: (0)         def pdf_load_encoding(estrings, encoding):
38470: (4)             r"""
38471: (4)             Class-aware wrapper for `::pdf_load_encoding()`.
38472: (4)             This function has out-params. Python/C# wrappers look like:
38473: (8)                 `pdf_load_encoding(const char *encoding)` => const char *estrings
38474: (4)             """
38475: (4)             return _mupdf.pdf_load_encoding(estrings, encoding)
38476: (0)         def pdf_load_field_name(field):
38477: (4)             r""" Class-aware wrapper for `::pdf_load_field_name()`."""
38478: (4)             return _mupdf.pdf_load_field_name(field)
38479: (0)         def pdf_load_field_name2(field):
38480: (4)             r""" Alternative to `pdf_load_field_name()` that returns a std::string."""
38481: (4)             return _mupdf.pdf_load_field_name2(field)
38482: (0)         def pdf_load_function(ref, _in, out):
38483: (4)             r""" Class-aware wrapper for `::pdf_load_function()`."""
38484: (4)             return _mupdf.pdf_load_function(ref, _in, out)
38485: (0)         def pdf_load_image(doc, obj):
38486: (4)             r""" Class-aware wrapper for `::pdf_load_image()`."""
38487: (4)             return _mupdf.pdf_load_image(doc, obj)
38488: (0)         def pdf_load_inline_image(doc, rdb, dict, file):
38489: (4)             r""" Class-aware wrapper for `::pdf_load_inline_image()`."""
38490: (4)             return _mupdf.pdf_load_inline_image(doc, rdb, dict, file)
38491: (0)         def pdf_load_journal(doc, filename):
38492: (4)             r""" Class-aware wrapper for `::pdf_load_journal()`."""
38493: (4)             return _mupdf.pdf_load_journal(doc, filename)
38494: (0)         def pdf_load_link_annots(arg_0, arg_1, annots, pagenum, page_ctm):
38495: (4)             r""" Class-aware wrapper for `::pdf_load_link_annots()`."""
38496: (4)             return _mupdf.pdf_load_link_annots(arg_0, arg_1, annots, pagenum,
page_ctm)
38497: (0)         def pdf_load_links(page):
```

```
38498: (4)             r""" Class-aware wrapper for `::pdf_load_links()`."""
38499: (4)             return _mupdf.pdf_load_links(page)
38500: (0)         def pdf_load_name_tree(doc, which):
38501: (4)             r""" Class-aware wrapper for `::pdf_load_name_tree()`."""
38502: (4)             return _mupdf.pdf_load_name_tree(doc, which)
38503: (0)         def pdf_load_object(doc, num):
38504: (4)             r"""
38505: (4)             Class-aware wrapper for `::pdf_load_object()`.
38506: (8)                 Load a given object.
38507: (8)                 This can cause xref reorganisations (solidifications etc) due to
38508: (8)                 repairs, so all held pdf_xref_entries should be considered
38509: (8)                 invalid after this call (other than the returned one).
38510: (4)             """
38511: (4)             return _mupdf.pdf_load_object(doc, num)
38512: (0)         def pdf_load_outline(doc):
38513: (4)             r""" Class-aware wrapper for `::pdf_load_outline()`."""
38514: (4)             return _mupdf.pdf_load_outline(doc)
38515: (0)         def pdf_load_page(doc, number):
38516: (4)             r""" Class-aware wrapper for `::pdf_load_page()`."""
38517: (4)             return _mupdf.pdf_load_page(doc, number)
38518: (0)         def pdf_load_page_imp(doc, chapter, number):
38519: (4)             r""" Class-aware wrapper for `::pdf_load_page_imp()`."""
38520: (4)             return _mupdf.pdf_load_page_imp(doc, chapter, number)
38521: (0)         def pdf_load_page_tree(doc):
38522: (4)             r""" Class-aware wrapper for `::pdf_load_page_tree()`."""
38523: (4)             return _mupdf.pdf_load_page_tree(doc)
38524: (0)         def pdf_load_pattern(doc, obj):
38525: (4)             r""" Class-aware wrapper for `::pdf_load_pattern()`."""
38526: (4)             return _mupdf.pdf_load_pattern(doc, obj)
38527: (0)         def pdf_load_raw_stream(ref):
38528: (4)             r""" Class-aware wrapper for `::pdf_load_raw_stream()`."""
38529: (4)             return _mupdf.pdf_load_raw_stream(ref)
38530: (0)         def pdf_load_raw_stream_number(doc, num):
38531: (4)             r""" Class-aware wrapper for `::pdf_load_raw_stream_number()`."""
38532: (4)             return _mupdf.pdf_load_raw_stream_number(doc, num)
38533: (0)         def pdf_load_shading(doc, obj):
38534: (4)             r""" Class-aware wrapper for `::pdf_load_shading()`."""
38535: (4)             return _mupdf.pdf_load_shading(doc, obj)
38536: (0)         def pdf_load_stream(ref):
38537: (4)             r""" Class-aware wrapper for `::pdf_load_stream()`."""
38538: (4)             return _mupdf.pdf_load_stream(ref)
38539: (0)         def pdf_load_stream_number(doc, num):
38540: (4)             r""" Class-aware wrapper for `::pdf_load_stream_number()`."""
38541: (4)             return _mupdf.pdf_load_stream_number(doc, num)
38542: (0)         def pdf_load_stream_or_string_as_utf8(src):
38543: (4)             r""" Class-aware wrapper for `::pdf_load_stream_or_string_as_utf8()`."""
38544: (4)             return _mupdf.pdf_load_stream_or_string_as_utf8(src)
38545: (0)         def pdf_load_system_cmap(name):
38546: (4)             r""" Class-aware wrapper for `::pdf_load_system_cmap()`."""
38547: (4)             return _mupdf.pdf_load_system_cmap(name)
38548: (0)         def pdf_load_to_unicode(doc, font, strings, collection, cmapstm):
38549: (4)             r"""
38550: (4)             Class-aware wrapper for `::pdf_load_to_unicode()`.
38551: (4)             This function has out-params. Python/C# wrappers look like:
38552: (8)                 `pdf_load_to_unicode(::pdf_document *doc, ::pdf_font_desc *font, char
*collection, ::pdf_obj *cmapstm)` => const char *strings
38553: (4)             """
38554: (4)             return _mupdf.pdf_load_to_unicode(doc, font, strings, collection, cmapstm)
38555: (0)         def pdf_load_type3_glyphs(doc, fontdesc):
38556: (4)             r""" Class-aware wrapper for `::pdf_load_type3_glyphs()`."""
38557: (4)             return _mupdf.pdf_load_type3_glyphs(doc, fontdesc)
38558: (0)         def pdf_load_unencrypted_object(doc, num):
38559: (4)             r""" Class-aware wrapper for `::pdf_load_unencrypted_object()`."""
38560: (4)             return _mupdf.pdf_load_unencrypted_object(doc, num)
38561: (0)         def pdf_lookup_cmap(cmap, cpt):
38562: (4)             r""" Class-aware wrapper for `::pdf_lookup_cmap()`."""
38563: (4)             return _mupdf.pdf_lookup_cmap(cmap, cpt)
38564: (0)         def pdf_lookup_cmap_full(cmap, cpt, out):
38565: (4)             r"""
```

```
38566: (4)              Class-aware wrapper for `::pdf_lookup_cmap_full()`.
38567: (4)              This function has out-params. Python/C# wrappers look like:
38568: (8)                  `pdf_lookup_cmap_full(::pdf_cmap *cmap, unsigned int cpt)` => `(int,
int out)`
38569: (4)              """
38570: (4)              return _mupdf.pdf_lookup_cmap_full(cmap, cpt, out)
38571: (0)          def pdf_lookup_dest(doc, needle):
38572: (4)              r""" Class-aware wrapper for `::pdf_lookup_dest()`."""
38573: (4)              return _mupdf.pdf_lookup_dest(doc, needle)
38574: (0)          def pdf_lookup_field(form, name):
38575: (4)              r""" Class-aware wrapper for `::pdf_lookup_field()`."""
38576: (4)              return _mupdf.pdf_lookup_field(form, name)
38577: (0)          def pdf_lookup_metadata(doc, key, ptr, size):
38578: (4)              r""" Class-aware wrapper for `::pdf_lookup_metadata()`."""
38579: (4)              return _mupdf.pdf_lookup_metadata(doc, key, ptr, size)
38580: (0)          def pdf_lookup_metadata2(doc, key):
38581: (4)              r"""
38582: (4)              Class-aware wrapper for `::pdf_lookup_metadata2()`.
38583: (4)              C++ alternative to `pdf_lookup_metadata()` that returns a `std::string`
38584: (4)              or calls `fz_throw()` if not found.
38585: (4)              """
38586: (4)              return _mupdf.pdf_lookup_metadata2(doc, key)
38587: (0)          def pdf_lookup_name(doc, which, needle):
38588: (4)              r""" Class-aware wrapper for `::pdf_lookup_name()`."""
38589: (4)              return _mupdf.pdf_lookup_name(doc, which, needle)
38590: (0)          def pdf_lookup_number(root, needle):
38591: (4)              r""" Class-aware wrapper for `::pdf_lookup_number()`."""
38592: (4)              return _mupdf.pdf_lookup_number(root, needle)
38593: (0)          def pdf_lookup_page_loc(doc, needle, parentp, indexp):
38594: (4)              r"""
38595: (4)              Class-aware wrapper for `::pdf_lookup_page_loc()`.
38596: (4)              This function has out-params. Python/C# wrappers look like:
38597: (8)                  `pdf_lookup_page_loc(::pdf_document *doc, int needle, ::pdf_obj
**parentp)` => `(pdf_obj *, int indexp)`
38598: (4)              """
38599: (4)              return _mupdf.pdf_lookup_page_loc(doc, needle, parentp, indexp)
38600: (0)          def pdf_lookup_page_number(doc, pageobj):
38601: (4)              r""" Class-aware wrapper for `::pdf_lookup_page_number()`."""
38602: (4)              return _mupdf.pdf_lookup_page_number(doc, pageobj)
38603: (0)          def pdf_lookup_page_obj(doc, needle):
38604: (4)              r""" Class-aware wrapper for `::pdf_lookup_page_obj()`."""
38605: (4)              return _mupdf.pdf_lookup_page_obj(doc, needle)
38606: (0)          def pdf_lookup_substitute_font(mono, serif, bold, italic, len):
38607: (4)              r"""
38608: (4)              Class-aware wrapper for `::pdf_lookup_substitute_font()`.
38609: (4)              This function has out-params. Python/C# wrappers look like:
38610: (8)                  `pdf_lookup_substitute_font(int mono, int serif, int bold, int
italic)` => `(const unsigned char *, int len)`
38611: (4)              """
38612: (4)              return _mupdf.pdf_lookup_substitute_font(mono, serif, bold, italic, len)
38613: (0)          def pdf_map_one_to_many(cmap, one, many, len):
38614: (4)              r"""
38615: (4)              Class-aware wrapper for `::pdf_map_one_to_many()`.
38616: (4)              This function has out-params. Python/C# wrappers look like:
38617: (8)                  `pdf_map_one_to_many(::pdf_cmap *cmap, unsigned int one, size_t len)`
=> int many
38618: (4)              """
38619: (4)              return _mupdf.pdf_map_one_to_many(cmap, one, many, len)
38620: (0)          def pdf_map_range_to_range(cmap, srclo, srchi, dstlo):
38621: (4)              r""" Class-aware wrapper for `::pdf_map_range_to_range()`."""
38622: (4)              return _mupdf.pdf_map_range_to_range(cmap, srclo, srchi, dstlo)
38623: (0)          def pdf_mark_bits_reset(marks):
38624: (4)              r""" Class-aware wrapper for `::pdf_mark_bits_reset()`."""
38625: (4)              return _mupdf.pdf_mark_bits_reset(marks)
38626: (0)          def pdf_mark_bits_set(marks, obj):
38627: (4)              r""" Class-aware wrapper for `::pdf_mark_bits_set()`."""
38628: (4)              return _mupdf.pdf_mark_bits_set(marks, obj)
38629: (0)          def pdf_mark_list_check(list, obj):
38630: (4)              r""" Class-aware wrapper for `::pdf_mark_list_check()`."""
```

```
38631: (4)             return _mupdf.pdf_mark_list_check(list, obj)
38632: (0)         def pdf_mark_list_free(list):
38633: (4)             r""" Class-aware wrapper for `::pdf_mark_list_free()`."""
38634: (4)             return _mupdf.pdf_mark_list_free(list)
38635: (0)         def pdf_mark_list_init(list):
38636: (4)             r""" Class-aware wrapper for `::pdf_mark_list_init()`."""
38637: (4)             return _mupdf.pdf_mark_list_init(list)
38638: (0)         def pdf_mark_list_pop(list):
38639: (4)             r""" Class-aware wrapper for `::pdf_mark_list_pop()`."""
38640: (4)             return _mupdf.pdf_mark_list_pop(list)
38641: (0)         def pdf_mark_list_push(list, obj):
38642: (4)             r""" Class-aware wrapper for `::pdf_mark_list_push()`."""
38643: (4)             return _mupdf.pdf_mark_list_push(list, obj)
38644: (0)         def pdf_mark_obj(obj):
38645: (4)             r""" Class-aware wrapper for `::pdf_mark_obj()`."""
38646: (4)             return _mupdf.pdf_mark_obj(obj)
38647: (0)         def pdf_mark_xref(doc):
38648: (4)             r""" Class-aware wrapper for `::pdf_mark_xref()`."""
38649: (4)             return _mupdf.pdf_mark_xref(doc)
38650: (0)         def pdf_metadata(doc):
38651: (4)             r""" Class-aware wrapper for `::pdf_metadata()`."""
38652: (4)             return _mupdf.pdf_metadata(doc)
38653: (0)         def pdf_minimize_document(doc):
38654: (4)             r""" Class-aware wrapper for `::pdf_minimize_document()`."""
38655: (4)             return _mupdf.pdf_minimize_document(doc)
38656: (0)         def pdf_name_eq(a, b):
38657: (4)             r""" Class-aware wrapper for `::pdf_name_eq()`."""
38658: (4)             return _mupdf.pdf_name_eq(a, b)
38659: (0)         def pdf_name_from_intent(intent):
38660: (4)             r""" Class-aware wrapper for `::pdf_name_from_intent()`."""
38661: (4)             return _mupdf.pdf_name_from_intent(intent)
38662: (0)         def pdf_name_from_line_ending(end):
38663: (4)             r""" Class-aware wrapper for `::pdf_name_from_line_ending()`."""
38664: (4)             return _mupdf.pdf_name_from_line_ending(end)
38665: (0)         def pdf_needs_password(doc):
38666: (4)             r""" Class-aware wrapper for `::pdf_needs_password()`."""
38667: (4)             return _mupdf.pdf_needs_password(doc)
38668: (0)         def pdf_new_action_from_link(doc, uri):
38669: (4)             r""" Class-aware wrapper for `::pdf_new_action_from_link()`."""
38670: (4)             return _mupdf.pdf_new_action_from_link(doc, uri)
38671: (0)         def pdf_new_array(doc, initialcap):
38672: (4)             r""" Class-aware wrapper for `::pdf_new_array()`."""
38673: (4)             return _mupdf.pdf_new_array(doc, initialcap)
38674: (0)         def pdf_new_buffer_processor(buffer, ahxencode, newlines):
38675: (4)             r""" Class-aware wrapper for `::pdf_new_buffer_processor()`."""
38676: (4)             return _mupdf.pdf_new_buffer_processor(buffer, ahxencode, newlines)
38677: (0)         def pdf_new_cmap():
38678: (4)             r""" Class-aware wrapper for `::pdf_new_cmap()`."""
38679: (4)             return _mupdf.pdf_new_cmap()
38680: (0)         def pdf_new_color_filter(doc, chain, struct_parents, transform, options,
copts):
38681: (4)             r""" Class-aware wrapper for `::pdf_new_color_filter()`."""
38682: (4)             return _mupdf.pdf_new_color_filter(doc, chain, struct_parents, transform,
options, copts)
38683: (0)         def pdf_new_date(doc, time):
38684: (4)             r""" Class-aware wrapper for `::pdf_new_date()`."""
38685: (4)             return _mupdf.pdf_new_date(doc, time)
38686: (0)         def pdf_new_dest_from_link(doc, uri, is_remote):
38687: (4)             r""" Class-aware wrapper for `::pdf_new_dest_from_link()`."""
38688: (4)             return _mupdf.pdf_new_dest_from_link(doc, uri, is_remote)
38689: (0)         def pdf_new_dict(doc, initialcap):
38690: (4)             r""" Class-aware wrapper for `::pdf_new_dict()`."""
38691: (4)             return _mupdf.pdf_new_dict(doc, initialcap)
38692: (0)         def pdf_new_display_list_from_annot(annot):
38693: (4)             r""" Class-aware wrapper for `::pdf_new_display_list_from_annot()`."""
38694: (4)             return _mupdf.pdf_new_display_list_from_annot(annot)
38695: (0)         def pdf_new_graft_map(dst):
38696: (4)             r""" Class-aware wrapper for `::pdf_new_graft_map()`."""
38697: (4)             return _mupdf.pdf_new_graft_map(dst)
```

```
38698: (0)          def pdf_new_identity_cmap(wmode, bytes):
38699: (4)              r""" Class-aware wrapper for `::pdf_new_identity_cmap()`."""
38700: (4)              return _mupdf.pdf_new_identity_cmap(wmode, bytes)
38701: (0)          def pdf_new_indirect(doc, num, gen):
38702: (4)              r""" Class-aware wrapper for `::pdf_new_indirect()`."""
38703: (4)              return _mupdf.pdf_new_indirect(doc, num, gen)
38704: (0)          def pdf_new_int(i):
38705: (4)              r""" Class-aware wrapper for `::pdf_new_int()`."""
38706: (4)              return _mupdf.pdf_new_int(i)
38707: (0)          def pdf_new_link(page, rect, uri, obj):
38708: (4)              r""" Class-aware wrapper for `::pdf_new_link()`."""
38709: (4)              return _mupdf.pdf_new_link(page, rect, uri, obj)
38710: (0)          def pdf_new_matrix(doc, mtx):
38711: (4)              r""" Class-aware wrapper for `::pdf_new_matrix()`."""
38712: (4)              return _mupdf.pdf_new_matrix(doc, mtx)
38713: (0)          def pdf_new_name(str):
38714: (4)              r""" Class-aware wrapper for `::pdf_new_name()`."""
38715: (4)              return _mupdf.pdf_new_name(str)
38716: (0)          def pdf_new_output_processor(out, ahxencode, newlines):
38717: (4)              r""" Class-aware wrapper for `::pdf_new_output_processor()`."""
38718: (4)              return _mupdf.pdf_new_output_processor(out, ahxencode, newlines)
38719: (0)          def pdf_new_pdf_device(doc, topctm, resources, contents):
38720: (4)              r""" Class-aware wrapper for `::pdf_new_pdf_device()`."""
38721: (4)              return _mupdf.pdf_new_pdf_device(doc, topctm, resources, contents)
38722: (0)          def pdf_new_pixmap_from_annot(annot, ctm, cs, seps, alpha):
38723: (4)              r""" Class-aware wrapper for `::pdf_new_pixmap_from_annot()`."""
38724: (4)              return _mupdf.pdf_new_pixmap_from_annot(annot, ctm, cs, seps, alpha)
38725: (0)          def pdf_new_pixmap_from_page_contents_with_separations_and_usage(page, ctm,
cs, seps, alpha, usage, box):
38726: (4)              r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_contents_with_separations_and_usage()`."""
38727: (4)              return
_mupdf.pdf_new_pixmap_from_page_contents_with_separations_and_usage(page, ctm, cs, seps, alpha,
usage, box)
38728: (0)          def pdf_new_pixmap_from_page_contents_with_usage(page, ctm, cs, alpha, usage,
box):
38729: (4)              r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_contents_with_usage()`."""
38730: (4)              return _mupdf.pdf_new_pixmap_from_page_contents_with_usage(page, ctm, cs,
alpha, usage, box)
38731: (0)          def pdf_new_pixmap_from_page_with_separations_and_usage(page, ctm, cs, seps,
alpha, usage, box):
38732: (4)              r""" Class-aware wrapper for
`::pdf_new_pixmap_from_page_with_separations_and_usage()`."""
38733: (4)              return _mupdf.pdf_new_pixmap_from_page_with_separations_and_usage(page,
ctm, cs, seps, alpha, usage, box)
38734: (0)          def pdf_new_pixmap_from_page_with_usage(page, ctm, cs, alpha, usage, box):
38735: (4)              r""" Class-aware wrapper for `::pdf_new_pixmap_from_page_with_usage()`."""
38736: (4)              return _mupdf.pdf_new_pixmap_from_page_with_usage(page, ctm, cs, alpha,
usage, box)
38737: (0)          def pdf_new_processor(size):
38738: (4)              r""" Class-aware wrapper for `::pdf_new_processor()`."""
38739: (4)              return _mupdf.pdf_new_processor(size)
38740: (0)          def pdf_new_real(f):
38741: (4)              r""" Class-aware wrapper for `::pdf_new_real()`."""
38742: (4)              return _mupdf.pdf_new_real(f)
38743: (0)          def pdf_new_rect(doc, rect):
38744: (4)              r""" Class-aware wrapper for `::pdf_new_rect()`."""
38745: (4)              return _mupdf.pdf_new_rect(doc, rect)
38746: (0)          def pdf_new_run_processor(doc, dev, ctm, struct_parent, usage, gstate,
default_cs, cookie):
38747: (4)              r""" Class-aware wrapper for `::pdf_new_run_processor()`."""
38748: (4)              return _mupdf.pdf_new_run_processor(doc, dev, ctm, struct_parent, usage,
gstate, default_cs, cookie)
38749: (0)          def pdf_new_sanitize_filter(doc, chain, struct_parents, transform, options,
sopts):
38750: (4)              r""" Class-aware wrapper for `::pdf_new_sanitize_filter()`."""
38751: (4)              return _mupdf.pdf_new_sanitize_filter(doc, chain, struct_parents,
transform, options, sopts)
```

```
38752: (0)          def pdf_new_string(str, len):
38753: (4)              r""" Class-aware wrapper for `::pdf_new_string()`."""
38754: (4)              return _mupdf.pdf_new_string(str, len)
38755: (0)          def pdf_new_text_string(s):
38756: (4)              r""" Class-aware wrapper for `::pdf_new_text_string()`."""
38757: (4)              return _mupdf.pdf_new_text_string(s)
38758: (0)          def pdf_new_uri_from_explicit_dest(dest):
38759: (4)              r""" Class-aware wrapper for `::pdf_new_uri_from_explicit_dest()`."""
38760: (4)              return _mupdf.pdf_new_uri_from_explicit_dest(dest)
38761: (0)          def pdf_new_uri_from_path_and_explicit_dest(path, dest):
38762: (4)              r""" Class-aware wrapper for
`::pdf_new_uri_from_path_and_explicit_dest()`."""
38763: (4)              return _mupdf.pdf_new_uri_from_path_and_explicit_dest(path, dest)
38764: (0)          def pdf_new_uri_from_path_and_named_dest(path, name):
38765: (4)              r""" Class-aware wrapper for
`::pdf_new_uri_from_path_and_named_dest()`."""
38766: (4)              return _mupdf.pdf_new_uri_from_path_and_named_dest(path, name)
38767: (0)          def pdf_new_utf8_from_pdf_stream_obj(src):
38768: (4)              r""" Class-aware wrapper for `::pdf_new_utf8_from_pdf_stream_obj()`."""
38769: (4)              return _mupdf.pdf_new_utf8_from_pdf_stream_obj(src)
38770: (0)          def pdf_new_utf8_from_pdf_string(srcptr, srclen):
38771: (4)              r""" Class-aware wrapper for `::pdf_new_utf8_from_pdf_string()`."""
38772: (4)              return _mupdf.pdf_new_utf8_from_pdf_string(srcptr, srclen)
38773: (0)          def pdf_new_utf8_from_pdf_string_obj(src):
38774: (4)              r""" Class-aware wrapper for `::pdf_new_utf8_from_pdf_string_obj()`."""
38775: (4)              return _mupdf.pdf_new_utf8_from_pdf_string_obj(src)
38776: (0)          def pdf_new_xobject(doc, bbox, matrix, res, buffer):
38777: (4)              r""" Class-aware wrapper for `::pdf_new_xobject()`."""
38778: (4)              return _mupdf.pdf_new_xobject(doc, bbox, matrix, res, buffer)
38779: (0)          def pdf_next_annot(annot):
38780: (4)              r""" Class-aware wrapper for `::pdf_next_annot()`."""
38781: (4)              return _mupdf.pdf_next_annot(annot)
38782: (0)          def pdf_next_widget(previous):
38783: (4)              r""" Class-aware wrapper for `::pdf_next_widget()`."""
38784: (4)              return _mupdf.pdf_next_widget(previous)
38785: (0)          def pdf_obj_is_dirty(obj):
38786: (4)              r""" Class-aware wrapper for `::pdf_obj_is_dirty()`."""
38787: (4)              return _mupdf.pdf_obj_is_dirty(obj)
38788: (0)          def pdf_obj_is_incremental(obj):
38789: (4)              r""" Class-aware wrapper for `::pdf_obj_is_incremental()`."""
38790: (4)              return _mupdf.pdf_obj_is_incremental(obj)
38791: (0)          def pdf_obj_marked(obj):
38792: (4)              r""" Class-aware wrapper for `::pdf_obj_marked()`."""
38793: (4)              return _mupdf.pdf_obj_marked(obj)
38794: (0)          def pdf_obj_memo(obj, bit, memo):
38795: (4)              r"""
38796: (4)              Class-aware wrapper for `::pdf_obj_memo()`.
38797: (4)              This function has out-params. Python/C# wrappers look like:
38798: (8)                  `pdf_obj_memo(::pdf_obj *obj, int bit)` => `(int, int memo)`
38799: (4)              """
38800: (4)              return _mupdf.pdf_obj_memo(obj, bit, memo)
38801: (0)          def pdf_obj_num_is_stream(doc, num):
38802: (4)              r""" Class-aware wrapper for `::pdf_obj_num_is_stream()`."""
38803: (4)              return _mupdf.pdf_obj_num_is_stream(doc, num)
38804: (0)          def pdf_obj_parent_num(obj):
38805: (4)              r""" Class-aware wrapper for `::pdf_obj_parent_num()`."""
38806: (4)              return _mupdf.pdf_obj_parent_num(obj)
38807: (0)          def pdf_obj_refs(ref):
38808: (4)              r""" Class-aware wrapper for `::pdf_obj_refs()`."""
38809: (4)              return _mupdf.pdf_obj_refs(ref)
38810: (0)          def pdf_objcmp(a, b):
38811: (4)              r""" Class-aware wrapper for `::pdf_objcmp()`."""
38812: (4)              return _mupdf.pdf_objcmp(a, b)
38813: (0)          def pdf_objcmp_deep(a, b):
38814: (4)              r""" Class-aware wrapper for `::pdf_objcmp_deep()`."""
38815: (4)              return _mupdf.pdf_objcmp_deep(a, b)
38816: (0)          def pdf_objcmp_resolve(a, b):
38817: (4)              r""" Class-aware wrapper for `::pdf_objcmp_resolve()`."""
38818: (4)              return _mupdf.pdf_objcmp_resolve(a, b)
```

```
38819: (0)          def pdf_open_contents_stream(doc, obj):
38820: (4)              r""" Class-aware wrapper for `::pdf_open_contents_stream()`."""
38821: (4)              return _mupdf.pdf_open_contents_stream(doc, obj)
38822: (0)          def pdf_open_crypt(chain, crypt, num, gen):
38823: (4)              r""" Class-aware wrapper for `::pdf_open_crypt()`."""
38824: (4)              return _mupdf.pdf_open_crypt(chain, crypt, num, gen)
38825: (0)          def pdf_open_crypt_with_filter(chain, crypt, name, num, gen):
38826: (4)              r""" Class-aware wrapper for `::pdf_open_crypt_with_filter()`."""
38827: (4)              return _mupdf.pdf_open_crypt_with_filter(chain, crypt, name, num, gen)
38828: (0)          def pdf_open_document(filename):
38829: (4)              r""" Class-aware wrapper for `::pdf_open_document()`."""
38830: (4)              return _mupdf.pdf_open_document(filename)
38831: (0)          def pdf_open_document_with_stream(file):
38832: (4)              r""" Class-aware wrapper for `::pdf_open_document_with_stream()`."""
38833: (4)              return _mupdf.pdf_open_document_with_stream(file)
38834: (0)          def pdf_open_inline_stream(doc, stmobj, length, chain, params):
38835: (4)              r""" Class-aware wrapper for `::pdf_open_inline_stream()`."""
38836: (4)              return _mupdf.pdf_open_inline_stream(doc, stmobj, length, chain, params)
38837: (0)          def pdf_open_raw_stream(ref):
38838: (4)              r""" Class-aware wrapper for `::pdf_open_raw_stream()`."""
38839: (4)              return _mupdf.pdf_open_raw_stream(ref)
38840: (0)          def pdf_open_raw_stream_number(doc, num):
38841: (4)              r""" Class-aware wrapper for `::pdf_open_raw_stream_number()`."""
38842: (4)              return _mupdf.pdf_open_raw_stream_number(doc, num)
38843: (0)          def pdf_open_stream(ref):
38844: (4)              r""" Class-aware wrapper for `::pdf_open_stream()`."""
38845: (4)              return _mupdf.pdf_open_stream(ref)
38846: (0)          def pdf_open_stream_number(doc, num):
38847: (4)              r""" Class-aware wrapper for `::pdf_open_stream_number()`."""
38848: (4)              return _mupdf.pdf_open_stream_number(doc, num)
38849: (0)          def pdf_open_stream_with_offset(doc, num, dict, stm_ofs):
38850: (4)              r""" Class-aware wrapper for `::pdf_open_stream_with_offset()`."""
38851: (4)              return _mupdf.pdf_open_stream_with_offset(doc, num, dict, stm_ofs)
38852: (0)          def pdf_page_contents(page):
38853: (4)              r""" Class-aware wrapper for `::pdf_page_contents()`."""
38854: (4)              return _mupdf.pdf_page_contents(page)
38855: (0)          def pdf_page_event_close(page):
38856: (4)              r""" Class-aware wrapper for `::pdf_page_event_close()`."""
38857: (4)              return _mupdf.pdf_page_event_close(page)
38858: (0)          def pdf_page_event_open(page):
38859: (4)              r""" Class-aware wrapper for `::pdf_page_event_open()`."""
38860: (4)              return _mupdf.pdf_page_event_open(page)
38861: (0)          def pdf_page_from_fz_page(ptr):
38862: (4)              r""" Class-aware wrapper for `::pdf_page_from_fz_page()`."""
38863: (4)              return _mupdf.pdf_page_from_fz_page(ptr)
38864: (0)          def pdf_page_group(page):
38865: (4)              r""" Class-aware wrapper for `::pdf_page_group()`."""
38866: (4)              return _mupdf.pdf_page_group(page)
38867: (0)          def pdf_page_has_transparency(page):
38868: (4)              r""" Class-aware wrapper for `::pdf_page_has_transparency()`."""
38869: (4)              return _mupdf.pdf_page_has_transparency(page)
38870: (0)          def pdf_page_label(doc, page, buf, size):
38871: (4)              r""" Class-aware wrapper for `::pdf_page_label()`."""
38872: (4)              return _mupdf.pdf_page_label(doc, page, buf, size)
38873: (0)          def pdf_page_label_imp(doc, chapter, page, buf, size):
38874: (4)              r""" Class-aware wrapper for `::pdf_page_label_imp()`."""
38875: (4)              return _mupdf.pdf_page_label_imp(doc, chapter, page, buf, size)
38876: (0)          def pdf_page_obj_transform(pageobj, outbox, outctm):
38877: (4)              r""" Class-aware wrapper for `::pdf_page_obj_transform()`."""
38878: (4)              return _mupdf.pdf_page_obj_transform(pageobj, outbox, outctm)
38879: (0)          def pdf_page_obj_transform_box(pageobj, outbox, out, box):
38880: (4)              r""" Class-aware wrapper for `::pdf_page_obj_transform_box()`."""
38881: (4)              return _mupdf.pdf_page_obj_transform_box(pageobj, outbox, out, box)
38882: (0)          def pdf_page_presentation(page, transition, duration):
38883: (4)              r"""
38884: (4)              Class-aware wrapper for `::pdf_page_presentation()`.
38885: (4)              This function has out-params. Python/C# wrappers look like:
38886: (8)                  `pdf_page_presentation(::pdf_page *page, ::fz_transition *transition)`
=> `(fz_transition *, float duration)`
```

```
38887: (4)              """
38888: (4)              return _mupdf.pdf_page_presentation(page, transition, duration)
38889: (0)          def pdf_page_resources(page):
38890: (4)              r""" Class-aware wrapper for `::pdf_page_resources()`."""
38891: (4)              return _mupdf.pdf_page_resources(page)
38892: (0)          def pdf_page_separations(page):
38893: (4)              r""" Class-aware wrapper for `::pdf_page_separations()`."""
38894: (4)              return _mupdf.pdf_page_separations(page)
38895: (0)          def pdf_page_transform(page, mediabox, ctm):
38896: (4)              r""" Class-aware wrapper for `::pdf_page_transform()`."""
38897: (4)              return _mupdf.pdf_page_transform(page, mediabox, ctm)
38898: (0)          def pdf_page_transform_box(page, mediabox, ctm, box):
38899: (4)              r""" Class-aware wrapper for `::pdf_page_transform_box()`."""
38900: (4)              return _mupdf.pdf_page_transform_box(page, mediabox, ctm, box)
38901: (0)          def pdf_page_write(doc, mediabox, presources, pcontents):
38902: (4)              r"""
38903: (4)              Class-aware wrapper for `::pdf_page_write()`.
38904: (4)              This function has out-params. Python/C# wrappers look like:
38905: (8)                  `pdf_page_write(::pdf_document *doc, ::fz_rect mediabox, ::pdf_obj
**presources, ::fz_buffer **pcontents)` => `(fz_device *)`
38906: (4)              """
38907: (4)              return _mupdf.pdf_page_write(doc, mediabox, presources, pcontents)
38908: (0)          def pdf_parse_array(doc, f, buf):
38909: (4)              r""" Class-aware wrapper for `::pdf_parse_array()`."""
38910: (4)              return _mupdf.pdf_parse_array(doc, f, buf)
38911: (0)          def pdf_parse_date(s):
38912: (4)              r""" Class-aware wrapper for `::pdf_parse_date()`."""
38913: (4)              return _mupdf.pdf_parse_date(s)
38914: (0)          def pdf_parse_default_appearance(da, font, size, n, color):
38915: (4)              r"""
38916: (4)              Class-aware wrapper for `::pdf_parse_default_appearance()`.
38917: (4)              This function has out-params. Python/C# wrappers look like:
38918: (8)                  `pdf_parse_default_appearance(const char *da, float color[4])` =>
`(const char *font, float size, int n)`
38919: (4)              """
38920: (4)              return _mupdf.pdf_parse_default_appearance(da, font, size, n, color)
38921: (0)          def pdf_parse_dict(doc, f, buf):
38922: (4)              r""" Class-aware wrapper for `::pdf_parse_dict()`."""
38923: (4)              return _mupdf.pdf_parse_dict(doc, f, buf)
38924: (0)          def pdf_parse_ind_obj(doc, f, num, gen, stm_ofs, try_repair):
38925: (4)              r"""
38926: (4)              Class-aware wrapper for `::pdf_parse_ind_obj()`.
38927: (4)              This function has out-params. Python/C# wrappers look like:
38928: (8)                  `pdf_parse_ind_obj(::pdf_document *doc, ::fz_stream *f)` => `(pdf_obj
*, int num, int gen, int64_t stm_ofs, int try_repair)`
38929: (4)              """
38930: (4)              return _mupdf.pdf_parse_ind_obj(doc, f, num, gen, stm_ofs, try_repair)
38931: (0)          def pdf_parse_journal_obj(doc, stm, onum, ostm, newobj):
38932: (4)              r"""
38933: (4)              Class-aware wrapper for `::pdf_parse_journal_obj()`.
38934: (4)              This function has out-params. Python/C# wrappers look like:
38935: (8)                  `pdf_parse_journal_obj(::pdf_document *doc, ::fz_stream *stm,
::fz_buffer **ostm)` => `(pdf_obj *, int onum, int newobj)`
38936: (4)              """
38937: (4)              return _mupdf.pdf_parse_journal_obj(doc, stm, onum, ostm, newobj)
38938: (0)          def pdf_parse_stm_obj(doc, f, buf):
38939: (4)              r""" Class-aware wrapper for `::pdf_parse_stm_obj()`."""
38940: (4)              return _mupdf.pdf_parse_stm_obj(doc, f, buf)
38941: (0)          def pdf_parse_write_options(opts, args):
38942: (4)              r""" Class-aware wrapper for `::pdf_parse_write_options()`."""
38943: (4)              return _mupdf.pdf_parse_write_options(opts, args)
38944: (0)          def pdf_pin_document(obj):
38945: (4)              r""" Class-aware wrapper for `::pdf_pin_document()`."""
38946: (4)              return _mupdf.pdf_pin_document(obj)
38947: (0)          def pdf_preview_signature_as_display_list(w, h, lang, signer,
appearance_flags, graphic, reason, location):
38948: (4)              r""" Class-aware wrapper for
`::pdf_preview_signature_as_display_list()`."""
38949: (4)              return _mupdf.pdf_preview_signature_as_display_list(w, h, lang, signer,
```

```
appearance_flags, graphic, reason, location)
38950: (0)          def pdf_preview_signature_as_pixmap(w, h, lang, signer, appearance_flags,
graphic, reason, location):
38951: (4)              r""" Class-aware wrapper for `::pdf_preview_signature_as_pixmap()`."""
38952: (4)              return _mupdf.pdf_preview_signature_as_pixmap(w, h, lang, signer,
appearance_flags, graphic, reason, location)
38953: (0)          def pdf_print_crypt(out, crypt):
38954: (4)              r""" Class-aware wrapper for `::pdf_print_crypt()`."""
38955: (4)              return _mupdf.pdf_print_crypt(out, crypt)
38956: (0)          def pdf_print_default_appearance(buf, nbuf, font, size, n, color):
38957: (4)              r""" Class-aware wrapper for `::pdf_print_default_appearance()`."""
38958: (4)              return _mupdf.pdf_print_default_appearance(buf, nbuf, font, size, n,
color)
38959: (0)          def pdf_print_encrypted_obj(out, obj, tight, ascii, crypt, num, gen, sep):
38960: (4)              r"""
38961: (4)              Class-aware wrapper for `::pdf_print_encrypted_obj()`.
38962: (4)              This function has out-params. Python/C# wrappers look like:
38963: (8)                  `pdf_print_encrypted_obj(::fz_output *out, ::pdf_obj *obj, int tight,
int ascii, ::pdf_crypt *crypt, int num, int gen)` => int sep
38964: (4)              """
38965: (4)              return _mupdf.pdf_print_encrypted_obj(out, obj, tight, ascii, crypt, num,
gen, sep)
38966: (0)          def pdf_print_font(out, fontdesc):
38967: (4)              r""" Class-aware wrapper for `::pdf_print_font()`."""
38968: (4)              return _mupdf.pdf_print_font(out, fontdesc)
38969: (0)          def pdf_print_obj(out, obj, tight, ascii):
38970: (4)              r""" Class-aware wrapper for `::pdf_print_obj()`."""
38971: (4)              return _mupdf.pdf_print_obj(out, obj, tight, ascii)
38972: (0)          def pdf_process_annot(proc, annot, cookie):
38973: (4)              r""" Class-aware wrapper for `::pdf_process_annot()`."""
38974: (4)              return _mupdf.pdf_process_annot(proc, annot, cookie)
38975: (0)          def pdf_process_contents(proc, doc, res, stm, cookie, out_res):
38976: (4)              r"""
38977: (4)              Class-aware wrapper for `::pdf_process_contents()`.
38978: (4)              This function has out-params. Python/C# wrappers look like:
38979: (8)                  `pdf_process_contents(::pdf_processor *proc, ::pdf_document *doc,
::pdf_obj *res, ::pdf_obj *stm, ::fz_cookie *cookie, ::pdf_obj **out_res)` =>
38980: (4)              """
38981: (4)              return _mupdf.pdf_process_contents(proc, doc, res, stm, cookie, out_res)
38982: (0)          def pdf_process_glyph(proc, doc, resources, contents):
38983: (4)              r""" Class-aware wrapper for `::pdf_process_glyph()`."""
38984: (4)              return _mupdf.pdf_process_glyph(proc, doc, resources, contents)
38985: (0)          def pdf_process_raw_contents(proc, doc, rdb, stmobj, cookie):
38986: (4)              r""" Class-aware wrapper for `::pdf_process_raw_contents()`."""
38987: (4)              return _mupdf.pdf_process_raw_contents(proc, doc, rdb, stmobj, cookie)
38988: (0)          def pdf_processor_pop_resources(proc):
38989: (4)              r""" Class-aware wrapper for `::pdf_processor_pop_resources()`."""
38990: (4)              return _mupdf.pdf_processor_pop_resources(proc)
38991: (0)          def pdf_processor_push_resources(proc, res):
38992: (4)              r""" Class-aware wrapper for `::pdf_processor_push_resources()`."""
38993: (4)              return _mupdf.pdf_processor_push_resources(proc, res)
38994: (0)          def pdf_progressive_advance(doc, pagenum):
38995: (4)              r""" Class-aware wrapper for `::pdf_progressive_advance()`."""
38996: (4)              return _mupdf.pdf_progressive_advance(doc, pagenum)
38997: (0)          def pdf_purge_local_font_resources(doc):
38998: (4)              r""" Class-aware wrapper for `::pdf_purge_local_font_resources()`."""
38999: (4)              return _mupdf.pdf_purge_local_font_resources(doc)
39000: (0)          def pdf_purge_locals_from_store(doc):
39001: (4)              r""" Class-aware wrapper for `::pdf_purge_locals_from_store()`."""
39002: (4)              return _mupdf.pdf_purge_locals_from_store(doc)
39003: (0)          def pdf_read_journal(doc, stm):
39004: (4)              r""" Class-aware wrapper for `::pdf_read_journal()`."""
39005: (4)              return _mupdf.pdf_read_journal(doc, stm)
39006: (0)          def pdf_rearrange_pages(doc, count, pages):
39007: (4)              r""" Class-aware wrapper for `::pdf_rearrange_pages()`."""
39008: (4)              return _mupdf.pdf_rearrange_pages(doc, count, pages)
39009: (0)          def pdf_rearrange_pages2(doc, pages):
39010: (4)              r""" Class-aware wrapper for `::pdf_rearrange_pages2()`.   Swig-friendly
wrapper for pdf_rearrange_pages()."""
```

```
39011: (4)             return _mupdf.pdf_rearrange_pages2(doc, pages)
39012: (0)         def pdf_recolor_shade(shade, reshade, opaque):
39013: (4)             r"""
39014: (4)             Class-aware wrapper for `::pdf_recolor_shade()`.
39015: (8)                 Recolor a shade.
39016: (4)             """
39017: (4)             return _mupdf.pdf_recolor_shade(shade, reshade, opaque)
39018: (0)         def pdf_redact_page(doc, page, opts):
39019: (4)             r""" Class-aware wrapper for `::pdf_redact_page()`."""
39020: (4)             return _mupdf.pdf_redact_page(doc, page, opts)
39021: (0)         def pdf_redo(doc):
39022: (4)             r""" Class-aware wrapper for `::pdf_redo()`."""
39023: (4)             return _mupdf.pdf_redo(doc)
39024: (0)         def pdf_remove_item(drop, key):
39025: (4)             r""" Class-aware wrapper for `::pdf_remove_item()`."""
39026: (4)             return _mupdf.pdf_remove_item(drop, key)
39027: (0)         def pdf_repair_obj(doc, buf, stmofsp, stmlenp, encrypt, id, page, tmpofs,
root):
39028: (4)             r"""
39029: (4)             Class-aware wrapper for `::pdf_repair_obj()`.
39030: (4)             This function has out-params. Python/C# wrappers look like:
39031: (8)                 `pdf_repair_obj(::pdf_document *doc, ::pdf_lexbuf *buf, ::pdf_obj
**encrypt, ::pdf_obj **id, ::pdf_obj **page, ::pdf_obj **root)` => `(int, int64_t stmofsp, int64_t
stmlenp, int64_t tmpofs)`
39032: (4)             """
39033: (4)             return _mupdf.pdf_repair_obj(doc, buf, stmofsp, stmlenp, encrypt, id,
page, tmpofs, root)
39034: (0)         def pdf_repair_obj_stms(doc):
39035: (4)             r""" Class-aware wrapper for `::pdf_repair_obj_stms()`."""
39036: (4)             return _mupdf.pdf_repair_obj_stms(doc)
39037: (0)         def pdf_repair_trailer(doc):
39038: (4)             r""" Class-aware wrapper for `::pdf_repair_trailer()`."""
39039: (4)             return _mupdf.pdf_repair_trailer(doc)
39040: (0)         def pdf_repair_xref(doc):
39041: (4)             r""" Class-aware wrapper for `::pdf_repair_xref()`."""
39042: (4)             return _mupdf.pdf_repair_xref(doc)
39043: (0)         def pdf_replace_xref(doc, entries, n):
39044: (4)             r""" Class-aware wrapper for `::pdf_replace_xref()`."""
39045: (4)             return _mupdf.pdf_replace_xref(doc, entries, n)
39046: (0)         def pdf_reset_form(doc, fields, exclude):
39047: (4)             r""" Class-aware wrapper for `::pdf_reset_form()`."""
39048: (4)             return _mupdf.pdf_reset_form(doc, fields, exclude)
39049: (0)         def pdf_reset_processor(proc):
39050: (4)             r""" Class-aware wrapper for `::pdf_reset_processor()`."""
39051: (4)             return _mupdf.pdf_reset_processor(proc)
39052: (0)         def pdf_resolve_indirect(ref):
39053: (4)             r"""
39054: (4)             Class-aware wrapper for `::pdf_resolve_indirect()`.
39055: (8)                 Resolve an indirect object (or chain of objects).
39056: (8)                 This can cause xref reorganisations (solidifications etc) due to
39057: (8)                 repairs, so all held pdf_xref_entries should be considered
39058: (8)                 invalid after this call (other than the returned one).
39059: (4)             """
39060: (4)             return _mupdf.pdf_resolve_indirect(ref)
39061: (0)         def pdf_resolve_indirect_chain(ref):
39062: (4)             r""" Class-aware wrapper for `::pdf_resolve_indirect_chain()`."""
39063: (4)             return _mupdf.pdf_resolve_indirect_chain(ref)
39064: (0)         def pdf_resolve_link(doc, uri, xp, yp):
39065: (4)             r"""
39066: (4)             Class-aware wrapper for `::pdf_resolve_link()`.
39067: (4)             This function has out-params. Python/C# wrappers look like:
39068: (8)                 `pdf_resolve_link(::pdf_document *doc, const char *uri)` => `(int,
float xp, float yp)`
39069: (4)             """
39070: (4)             return _mupdf.pdf_resolve_link(doc, uri, xp, yp)
39071: (0)         def pdf_rewrite_images(doc, opts):
39072: (4)             r""" Class-aware wrapper for `::pdf_rewrite_images()`."""
39073: (4)             return _mupdf.pdf_rewrite_images(doc, opts)
39074: (0)         def pdf_run_annot(annot, dev, ctm, cookie):
```

```
39075: (4)             r""" Class-aware wrapper for `::pdf_run_annot()`."""
39076: (4)             return _mupdf.pdf_run_annot(annot, dev, ctm, cookie)
39077: (0)         def pdf_run_document_structure(doc, dev, cookie):
39078: (4)             r""" Class-aware wrapper for `::pdf_run_document_structure()`."""
39079: (4)             return _mupdf.pdf_run_document_structure(doc, dev, cookie)
39080: (0)         def pdf_run_glyph(doc, resources, contents, dev, ctm, gstate, default_cs):
39081: (4)             r""" Class-aware wrapper for `::pdf_run_glyph()`."""
39082: (4)             return _mupdf.pdf_run_glyph(doc, resources, contents, dev, ctm, gstate,
default_cs)
39083: (0)         def pdf_run_page(page, dev, ctm, cookie):
39084: (4)             r""" Class-aware wrapper for `::pdf_run_page()`."""
39085: (4)             return _mupdf.pdf_run_page(page, dev, ctm, cookie)
39086: (0)         def pdf_run_page_annots(page, dev, ctm, cookie):
39087: (4)             r""" Class-aware wrapper for `::pdf_run_page_annots()`."""
39088: (4)             return _mupdf.pdf_run_page_annots(page, dev, ctm, cookie)
39089: (0)         def pdf_run_page_annots_with_usage(page, dev, ctm, usage, cookie):
39090: (4)             r""" Class-aware wrapper for `::pdf_run_page_annots_with_usage()`."""
39091: (4)             return _mupdf.pdf_run_page_annots_with_usage(page, dev, ctm, usage,
cookie)
39092: (0)         def pdf_run_page_contents(page, dev, ctm, cookie):
39093: (4)             r""" Class-aware wrapper for `::pdf_run_page_contents()`."""
39094: (4)             return _mupdf.pdf_run_page_contents(page, dev, ctm, cookie)
39095: (0)         def pdf_run_page_contents_with_usage(page, dev, ctm, usage, cookie):
39096: (4)             r""" Class-aware wrapper for `::pdf_run_page_contents_with_usage()`."""
39097: (4)             return _mupdf.pdf_run_page_contents_with_usage(page, dev, ctm, usage,
cookie)
39098: (0)         def pdf_run_page_widgets(page, dev, ctm, cookie):
39099: (4)             r""" Class-aware wrapper for `::pdf_run_page_widgets()`."""
39100: (4)             return _mupdf.pdf_run_page_widgets(page, dev, ctm, cookie)
39101: (0)         def pdf_run_page_widgets_with_usage(page, dev, ctm, usage, cookie):
39102: (4)             r""" Class-aware wrapper for `::pdf_run_page_widgets_with_usage()`."""
39103: (4)             return _mupdf.pdf_run_page_widgets_with_usage(page, dev, ctm, usage,
cookie)
39104: (0)         def pdf_run_page_with_usage(page, dev, ctm, usage, cookie):
39105: (4)             r""" Class-aware wrapper for `::pdf_run_page_with_usage()`."""
39106: (4)             return _mupdf.pdf_run_page_with_usage(page, dev, ctm, usage, cookie)
39107: (0)         def pdf_sample_shade_function(samples, n, funcs, func, t0, t1):
39108: (4)             r"""
39109: (4)             Class-aware wrapper for `::pdf_sample_shade_function()`.
39110: (4)             This function has out-params. Python/C# wrappers look like:
39111: (8)                 `pdf_sample_shade_function(int n, int funcs, ::pdf_function **func,
float t0, float t1)` => `(float samples)`
39112: (4)             """
39113: (4)             return _mupdf.pdf_sample_shade_function(samples, n, funcs, func, t0, t1)
39114: (0)         def pdf_save_document(doc, filename, opts):
39115: (4)             r""" Class-aware wrapper for `::pdf_save_document()`."""
39116: (4)             return _mupdf.pdf_save_document(doc, filename, opts)
39117: (0)         def pdf_save_journal(doc, filename):
39118: (4)             r""" Class-aware wrapper for `::pdf_save_journal()`."""
39119: (4)             return _mupdf.pdf_save_journal(doc, filename)
39120: (0)         def pdf_save_snapshot(doc, filename):
39121: (4)             r""" Class-aware wrapper for `::pdf_save_snapshot()`."""
39122: (4)             return _mupdf.pdf_save_snapshot(doc, filename)
39123: (0)         def pdf_select_layer_config(doc, config_num):
39124: (4)             r""" Class-aware wrapper for `::pdf_select_layer_config()`."""
39125: (4)             return _mupdf.pdf_select_layer_config(doc, config_num)
39126: (0)         def pdf_select_layer_config_ui(doc, ui):
39127: (4)             r""" Class-aware wrapper for `::pdf_select_layer_config_ui()`."""
39128: (4)             return _mupdf.pdf_select_layer_config_ui(doc, ui)
39129: (0)         def pdf_serialise_journal(doc, out):
39130: (4)             r""" Class-aware wrapper for `::pdf_serialise_journal()`."""
39131: (4)             return _mupdf.pdf_serialise_journal(doc, out)
39132: (0)         def pdf_set_annot_active(annot, active):
39133: (4)             r""" Class-aware wrapper for `::pdf_set_annot_active()`."""
39134: (4)             return _mupdf.pdf_set_annot_active(annot, active)
39135: (0)         def pdf_set_annot_appearance(annot, appearance, state, ctm, bbox, res,
contents):
39136: (4)             r""" Class-aware wrapper for `::pdf_set_annot_appearance()`."""
39137: (4)             return _mupdf.pdf_set_annot_appearance(annot, appearance, state, ctm,
```

```
bbox, res, contents)
39138: (0)          def pdf_set_annot_appearance_from_display_list(annot, appearance, state, ctm,
list):
39139: (4)              r""" Class-aware wrapper for
`::pdf_set_annot_appearance_from_display_list()`."""
39140: (4)              return _mupdf.pdf_set_annot_appearance_from_display_list(annot,
appearance, state, ctm, list)
39141: (0)          def pdf_set_annot_author(annot, author):
39142: (4)              r""" Class-aware wrapper for `::pdf_set_annot_author()`."""
39143: (4)              return _mupdf.pdf_set_annot_author(annot, author)
39144: (0)          def pdf_set_annot_border(annot, width):
39145: (4)              r""" Class-aware wrapper for `::pdf_set_annot_border()`."""
39146: (4)              return _mupdf.pdf_set_annot_border(annot, width)
39147: (0)          def pdf_set_annot_border_effect(annot, effect):
39148: (4)              r""" Class-aware wrapper for `::pdf_set_annot_border_effect()`."""
39149: (4)              return _mupdf.pdf_set_annot_border_effect(annot, effect)
39150: (0)          def pdf_set_annot_border_effect_intensity(annot, intensity):
39151: (4)              r""" Class-aware wrapper for
`::pdf_set_annot_border_effect_intensity()`."""
39152: (4)              return _mupdf.pdf_set_annot_border_effect_intensity(annot, intensity)
39153: (0)          def pdf_set_annot_border_style(annot, style):
39154: (4)              r""" Class-aware wrapper for `::pdf_set_annot_border_style()`."""
39155: (4)              return _mupdf.pdf_set_annot_border_style(annot, style)
39156: (0)          def pdf_set_annot_border_width(annot, width):
39157: (4)              r""" Class-aware wrapper for `::pdf_set_annot_border_width()`."""
39158: (4)              return _mupdf.pdf_set_annot_border_width(annot, width)
39159: (0)          def pdf_set_annot_color(annot, n, color):
39160: (4)              r""" Class-aware wrapper for `::pdf_set_annot_color()`."""
39161: (4)              return _mupdf.pdf_set_annot_color(annot, n, color)
39162: (0)          def pdf_set_annot_contents(annot, text):
39163: (4)              r""" Class-aware wrapper for `::pdf_set_annot_contents()`."""
39164: (4)              return _mupdf.pdf_set_annot_contents(annot, text)
39165: (0)          def pdf_set_annot_creation_date(annot, time):
39166: (4)              r""" Class-aware wrapper for `::pdf_set_annot_creation_date()`."""
39167: (4)              return _mupdf.pdf_set_annot_creation_date(annot, time)
39168: (0)          def pdf_set_annot_default_appearance(annot, font, size, n, color):
39169: (4)              r""" Class-aware wrapper for `::pdf_set_annot_default_appearance()`."""
39170: (4)              return _mupdf.pdf_set_annot_default_appearance(annot, font, size, n,
color)
39171: (0)          def pdf_set_annot_field_value(doc, widget, text, ignore_trigger_events):
39172: (4)              r""" Class-aware wrapper for `::pdf_set_annot_field_value()`."""
39173: (4)              return _mupdf.pdf_set_annot_field_value(doc, widget, text,
ignore_trigger_events)
39174: (0)          def pdf_set_annot_filespec(annot, obj):
39175: (4)              r""" Class-aware wrapper for `::pdf_set_annot_filespec()`."""
39176: (4)              return _mupdf.pdf_set_annot_filespec(annot, obj)
39177: (0)          def pdf_set_annot_flags(annot, flags):
39178: (4)              r""" Class-aware wrapper for `::pdf_set_annot_flags()`."""
39179: (4)              return _mupdf.pdf_set_annot_flags(annot, flags)
39180: (0)          def pdf_set_annot_hidden_for_editing(annot, hidden):
39181: (4)              r""" Class-aware wrapper for `::pdf_set_annot_hidden_for_editing()`."""
39182: (4)              return _mupdf.pdf_set_annot_hidden_for_editing(annot, hidden)
39183: (0)          def pdf_set_annot_hot(annot, hot):
39184: (4)              r""" Class-aware wrapper for `::pdf_set_annot_hot()`."""
39185: (4)              return _mupdf.pdf_set_annot_hot(annot, hot)
39186: (0)          def pdf_set_annot_icon_name(annot, name):
39187: (4)              r""" Class-aware wrapper for `::pdf_set_annot_icon_name()`."""
39188: (4)              return _mupdf.pdf_set_annot_icon_name(annot, name)
39189: (0)          def pdf_set_annot_ink_list(annot, n, count, v):
39190: (4)              r""" Class-aware wrapper for `::pdf_set_annot_ink_list()`."""
39191: (4)              return _mupdf.pdf_set_annot_ink_list(annot, n, count, v)
39192: (0)          def pdf_set_annot_intent(annot, it):
39193: (4)              r""" Class-aware wrapper for `::pdf_set_annot_intent()`."""
39194: (4)              return _mupdf.pdf_set_annot_intent(annot, it)
39195: (0)          def pdf_set_annot_interior_color(annot, n, color):
39196: (4)              r""" Class-aware wrapper for `::pdf_set_annot_interior_color()`."""
39197: (4)              return _mupdf.pdf_set_annot_interior_color(annot, n, color)
39198: (0)          def pdf_set_annot_is_open(annot, is_open):
39199: (4)              r""" Class-aware wrapper for `::pdf_set_annot_is_open()`."""
```

```
39200: (4)             return _mupdf.pdf_set_annot_is_open(annot, is_open)
39201: (0)         def pdf_set_annot_language(annot, lang):
39202: (4)             r""" Class-aware wrapper for `::pdf_set_annot_language()`."""
39203: (4)             return _mupdf.pdf_set_annot_language(annot, lang)
39204: (0)         def pdf_set_annot_line(annot, a, b):
39205: (4)             r""" Class-aware wrapper for `::pdf_set_annot_line()`."""
39206: (4)             return _mupdf.pdf_set_annot_line(annot, a, b)
39207: (0)         def pdf_set_annot_line_end_style(annot, e):
39208: (4)             r""" Class-aware wrapper for `::pdf_set_annot_line_end_style()`."""
39209: (4)             return _mupdf.pdf_set_annot_line_end_style(annot, e)
39210: (0)         def pdf_set_annot_line_ending_styles(annot, start_style, end_style):
39211: (4)             r""" Class-aware wrapper for `::pdf_set_annot_line_ending_styles()`."""
39212: (4)             return _mupdf.pdf_set_annot_line_ending_styles(annot, start_style,
end_style)
39213: (0)         def pdf_set_annot_line_start_style(annot, s):
39214: (4)             r""" Class-aware wrapper for `::pdf_set_annot_line_start_style()`."""
39215: (4)             return _mupdf.pdf_set_annot_line_start_style(annot, s)
39216: (0)         def pdf_set_annot_modification_date(annot, time):
39217: (4)             r""" Class-aware wrapper for `::pdf_set_annot_modification_date()`."""
39218: (4)             return _mupdf.pdf_set_annot_modification_date(annot, time)
39219: (0)         def pdf_set_annot_opacity(annot, opacity):
39220: (4)             r""" Class-aware wrapper for `::pdf_set_annot_opacity()`."""
39221: (4)             return _mupdf.pdf_set_annot_opacity(annot, opacity)
39222: (0)         def pdf_set_annot_popup(annot, rect):
39223: (4)             r""" Class-aware wrapper for `::pdf_set_annot_popup()`."""
39224: (4)             return _mupdf.pdf_set_annot_popup(annot, rect)
39225: (0)         def pdf_set_annot_quad_points(annot, n, qv):
39226: (4)             r""" Class-aware wrapper for `::pdf_set_annot_quad_points()`."""
39227: (4)             return _mupdf.pdf_set_annot_quad_points(annot, n, qv)
39228: (0)         def pdf_set_annot_quadding(annot, q):
39229: (4)             r""" Class-aware wrapper for `::pdf_set_annot_quadding()`."""
39230: (4)             return _mupdf.pdf_set_annot_quadding(annot, q)
39231: (0)         def pdf_set_annot_rect(annot, rect):
39232: (4)             r""" Class-aware wrapper for `::pdf_set_annot_rect()`."""
39233: (4)             return _mupdf.pdf_set_annot_rect(annot, rect)
39234: (0)         def pdf_set_annot_resynthesised(annot):
39235: (4)             r""" Class-aware wrapper for `::pdf_set_annot_resynthesised()`."""
39236: (4)             return _mupdf.pdf_set_annot_resynthesised(annot)
39237: (0)         def pdf_set_annot_stamp_image(annot, image):
39238: (4)             r""" Class-aware wrapper for `::pdf_set_annot_stamp_image()`."""
39239: (4)             return _mupdf.pdf_set_annot_stamp_image(annot, image)
39240: (0)         def pdf_set_annot_vertex(annot, i, p):
39241: (4)             r""" Class-aware wrapper for `::pdf_set_annot_vertex()`."""
39242: (4)             return _mupdf.pdf_set_annot_vertex(annot, i, p)
39243: (0)         def pdf_set_annot_vertices(annot, n, v):
39244: (4)             r""" Class-aware wrapper for `::pdf_set_annot_vertices()`."""
39245: (4)             return _mupdf.pdf_set_annot_vertices(annot, n, v)
39246: (0)         def pdf_set_choice_field_value(widget, value):
39247: (4)             r""" Class-aware wrapper for `::pdf_set_choice_field_value()`."""
39248: (4)             return _mupdf.pdf_set_choice_field_value(widget, value)
39249: (0)         def pdf_set_cmap_wmode(cmap, wmode):
39250: (4)             r""" Class-aware wrapper for `::pdf_set_cmap_wmode()`."""
39251: (4)             return _mupdf.pdf_set_cmap_wmode(cmap, wmode)
39252: (0)         def pdf_set_default_hmtx(font, w):
39253: (4)             r""" Class-aware wrapper for `::pdf_set_default_hmtx()`."""
39254: (4)             return _mupdf.pdf_set_default_hmtx(font, w)
39255: (0)         def pdf_set_default_vmtx(font, y, w):
39256: (4)             r""" Class-aware wrapper for `::pdf_set_default_vmtx()`."""
39257: (4)             return _mupdf.pdf_set_default_vmtx(font, y, w)
39258: (0)         def pdf_set_doc_event_callback(doc, event_cb, free_event_data_cb, data):
39259: (4)             r""" Class-aware wrapper for `::pdf_set_doc_event_callback()`."""
39260: (4)             return _mupdf.pdf_set_doc_event_callback(doc, event_cb,
free_event_data_cb, data)
39261: (0)         def pdf_set_document_language(doc, lang):
39262: (4)             r""" Class-aware wrapper for `::pdf_set_document_language()`."""
39263: (4)             return _mupdf.pdf_set_document_language(doc, lang)
39264: (0)         def pdf_set_field_value(doc, field, text, ignore_trigger_events):
39265: (4)             r""" Class-aware wrapper for `::pdf_set_field_value()`."""
39266: (4)             return _mupdf.pdf_set_field_value(doc, field, text, ignore_trigger_events)
```

```
39267: (0)          def pdf_set_font_wmode(font, wmode):
39268: (4)              r""" Class-aware wrapper for `::pdf_set_font_wmode()`."""
39269: (4)              return _mupdf.pdf_set_font_wmode(font, wmode)
39270: (0)          def pdf_set_int(obj, i):
39271: (4)              r""" Class-aware wrapper for `::pdf_set_int()`."""
39272: (4)              return _mupdf.pdf_set_int(obj, i)
39273: (0)          def pdf_set_layer_config_as_default(doc):
39274: (4)              r""" Class-aware wrapper for `::pdf_set_layer_config_as_default()`."""
39275: (4)              return _mupdf.pdf_set_layer_config_as_default(doc)
39276: (0)          def pdf_set_obj_memo(obj, bit, memo):
39277: (4)              r""" Class-aware wrapper for `::pdf_set_obj_memo()`."""
39278: (4)              return _mupdf.pdf_set_obj_memo(obj, bit, memo)
39279: (0)          def pdf_set_obj_parent(obj, num):
39280: (4)              r""" Class-aware wrapper for `::pdf_set_obj_parent()`."""
39281: (4)              return _mupdf.pdf_set_obj_parent(obj, num)
39282: (0)          def pdf_set_page_box(page, box, rect):
39283: (4)              r""" Class-aware wrapper for `::pdf_set_page_box()`."""
39284: (4)              return _mupdf.pdf_set_page_box(page, box, rect)
39285: (0)          def pdf_set_page_labels(doc, index, style, prefix, start):
39286: (4)              r""" Class-aware wrapper for `::pdf_set_page_labels()`."""
39287: (4)              return _mupdf.pdf_set_page_labels(doc, index, style, prefix, start)
39288: (0)          def pdf_set_populating_xref_trailer(doc, trailer):
39289: (4)              r""" Class-aware wrapper for `::pdf_set_populating_xref_trailer()`."""
39290: (4)              return _mupdf.pdf_set_populating_xref_trailer(doc, trailer)
39291: (0)          def pdf_set_str_len(obj, newlen):
39292: (4)              r""" Class-aware wrapper for `::pdf_set_str_len()`."""
39293: (4)              return _mupdf.pdf_set_str_len(obj, newlen)
39294: (0)          def pdf_set_text_field_value(widget, value):
39295: (4)              r""" Class-aware wrapper for `::pdf_set_text_field_value()`."""
39296: (4)              return _mupdf.pdf_set_text_field_value(widget, value)
39297: (0)          def pdf_set_usecmap(cmap, usecmap):
39298: (4)              r""" Class-aware wrapper for `::pdf_set_usecmap()`."""
39299: (4)              return _mupdf.pdf_set_usecmap(cmap, usecmap)
39300: (0)          def pdf_set_widget_editing_state(widget, editing):
39301: (4)              r""" Class-aware wrapper for `::pdf_set_widget_editing_state()`."""
39302: (4)              return _mupdf.pdf_set_widget_editing_state(widget, editing)
39303: (0)          def pdf_sign_signature(widget, signer, appearance_flags, graphic, reason,
location):
39304: (4)              r""" Class-aware wrapper for `::pdf_sign_signature()`."""
39305: (4)              return _mupdf.pdf_sign_signature(widget, signer, appearance_flags,
graphic, reason, location)
39306: (0)          def pdf_sign_signature_with_appearance(widget, signer, date, disp_list):
39307: (4)              r""" Class-aware wrapper for `::pdf_sign_signature_with_appearance()`."""
39308: (4)              return _mupdf.pdf_sign_signature_with_appearance(widget, signer, date,
disp_list)
39309: (0)          def pdf_signature_appearance_signed(rect, lang, img, left_text, right_text,
include_logo):
39310: (4)              r""" Class-aware wrapper for `::pdf_signature_appearance_signed()`."""
39311: (4)              return _mupdf.pdf_signature_appearance_signed(rect, lang, img, left_text,
right_text, include_logo)
39312: (0)          def pdf_signature_appearance_unsigned(rect, lang):
39313: (4)              r""" Class-aware wrapper for `::pdf_signature_appearance_unsigned()`."""
39314: (4)              return _mupdf.pdf_signature_appearance_unsigned(rect, lang)
39315: (0)          def pdf_signature_byte_range(doc, signature, byte_range):
39316: (4)              r""" Class-aware wrapper for `::pdf_signature_byte_range()`."""
39317: (4)              return _mupdf.pdf_signature_byte_range(doc, signature, byte_range)
39318: (0)          def pdf_signature_contents(doc, signature, contents):
39319: (4)              r"""
39320: (4)              Class-aware wrapper for `::pdf_signature_contents()`.
39321: (4)              This function has out-params. Python/C# wrappers look like:
39322: (8)                  `pdf_signature_contents(::pdf_document *doc, ::pdf_obj *signature)` =>
`(size_t, char *contents)`
39323: (4)              """
39324: (4)              return _mupdf.pdf_signature_contents(doc, signature, contents)
39325: (0)          def pdf_signature_drop_distinguished_name(name):
39326: (4)              r""" Class-aware wrapper for
`::pdf_signature_drop_distinguished_name()`."""
39327: (4)              return _mupdf.pdf_signature_drop_distinguished_name(name)
39328: (0)          def pdf_signature_error_description(err):
```

```
39329: (4)             r""" Class-aware wrapper for `::pdf_signature_error_description()`."""
39330: (4)             return _mupdf.pdf_signature_error_description(err)
39331: (0)         def pdf_signature_format_distinguished_name(name):
39332: (4)             r""" Class-aware wrapper for
`::pdf_signature_format_distinguished_name()`."""
39333: (4)             return _mupdf.pdf_signature_format_distinguished_name(name)
39334: (0)         def pdf_signature_hash_bytes(doc, signature):
39335: (4)             r""" Class-aware wrapper for `::pdf_signature_hash_bytes()`."""
39336: (4)             return _mupdf.pdf_signature_hash_bytes(doc, signature)
39337: (0)         def pdf_signature_incremental_change_since_signing(doc, signature):
39338: (4)             r""" Class-aware wrapper for
`::pdf_signature_incremental_change_since_signing()`."""
39339: (4)             return _mupdf.pdf_signature_incremental_change_since_signing(doc,
signature)
39340: (0)         def pdf_signature_info(name, dn, reason, location, date, include_labels):
39341: (4)             r""" Class-aware wrapper for `::pdf_signature_info()`."""
39342: (4)             return _mupdf.pdf_signature_info(name, dn, reason, location, date,
include_labels)
39343: (0)         def pdf_signature_is_signed(doc, field):
39344: (4)             r""" Class-aware wrapper for `::pdf_signature_is_signed()`."""
39345: (4)             return _mupdf.pdf_signature_is_signed(doc, field)
39346: (0)         def pdf_signature_set_value(doc, field, signer, stime):
39347: (4)             r""" Class-aware wrapper for `::pdf_signature_set_value()`."""
39348: (4)             return _mupdf.pdf_signature_set_value(doc, field, signer, stime)
39349: (0)         def pdf_sort_cmap(cmap):
39350: (4)             r""" Class-aware wrapper for `::pdf_sort_cmap()`."""
39351: (4)             return _mupdf.pdf_sort_cmap(cmap)
39352: (0)         def pdf_sort_dict(dict):
39353: (4)             r""" Class-aware wrapper for `::pdf_sort_dict()`."""
39354: (4)             return _mupdf.pdf_sort_dict(dict)
39355: (0)         def pdf_specifics(doc):
39356: (4)             r""" Class-aware wrapper for `::pdf_specifics()`."""
39357: (4)             return _mupdf.pdf_specifics(doc)
39358: (0)         def pdf_sprint_obj(buf, cap, len, obj, tight, ascii):
39359: (4)             r"""
39360: (4)             Class-aware wrapper for `::pdf_sprint_obj()`.
39361: (4)             This function has out-params. Python/C# wrappers look like:
39362: (8)                 `pdf_sprint_obj(char *buf, size_t cap, ::pdf_obj *obj, int tight, int
ascii)` => `(char *, size_t len)`
39363: (4)             """
39364: (4)             return _mupdf.pdf_sprint_obj(buf, cap, len, obj, tight, ascii)
39365: (0)         def pdf_store_item(key, val, itemsize):
39366: (4)             r""" Class-aware wrapper for `::pdf_store_item()`."""
39367: (4)             return _mupdf.pdf_store_item(key, val, itemsize)
39368: (0)         def pdf_string_from_annot_type(type):
39369: (4)             r""" Class-aware wrapper for `::pdf_string_from_annot_type()`."""
39370: (4)             return _mupdf.pdf_string_from_annot_type(type)
39371: (0)         def pdf_string_from_intent(intent):
39372: (4)             r""" Class-aware wrapper for `::pdf_string_from_intent()`."""
39373: (4)             return _mupdf.pdf_string_from_intent(intent)
39374: (0)         def pdf_string_from_line_ending(end):
39375: (4)             r""" Class-aware wrapper for `::pdf_string_from_line_ending()`."""
39376: (4)             return _mupdf.pdf_string_from_line_ending(end)
39377: (0)         def pdf_structure_type(role_map, tag):
39378: (4)             r""" Class-aware wrapper for `::pdf_structure_type()`."""
39379: (4)             return _mupdf.pdf_structure_type(role_map, tag)
39380: (0)         def pdf_subset_fonts(doc, pages_len, pages):
39381: (4)             r""" Class-aware wrapper for `::pdf_subset_fonts()`."""
39382: (4)             return _mupdf.pdf_subset_fonts(doc, pages_len, pages)
39383: (0)         def pdf_subset_fonts2(doc, pages):
39384: (4)             r""" Class-aware wrapper for `::pdf_subset_fonts2()`.   Swig-friendly
wrapper for pdf_subset_fonts()."""
39385: (4)             return _mupdf.pdf_subset_fonts2(doc, pages)
39386: (0)         def pdf_text_widget_format(tw):
39387: (4)             r""" Class-aware wrapper for `::pdf_text_widget_format()`."""
39388: (4)             return _mupdf.pdf_text_widget_format(tw)
39389: (0)         def pdf_text_widget_max_len(tw):
39390: (4)             r""" Class-aware wrapper for `::pdf_text_widget_max_len()`."""
39391: (4)             return _mupdf.pdf_text_widget_max_len(tw)
```

```
39392: (0)          def pdf_to_bool(obj):
39393: (4)              r""" Class-aware wrapper for `::pdf_to_bool()`."""
39394: (4)              return _mupdf.pdf_to_bool(obj)
39395: (0)          def pdf_to_bool_default(obj, _def):
39396: (4)              r""" Class-aware wrapper for `::pdf_to_bool_default()`."""
39397: (4)              return _mupdf.pdf_to_bool_default(obj, _def)
39398: (0)          def pdf_to_date(time):
39399: (4)              r""" Class-aware wrapper for `::pdf_to_date()`."""
39400: (4)              return _mupdf.pdf_to_date(time)
39401: (0)          def pdf_to_gen(obj):
39402: (4)              r""" Class-aware wrapper for `::pdf_to_gen()`."""
39403: (4)              return _mupdf.pdf_to_gen(obj)
39404: (0)          def pdf_to_int(obj):
39405: (4)              r""" Class-aware wrapper for `::pdf_to_int()`."""
39406: (4)              return _mupdf.pdf_to_int(obj)
39407: (0)          def pdf_to_int64(obj):
39408: (4)              r""" Class-aware wrapper for `::pdf_to_int64()`."""
39409: (4)              return _mupdf.pdf_to_int64(obj)
39410: (0)          def pdf_to_int_default(obj, _def):
39411: (4)              r""" Class-aware wrapper for `::pdf_to_int_default()`."""
39412: (4)              return _mupdf.pdf_to_int_default(obj, _def)
39413: (0)          def pdf_to_matrix(array):
39414: (4)              r""" Class-aware wrapper for `::pdf_to_matrix()`."""
39415: (4)              return _mupdf.pdf_to_matrix(array)
39416: (0)          def pdf_to_name(obj):
39417: (4)              r""" Class-aware wrapper for `::pdf_to_name()`."""
39418: (4)              return _mupdf.pdf_to_name(obj)
39419: (0)          def pdf_to_num(obj):
39420: (4)              r""" Class-aware wrapper for `::pdf_to_num()`."""
39421: (4)              return _mupdf.pdf_to_num(obj)
39422: (0)          def pdf_to_quad(array, offset):
39423: (4)              r""" Class-aware wrapper for `::pdf_to_quad()`."""
39424: (4)              return _mupdf.pdf_to_quad(array, offset)
39425: (0)          def pdf_to_real(obj):
39426: (4)              r""" Class-aware wrapper for `::pdf_to_real()`."""
39427: (4)              return _mupdf.pdf_to_real(obj)
39428: (0)          def pdf_to_real_default(obj, _def):
39429: (4)              r""" Class-aware wrapper for `::pdf_to_real_default()`."""
39430: (4)              return _mupdf.pdf_to_real_default(obj, _def)
39431: (0)          def pdf_to_rect(array):
39432: (4)              r""" Class-aware wrapper for `::pdf_to_rect()`."""
39433: (4)              return _mupdf.pdf_to_rect(array)
39434: (0)          def pdf_to_str_buf(obj):
39435: (4)              r""" Class-aware wrapper for `::pdf_to_str_buf()`."""
39436: (4)              return _mupdf.pdf_to_str_buf(obj)
39437: (0)          def pdf_to_str_len(obj):
39438: (4)              r""" Class-aware wrapper for `::pdf_to_str_len()`."""
39439: (4)              return _mupdf.pdf_to_str_len(obj)
39440: (0)          def pdf_to_string(obj, sizep):
39441: (4)              r"""
39442: (4)              Class-aware wrapper for `::pdf_to_string()`.
39443: (4)              This function has out-params. Python/C# wrappers look like:
39444: (8)                  `pdf_to_string(::pdf_obj *obj)` => `(const char *, size_t sizep)`
39445: (4)              """
39446: (4)              return _mupdf.pdf_to_string(obj, sizep)
39447: (0)          def pdf_to_text_string(obj):
39448: (4)              r""" Class-aware wrapper for `::pdf_to_text_string()`."""
39449: (4)              return _mupdf.pdf_to_text_string(obj)
39450: (0)          def pdf_toggle_layer_config_ui(doc, ui):
39451: (4)              r""" Class-aware wrapper for `::pdf_toggle_layer_config_ui()`."""
39452: (4)              return _mupdf.pdf_toggle_layer_config_ui(doc, ui)
39453: (0)          def pdf_toggle_widget(widget):
39454: (4)              r""" Class-aware wrapper for `::pdf_toggle_widget()`."""
39455: (4)              return _mupdf.pdf_toggle_widget(widget)
39456: (0)          def pdf_tos_get_text(tos):
39457: (4)              r""" Class-aware wrapper for `::pdf_tos_get_text()`."""
39458: (4)              return _mupdf.pdf_tos_get_text(tos)
39459: (0)          def pdf_tos_make_trm(tos, text, fontdesc, cid, trm):
39460: (4)              r""" Class-aware wrapper for `::pdf_tos_make_trm()`."""
```

```
39461: (4)             return _mupdf.pdf_tos_make_trm(tos, text, fontdesc, cid, trm)
39462: (0)         def pdf_tos_move_after_char(tos):
39463: (4)             r""" Class-aware wrapper for `::pdf_tos_move_after_char()`."""
39464: (4)             return _mupdf.pdf_tos_move_after_char(tos)
39465: (0)         def pdf_tos_newline(tos, leading):
39466: (4)             r""" Class-aware wrapper for `::pdf_tos_newline()`."""
39467: (4)             return _mupdf.pdf_tos_newline(tos, leading)
39468: (0)         def pdf_tos_reset(tos, render):
39469: (4)             r""" Class-aware wrapper for `::pdf_tos_reset()`."""
39470: (4)             return _mupdf.pdf_tos_reset(tos, render)
39471: (0)         def pdf_tos_restore(tos, save):
39472: (4)             r""" Class-aware wrapper for `::pdf_tos_restore()`."""
39473: (4)             return _mupdf.pdf_tos_restore(tos, save)
39474: (0)         def pdf_tos_save(tos, save):
39475: (4)             r""" Class-aware wrapper for `::pdf_tos_save()`."""
39476: (4)             return _mupdf.pdf_tos_save(tos, save)
39477: (0)         def pdf_tos_set_matrix(tos, a, b, c, d, e, f):
39478: (4)             r""" Class-aware wrapper for `::pdf_tos_set_matrix()`."""
39479: (4)             return _mupdf.pdf_tos_set_matrix(tos, a, b, c, d, e, f)
39480: (0)         def pdf_tos_translate(tos, tx, ty):
39481: (4)             r""" Class-aware wrapper for `::pdf_tos_translate()`."""
39482: (4)             return _mupdf.pdf_tos_translate(tos, tx, ty)
39483: (0)         def pdf_trailer(doc):
39484: (4)             r""" Class-aware wrapper for `::pdf_trailer()`."""
39485: (4)             return _mupdf.pdf_trailer(doc)
39486: (0)         def pdf_undo(doc):
39487: (4)             r""" Class-aware wrapper for `::pdf_undo()`."""
39488: (4)             return _mupdf.pdf_undo(doc)
39489: (0)         def pdf_undoredo_state(doc, steps):
39490: (4)             r"""
39491: (4)             Class-aware wrapper for `::pdf_undoredo_state()`.
39492: (4)             This function has out-params. Python/C# wrappers look like:
39493: (8)                 `pdf_undoredo_state(::pdf_document *doc)` => `(int, int steps)`
39494: (4)             """
39495: (4)             return _mupdf.pdf_undoredo_state(doc, steps)
39496: (0)         def pdf_undoredo_step(doc, step):
39497: (4)             r""" Class-aware wrapper for `::pdf_undoredo_step()`."""
39498: (4)             return _mupdf.pdf_undoredo_step(doc, step)
39499: (0)         def pdf_unmark_obj(obj):
39500: (4)             r""" Class-aware wrapper for `::pdf_unmark_obj()`."""
39501: (4)             return _mupdf.pdf_unmark_obj(obj)
39502: (0)         def pdf_update_annot(annot):
39503: (4)             r""" Class-aware wrapper for `::pdf_update_annot()`."""
39504: (4)             return _mupdf.pdf_update_annot(annot)
39505: (0)         def pdf_update_default_colorspaces(old_cs, res):
39506: (4)             r""" Class-aware wrapper for `::pdf_update_default_colorspaces()`."""
39507: (4)             return _mupdf.pdf_update_default_colorspaces(old_cs, res)
39508: (0)         def pdf_update_object(doc, num, obj):
39509: (4)             r""" Class-aware wrapper for `::pdf_update_object()`."""
39510: (4)             return _mupdf.pdf_update_object(doc, num, obj)
39511: (0)         def pdf_update_page(page):
39512: (4)             r""" Class-aware wrapper for `::pdf_update_page()`."""
39513: (4)             return _mupdf.pdf_update_page(page)
39514: (0)         def pdf_update_stream(doc, ref, buf, compressed):
39515: (4)             r""" Class-aware wrapper for `::pdf_update_stream()`."""
39516: (4)             return _mupdf.pdf_update_stream(doc, ref, buf, compressed)
39517: (0)         def pdf_update_widget(widget):
39518: (4)             r""" Class-aware wrapper for `::pdf_update_widget()`."""
39519: (4)             return _mupdf.pdf_update_widget(widget)
39520: (0)         def pdf_update_xobject(doc, xobj, bbox, mat, res, buffer):
39521: (4)             r""" Class-aware wrapper for `::pdf_update_xobject()`."""
39522: (4)             return _mupdf.pdf_update_xobject(doc, xobj, bbox, mat, res, buffer)
39523: (0)         def pdf_validate_change_history(doc):
39524: (4)             r""" Class-aware wrapper for `::pdf_validate_change_history()`."""
39525: (4)             return _mupdf.pdf_validate_change_history(doc)
39526: (0)         def pdf_validate_changes(doc, version):
39527: (4)             r""" Class-aware wrapper for `::pdf_validate_changes()`."""
39528: (4)             return _mupdf.pdf_validate_changes(doc, version)
39529: (0)         def pdf_validate_signature(widget):
```

```
39530: (4)             r""" Class-aware wrapper for `::pdf_validate_signature()`."""
39531: (4)             return _mupdf.pdf_validate_signature(widget)
39532: (0)         def pdf_verify_embedded_file_checksum(fs):
39533: (4)             r""" Class-aware wrapper for `::pdf_verify_embedded_file_checksum()`."""
39534: (4)             return _mupdf.pdf_verify_embedded_file_checksum(fs)
39535: (0)         def pdf_version(doc):
39536: (4)             r""" Class-aware wrapper for `::pdf_version()`."""
39537: (4)             return _mupdf.pdf_version(doc)
39538: (0)         def pdf_walk_tree(tree, kid_name, arrive, leave, arg, names, values):
39539: (4)             r"""
39540: (4)             Class-aware wrapper for `::pdf_walk_tree()`.
39541: (4)             This function has out-params. Python/C# wrappers look like:
39542: (8)                 `pdf_walk_tree(::pdf_obj *tree, ::pdf_obj *kid_name, void (*arrive)
(::fz_context *, ::pdf_obj *, void *, ::pdf_obj **), void (*leave)(::fz_context *, ::pdf_obj *,
void *), void *arg, ::pdf_obj **names, ::pdf_obj **values)` => `()`
39543: (4)             """
39544: (4)             return _mupdf.pdf_walk_tree(tree, kid_name, arrive, leave, arg, names,
values)
39545: (0)         def pdf_was_pure_xfa(doc):
39546: (4)             r""" Class-aware wrapper for `::pdf_was_pure_xfa()`."""
39547: (4)             return _mupdf.pdf_was_pure_xfa(doc)
39548: (0)         def pdf_was_repaired(doc):
39549: (4)             r""" Class-aware wrapper for `::pdf_was_repaired()`."""
39550: (4)             return _mupdf.pdf_was_repaired(doc)
39551: (0)         def pdf_widget_is_readonly(widget):
39552: (4)             r""" Class-aware wrapper for `::pdf_widget_is_readonly()`."""
39553: (4)             return _mupdf.pdf_widget_is_readonly(widget)
39554: (0)         def pdf_widget_is_signed(widget):
39555: (4)             r""" Class-aware wrapper for `::pdf_widget_is_signed()`."""
39556: (4)             return _mupdf.pdf_widget_is_signed(widget)
39557: (0)         def pdf_widget_type(widget):
39558: (4)             r""" Class-aware wrapper for `::pdf_widget_type()`."""
39559: (4)             return _mupdf.pdf_widget_type(widget)
39560: (0)         def pdf_write_digest(out, byte_range, field, digest_offset, digest_length,
signer):
39561: (4)             r""" Class-aware wrapper for `::pdf_write_digest()`."""
39562: (4)             return _mupdf.pdf_write_digest(out, byte_range, field, digest_offset,
digest_length, signer)
39563: (0)         def pdf_write_document(doc, out, opts):
39564: (4)             r""" Class-aware wrapper for `::pdf_write_document()`."""
39565: (4)             return _mupdf.pdf_write_document(doc, out, opts)
39566: (0)         def pdf_write_journal(doc, out):
39567: (4)             r""" Class-aware wrapper for `::pdf_write_journal()`."""
39568: (4)             return _mupdf.pdf_write_journal(doc, out)
39569: (0)         def pdf_write_snapshot(doc, out):
39570: (4)             r""" Class-aware wrapper for `::pdf_write_snapshot()`."""
39571: (4)             return _mupdf.pdf_write_snapshot(doc, out)
39572: (0)         def pdf_xobject_bbox(xobj):
39573: (4)             r""" Class-aware wrapper for `::pdf_xobject_bbox()`."""
39574: (4)             return _mupdf.pdf_xobject_bbox(xobj)
39575: (0)         def pdf_xobject_colorspace(xobj):
39576: (4)             r""" Class-aware wrapper for `::pdf_xobject_colorspace()`."""
39577: (4)             return _mupdf.pdf_xobject_colorspace(xobj)
39578: (0)         def pdf_xobject_isolated(xobj):
39579: (4)             r""" Class-aware wrapper for `::pdf_xobject_isolated()`."""
39580: (4)             return _mupdf.pdf_xobject_isolated(xobj)
39581: (0)         def pdf_xobject_knockout(xobj):
39582: (4)             r""" Class-aware wrapper for `::pdf_xobject_knockout()`."""
39583: (4)             return _mupdf.pdf_xobject_knockout(xobj)
39584: (0)         def pdf_xobject_matrix(xobj):
39585: (4)             r""" Class-aware wrapper for `::pdf_xobject_matrix()`."""
39586: (4)             return _mupdf.pdf_xobject_matrix(xobj)
39587: (0)         def pdf_xobject_resources(xobj):
39588: (4)             r""" Class-aware wrapper for `::pdf_xobject_resources()`."""
39589: (4)             return _mupdf.pdf_xobject_resources(xobj)
39590: (0)         def pdf_xobject_transparency(xobj):
39591: (4)             r""" Class-aware wrapper for `::pdf_xobject_transparency()`."""
39592: (4)             return _mupdf.pdf_xobject_transparency(xobj)
39593: (0)         def pdf_xref_ensure_incremental_object(doc, num):
```

```
39594: (4)              r""" Class-aware wrapper for `::pdf_xref_ensure_incremental_object()`."""
39595: (4)              return _mupdf.pdf_xref_ensure_incremental_object(doc, num)
39596: (0)          def pdf_xref_ensure_local_object(doc, num):
39597: (4)              r""" Class-aware wrapper for `::pdf_xref_ensure_local_object()`."""
39598: (4)              return _mupdf.pdf_xref_ensure_local_object(doc, num)
39599: (0)          def pdf_xref_entry_map(doc, fn, arg):
39600: (4)              r""" Class-aware wrapper for `::pdf_xref_entry_map()`."""
39601: (4)              return _mupdf.pdf_xref_entry_map(doc, fn, arg)
39602: (0)          def pdf_xref_is_incremental(doc, num):
39603: (4)              r""" Class-aware wrapper for `::pdf_xref_is_incremental()`."""
39604: (4)              return _mupdf.pdf_xref_is_incremental(doc, num)
39605: (0)          def pdf_xref_len(doc):
39606: (4)              r""" Class-aware wrapper for `::pdf_xref_len()`."""
39607: (4)              return _mupdf.pdf_xref_len(doc)
39608: (0)          def pdf_xref_obj_is_unsaved_signature(doc, obj):
39609: (4)              r""" Class-aware wrapper for `::pdf_xref_obj_is_unsaved_signature()`."""
39610: (4)              return _mupdf.pdf_xref_obj_is_unsaved_signature(doc, obj)
39611: (0)          def pdf_xref_remove_unsaved_signature(doc, field):
39612: (4)              r""" Class-aware wrapper for `::pdf_xref_remove_unsaved_signature()`."""
39613: (4)              return _mupdf.pdf_xref_remove_unsaved_signature(doc, field)
39614: (0)          def pdf_xref_store_unsaved_signature(doc, field, signer):
39615: (4)              r""" Class-aware wrapper for `::pdf_xref_store_unsaved_signature()`."""
39616: (4)              return _mupdf.pdf_xref_store_unsaved_signature(doc, field, signer)
39617: (0)          def internal_env_flag(name):
39618: (4)              r""" Internal use only. Looks at environmental variable <name>; returns 0
if unset else int value."""
39619: (4)              return _mupdf.internal_env_flag(name)
39620: (0)          def internal_env_flag_check_unset(if_, name):
39621: (4)              r""" Internal use only. Looks at environmental variable <name>; returns 0
if unset else int value."""
39622: (4)              return _mupdf.internal_env_flag_check_unset(if_, name)
39623: (0)          def internal_context_get():
39624: (4)              r""" Internal use only. Returns `fz_context*` for use by current
thread."""
39625: (4)              return _mupdf.internal_context_get()
39626: (0)          class FzErrorBase(Exception):
39627: (4)              r""" Base class for exceptions."""
39628: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39629: (4)              __repr__ = _swig_repr
39630: (4)              m_code = property(_mupdf.FzErrorBase_m_code_get,
_mupdf.FzErrorBase_m_code_set)
39631: (4)              m_text = property(_mupdf.FzErrorBase_m_text_get,
_mupdf.FzErrorBase_m_text_set)
39632: (4)              m_what = property(_mupdf.FzErrorBase_m_what_get,
_mupdf.FzErrorBase_m_what_set)
39633: (4)              def what(self):
39634: (8)                  return _mupdf.FzErrorBase_what(self)
39635: (4)              def __init__(self, code, text):
39636: (8)                  _mupdf.FzErrorBase_swiginit(self, _mupdf.new_FzErrorBase(code, text))
39637: (4)              __swig_destroy__ = _mupdf.delete_FzErrorBase
39638: (0)          _mupdf.FzErrorBase_swigregister(FzErrorBase)
39639: (0)          class FzErrorNone(FzErrorBase):
39640: (4)              r""" For `FZ_ERROR_NONE`."""
39641: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39642: (4)              __repr__ = _swig_repr
39643: (4)              def __init__(self, message):
39644: (8)                  _mupdf.FzErrorNone_swiginit(self, _mupdf.new_FzErrorNone(message))
39645: (4)              __swig_destroy__ = _mupdf.delete_FzErrorNone
39646: (0)          _mupdf.FzErrorNone_swigregister(FzErrorNone)
39647: (0)          class FzErrorGeneric(FzErrorBase):
39648: (4)              r""" For `FZ_ERROR_GENERIC`."""
39649: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39650: (4)              __repr__ = _swig_repr
39651: (4)              def __init__(self, message):
39652: (8)                  _mupdf.FzErrorGeneric_swiginit(self,
_mupdf.new_FzErrorGeneric(message))
```

```
39653: (4)              __swig_destroy__ = _mupdf.delete_FzErrorGeneric
39654: (0)          _mupdf.FzErrorGeneric_swigregister(FzErrorGeneric)
39655: (0)          class FzErrorSystem(FzErrorBase):
39656: (4)              r""" For `FZ_ERROR_SYSTEM`."""
39657: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39658: (4)              __repr__ = _swig_repr
39659: (4)              def __init__(self, message):
39660: (8)                  _mupdf.FzErrorSystem_swiginit(self, _mupdf.new_FzErrorSystem(message))
39661: (4)              __swig_destroy__ = _mupdf.delete_FzErrorSystem
39662: (0)          _mupdf.FzErrorSystem_swigregister(FzErrorSystem)
39663: (0)          class FzErrorLibrary(FzErrorBase):
39664: (4)              r""" For `FZ_ERROR_LIBRARY`."""
39665: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39666: (4)              __repr__ = _swig_repr
39667: (4)              def __init__(self, message):
39668: (8)                  _mupdf.FzErrorLibrary_swiginit(self,
_mupdf.new_FzErrorLibrary(message))
39669: (4)              __swig_destroy__ = _mupdf.delete_FzErrorLibrary
39670: (0)          _mupdf.FzErrorLibrary_swigregister(FzErrorLibrary)
39671: (0)          class FzErrorArgument(FzErrorBase):
39672: (4)              r""" For `FZ_ERROR_ARGUMENT`."""
39673: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39674: (4)              __repr__ = _swig_repr
39675: (4)              def __init__(self, message):
39676: (8)                  _mupdf.FzErrorArgument_swiginit(self,
_mupdf.new_FzErrorArgument(message))
39677: (4)              __swig_destroy__ = _mupdf.delete_FzErrorArgument
39678: (0)          _mupdf.FzErrorArgument_swigregister(FzErrorArgument)
39679: (0)          class FzErrorLimit(FzErrorBase):
39680: (4)              r""" For `FZ_ERROR_LIMIT`."""
39681: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39682: (4)              __repr__ = _swig_repr
39683: (4)              def __init__(self, message):
39684: (8)                  _mupdf.FzErrorLimit_swiginit(self, _mupdf.new_FzErrorLimit(message))
39685: (4)              __swig_destroy__ = _mupdf.delete_FzErrorLimit
39686: (0)          _mupdf.FzErrorLimit_swigregister(FzErrorLimit)
39687: (0)          class FzErrorUnsupported(FzErrorBase):
39688: (4)              r""" For `FZ_ERROR_UNSUPPORTED`."""
39689: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39690: (4)              __repr__ = _swig_repr
39691: (4)              def __init__(self, message):
39692: (8)                  _mupdf.FzErrorUnsupported_swiginit(self,
_mupdf.new_FzErrorUnsupported(message))
39693: (4)              __swig_destroy__ = _mupdf.delete_FzErrorUnsupported
39694: (0)          _mupdf.FzErrorUnsupported_swigregister(FzErrorUnsupported)
39695: (0)          class FzErrorFormat(FzErrorBase):
39696: (4)              r""" For `FZ_ERROR_FORMAT`."""
39697: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39698: (4)              __repr__ = _swig_repr
39699: (4)              def __init__(self, message):
39700: (8)                  _mupdf.FzErrorFormat_swiginit(self, _mupdf.new_FzErrorFormat(message))
39701: (4)              __swig_destroy__ = _mupdf.delete_FzErrorFormat
39702: (0)          _mupdf.FzErrorFormat_swigregister(FzErrorFormat)
39703: (0)          class FzErrorSyntax(FzErrorBase):
39704: (4)              r""" For `FZ_ERROR_SYNTAX`."""
39705: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39706: (4)              __repr__ = _swig_repr
39707: (4)              def __init__(self, message):
39708: (8)                  _mupdf.FzErrorSyntax_swiginit(self, _mupdf.new_FzErrorSyntax(message))
39709: (4)              __swig_destroy__ = _mupdf.delete_FzErrorSyntax
39710: (0)          _mupdf.FzErrorSyntax_swigregister(FzErrorSyntax)
39711: (0)          class FzErrorTrylater(FzErrorBase):
```

```
39712: (4)              r""" For `FZ_ERROR_TRYLATER`."""
39713: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39714: (4)              __repr__ = _swig_repr
39715: (4)              def __init__(self, message):
39716: (8)                  _mupdf.FzErrorTrylater_swiginit(self,
_mupdf.new_FzErrorTrylater(message))
39717: (4)              __swig_destroy__ = _mupdf.delete_FzErrorTrylater
39718: (0)          _mupdf.FzErrorTrylater_swigregister(FzErrorTrylater)
39719: (0)          class FzErrorAbort(FzErrorBase):
39720: (4)              r""" For `FZ_ERROR_ABORT`."""
39721: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39722: (4)              __repr__ = _swig_repr
39723: (4)              def __init__(self, message):
39724: (8)                  _mupdf.FzErrorAbort_swiginit(self, _mupdf.new_FzErrorAbort(message))
39725: (4)              __swig_destroy__ = _mupdf.delete_FzErrorAbort
39726: (0)          _mupdf.FzErrorAbort_swigregister(FzErrorAbort)
39727: (0)          class FzErrorRepaired(FzErrorBase):
39728: (4)              r""" For `FZ_ERROR_REPAIRED`."""
39729: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39730: (4)              __repr__ = _swig_repr
39731: (4)              def __init__(self, message):
39732: (8)                  _mupdf.FzErrorRepaired_swiginit(self,
_mupdf.new_FzErrorRepaired(message))
39733: (4)              __swig_destroy__ = _mupdf.delete_FzErrorRepaired
39734: (0)          _mupdf.FzErrorRepaired_swigregister(FzErrorRepaired)
39735: (0)          def internal_throw_exception(ctx):
39736: (4)              r""" Throw exception appropriate for error in `ctx`."""
39737: (4)              return _mupdf.internal_throw_exception(ctx)
39738: (0)          def to_stdstring(s):
39739: (4)              return _mupdf.to_stdstring(s)
39740: (0)          def py_repr(x):
39741: (4)              return _mupdf.py_repr(x)
39742: (0)          def py_str(x):
39743: (4)              return _mupdf.py_str(x)
39744: (0)          def ll_fz_buffer_to_bytes_internal(buffer, clear):
39745: (4)              return _mupdf.ll_fz_buffer_to_bytes_internal(buffer, clear)
39746: (0)          def python_memoryview_from_memory(data, size, writable):
39747: (4)              return _mupdf.python_memoryview_from_memory(data, size, writable)
39748: (0)          def ll_fz_buffer_storage_memoryview(buffer, writable):
39749: (4)              return _mupdf.ll_fz_buffer_storage_memoryview(buffer, writable)
39750: (0)          def raw_to_python_bytes(*args):
39751: (4)              return _mupdf.raw_to_python_bytes(*args)
39752: (0)          def python_buffer_data(PYTHON_BUFFER_DATA):
39753: (4)              return _mupdf.python_buffer_data(PYTHON_BUFFER_DATA)
39754: (0)          def python_mutable_buffer_data(PYTHON_BUFFER_MUTABLE_DATA):
39755: (4)              return _mupdf.python_mutable_buffer_data(PYTHON_BUFFER_MUTABLE_DATA)
39756: (0)          def obj_enum_to_obj(n):
39757: (4)              return _mupdf.obj_enum_to_obj(n)
39758: (0)          def ll_pdf_set_annot_color2(annot, n, color0, color1, color2, color3):
39759: (4)              return _mupdf.ll_pdf_set_annot_color2(annot, n, color0, color1, color2,
color3)
39760: (0)          def ll_pdf_set_annot_interior_color2(annot, n, color0, color1, color2,
color3):
39761: (4)              return _mupdf.ll_pdf_set_annot_interior_color2(annot, n, color0, color1,
color2, color3)
39762: (0)          def ll_fz_fill_text2(dev, text, ctm, colorspace, color0, color1, color2,
color3, alpha, color_params):
39763: (4)              return _mupdf.ll_fz_fill_text2(dev, text, ctm, colorspace, color0, color1,
color2, color3, alpha, color_params)
39764: (0)          def fz_memrnd2(length):
39765: (4)              return _mupdf.fz_memrnd2(length)
39766: (0)          def ll_fz_pixmap_copy(pm, src, n):
39767: (4)              return _mupdf.ll_fz_pixmap_copy(pm, src, n)
39768: (0)          def ll_fz_pixmap_copy_raw(pm, samples):
39769: (4)              return _mupdf.ll_fz_pixmap_copy_raw(pm, samples)
39770: (0)          def fz_runetochar2(rune):
```

```
39771: (4)              return _mupdf.fz_runetochar2(rune)
39772: (0)          def ll_fz_make_bookmark2(doc, loc):
39773: (4)              return _mupdf.ll_fz_make_bookmark2(doc, loc)
39774: (0)          def ll_fz_lookup_bookmark2(doc, mark):
39775: (4)              return _mupdf.ll_fz_lookup_bookmark2(doc, mark)
39776: (0)          def fz_lookup_bookmark2(doc, mark):
39777: (4)              return _mupdf.fz_lookup_bookmark2(doc, mark)
39778: (0)          class fz_convert_color2_v(object):
39779: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39780: (4)              __repr__ = _swig_repr
39781: (4)              v0 = property(_mupdf.fz_convert_color2_v_v0_get,
_mupdf.fz_convert_color2_v_v0_set)
39782: (4)              v1 = property(_mupdf.fz_convert_color2_v_v1_get,
_mupdf.fz_convert_color2_v_v1_set)
39783: (4)              v2 = property(_mupdf.fz_convert_color2_v_v2_get,
_mupdf.fz_convert_color2_v_v2_set)
39784: (4)              v3 = property(_mupdf.fz_convert_color2_v_v3_get,
_mupdf.fz_convert_color2_v_v3_set)
39785: (4)              def __init__(self):
39786: (8)                  _mupdf.fz_convert_color2_v_swiginit(self,
_mupdf.new_fz_convert_color2_v())
39787: (4)              __swig_destroy__ = _mupdf.delete_fz_convert_color2_v
39788: (0)          _mupdf.fz_convert_color2_v_swigregister(fz_convert_color2_v)
39789: (0)          def ll_fz_convert_color2(*args):
39790: (4)              return _mupdf.ll_fz_convert_color2(*args)
39791: (0)          class DiagnosticCallback(object):
39792: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39793: (4)              __repr__ = _swig_repr
39794: (4)              def __init__(self, description):
39795: (8)                  if self.__class__ == DiagnosticCallback:
39796: (12)                     _self = None
39797: (8)                  else:
39798: (12)                     _self = self
39799: (8)                  _mupdf.DiagnosticCallback_swiginit(self,
_mupdf.new_DiagnosticCallback(_self, description))
39800: (4)              def _print(self, message):
39801: (8)                  return _mupdf.DiagnosticCallback__print(self, message)
39802: (4)              __swig_destroy__ = _mupdf.delete_DiagnosticCallback
39803: (4)              @staticmethod
39804: (4)              def s_print(self0, message):
39805: (8)                  return _mupdf.DiagnosticCallback_s_print(self0, message)
39806: (4)              m_description = property(_mupdf.DiagnosticCallback_m_description_get,
_mupdf.DiagnosticCallback_m_description_set)
39807: (4)              def __disown__(self):
39808: (8)                  self.this.disown()
39809: (8)                  _mupdf.disown_DiagnosticCallback(self)
39810: (8)                  return weakref.proxy(self)
39811: (0)          _mupdf.DiagnosticCallback_swigregister(DiagnosticCallback)
39812: (0)          class StoryPositionsCallback(object):
39813: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39814: (4)              __repr__ = _swig_repr
39815: (4)              def __init__(self):
39816: (8)                  if self.__class__ == StoryPositionsCallback:
39817: (12)                     _self = None
39818: (8)                  else:
39819: (12)                     _self = self
39820: (8)                  _mupdf.StoryPositionsCallback_swiginit(self,
_mupdf.new_StoryPositionsCallback(_self, ))
39821: (4)              def call(self, position):
39822: (8)                  return _mupdf.StoryPositionsCallback_call(self, position)
39823: (4)              @staticmethod
39824: (4)              def s_call(ctx, self0, position):
39825: (8)                  return _mupdf.StoryPositionsCallback_s_call(ctx, self0, position)
39826: (4)              __swig_destroy__ = _mupdf.delete_StoryPositionsCallback
39827: (4)              def __disown__(self):
39828: (8)                  self.this.disown()
```

```
39829: (8)                  _mupdf.disown_StoryPositionsCallback(self)
39830: (8)                  return weakref.proxy(self)
39831: (0)          _mupdf.StoryPositionsCallback_swigregister(StoryPositionsCallback)
39832: (0)          def ll_fz_story_positions_director(story, cb):
39833: (4)              return _mupdf.ll_fz_story_positions_director(story, cb)
39834: (0)          def Pixmap_set_alpha_helper(balen, n, data_len, zero_out, data, pix,
premultiply, bground, colors, bgcolor):
39835: (4)              return _mupdf.Pixmap_set_alpha_helper(balen, n, data_len, zero_out, data,
pix, premultiply, bground, colors, bgcolor)
39836: (0)          def page_merge_helper(old_annots, graft_map, doc_des, new_annots, n):
39837: (4)              return _mupdf.page_merge_helper(old_annots, graft_map, doc_des,
new_annots, n)
39838: (0)          class ll_fz_bidi_fragment_text_outparams(object):
39839: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39840: (4)              __repr__ = _swig_repr
39841: (4)              baseDir = property(_mupdf.ll_fz_bidi_fragment_text_outparams_baseDir_get,
_mupdf.ll_fz_bidi_fragment_text_outparams_baseDir_set)
39842: (4)              def __init__(self):
39843: (8)                  _mupdf.ll_fz_bidi_fragment_text_outparams_swiginit(self,
_mupdf.new_ll_fz_bidi_fragment_text_outparams())
39844: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_bidi_fragment_text_outparams
39845: (0)
_mupdf.ll_fz_bidi_fragment_text_outparams_swigregister(ll_fz_bidi_fragment_text_outparams)
39846: (0)          def ll_fz_bidi_fragment_text_outparams_fn(text, textlen, callback, arg, flags,
outparams):
39847: (4)              return _mupdf.ll_fz_bidi_fragment_text_outparams_fn(text, textlen,
callback, arg, flags, outparams)
39848: (0)          class ll_fz_bitmap_details_outparams(object):
39849: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39850: (4)              __repr__ = _swig_repr
39851: (4)              w = property(_mupdf.ll_fz_bitmap_details_outparams_w_get,
_mupdf.ll_fz_bitmap_details_outparams_w_set)
39852: (4)              h = property(_mupdf.ll_fz_bitmap_details_outparams_h_get,
_mupdf.ll_fz_bitmap_details_outparams_h_set)
39853: (4)              n = property(_mupdf.ll_fz_bitmap_details_outparams_n_get,
_mupdf.ll_fz_bitmap_details_outparams_n_set)
39854: (4)              stride = property(_mupdf.ll_fz_bitmap_details_outparams_stride_get,
_mupdf.ll_fz_bitmap_details_outparams_stride_set)
39855: (4)              def __init__(self):
39856: (8)                  _mupdf.ll_fz_bitmap_details_outparams_swiginit(self,
_mupdf.new_ll_fz_bitmap_details_outparams())
39857: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_bitmap_details_outparams
39858: (0)
_mupdf.ll_fz_bitmap_details_outparams_swigregister(ll_fz_bitmap_details_outparams)
39859: (0)          def ll_fz_bitmap_details_outparams_fn(bitmap, outparams):
39860: (4)              return _mupdf.ll_fz_bitmap_details_outparams_fn(bitmap, outparams)
39861: (0)          class ll_fz_buffer_extract_outparams(object):
39862: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39863: (4)              __repr__ = _swig_repr
39864: (4)              data = property(_mupdf.ll_fz_buffer_extract_outparams_data_get,
_mupdf.ll_fz_buffer_extract_outparams_data_set)
39865: (4)              def __init__(self):
39866: (8)                  _mupdf.ll_fz_buffer_extract_outparams_swiginit(self,
_mupdf.new_ll_fz_buffer_extract_outparams())
39867: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_buffer_extract_outparams
39868: (0)
_mupdf.ll_fz_buffer_extract_outparams_swigregister(ll_fz_buffer_extract_outparams)
39869: (0)          def ll_fz_buffer_extract_outparams_fn(buf, outparams):
39870: (4)              return _mupdf.ll_fz_buffer_extract_outparams_fn(buf, outparams)
39871: (0)          class ll_fz_buffer_storage_outparams(object):
39872: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39873: (4)              __repr__ = _swig_repr
39874: (4)              datap = property(_mupdf.ll_fz_buffer_storage_outparams_datap_get,
_mupdf.ll_fz_buffer_storage_outparams_datap_set)
39875: (4)              def __init__(self):
```

```
39876: (8)                  _mupdf.ll_fz_buffer_storage_outparams_swiginit(self,
_mupdf.new_ll_fz_buffer_storage_outparams())
39877: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_buffer_storage_outparams
39878: (0)
_mupdf.ll_fz_buffer_storage_outparams_swigregister(ll_fz_buffer_storage_outparams)
39879: (0)          def ll_fz_buffer_storage_outparams_fn(buf, outparams):
39880: (4)              return _mupdf.ll_fz_buffer_storage_outparams_fn(buf, outparams)
39881: (0)          class ll_fz_chartorune_outparams(object):
39882: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39883: (4)              __repr__ = _swig_repr
39884: (4)              rune = property(_mupdf.ll_fz_chartorune_outparams_rune_get,
_mupdf.ll_fz_chartorune_outparams_rune_set)
39885: (4)              def __init__(self):
39886: (8)                  _mupdf.ll_fz_chartorune_outparams_swiginit(self,
_mupdf.new_ll_fz_chartorune_outparams())
39887: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_chartorune_outparams
39888: (0)          _mupdf.ll_fz_chartorune_outparams_swigregister(ll_fz_chartorune_outparams)
39889: (0)          def ll_fz_chartorune_outparams_fn(str, outparams):
39890: (4)              return _mupdf.ll_fz_chartorune_outparams_fn(str, outparams)
39891: (0)          class ll_fz_clamp_color_outparams(object):
39892: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39893: (4)              __repr__ = _swig_repr
39894: (4)              out = property(_mupdf.ll_fz_clamp_color_outparams_out_get,
_mupdf.ll_fz_clamp_color_outparams_out_set)
39895: (4)              def __init__(self):
39896: (8)                  _mupdf.ll_fz_clamp_color_outparams_swiginit(self,
_mupdf.new_ll_fz_clamp_color_outparams())
39897: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_clamp_color_outparams
39898: (0)          _mupdf.ll_fz_clamp_color_outparams_swigregister(ll_fz_clamp_color_outparams)
39899: (0)          def ll_fz_clamp_color_outparams_fn(cs, _in, outparams):
39900: (4)              return _mupdf.ll_fz_clamp_color_outparams_fn(cs, _in, outparams)
39901: (0)          class ll_fz_convert_color_outparams(object):
39902: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39903: (4)              __repr__ = _swig_repr
39904: (4)              dv = property(_mupdf.ll_fz_convert_color_outparams_dv_get,
_mupdf.ll_fz_convert_color_outparams_dv_set)
39905: (4)              def __init__(self):
39906: (8)                  _mupdf.ll_fz_convert_color_outparams_swiginit(self,
_mupdf.new_ll_fz_convert_color_outparams())
39907: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_convert_color_outparams
39908: (0)
_mupdf.ll_fz_convert_color_outparams_swigregister(ll_fz_convert_color_outparams)
39909: (0)          def ll_fz_convert_color_outparams_fn(ss, sv, ds, _is, params, outparams):
39910: (4)              return _mupdf.ll_fz_convert_color_outparams_fn(ss, sv, ds, _is, params,
outparams)
39911: (0)          class ll_fz_convert_error_outparams(object):
39912: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39913: (4)              __repr__ = _swig_repr
39914: (4)              code = property(_mupdf.ll_fz_convert_error_outparams_code_get,
_mupdf.ll_fz_convert_error_outparams_code_set)
39915: (4)              def __init__(self):
39916: (8)                  _mupdf.ll_fz_convert_error_outparams_swiginit(self,
_mupdf.new_ll_fz_convert_error_outparams())
39917: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_convert_error_outparams
39918: (0)
_mupdf.ll_fz_convert_error_outparams_swigregister(ll_fz_convert_error_outparams)
39919: (0)          def ll_fz_convert_error_outparams_fn(outparams):
39920: (4)              return _mupdf.ll_fz_convert_error_outparams_fn(outparams)
39921: (0)          class ll_fz_convert_separation_colors_outparams(object):
39922: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39923: (4)              __repr__ = _swig_repr
39924: (4)              dst_color =
property(_mupdf.ll_fz_convert_separation_colors_outparams_dst_color_get,
_mupdf.ll_fz_convert_separation_colors_outparams_dst_color_set)
```

```
39925: (4)              def __init__(self):
39926: (8)                  _mupdf.ll_fz_convert_separation_colors_outparams_swiginit(self,
_mupdf.new_ll_fz_convert_separation_colors_outparams())
39927: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_convert_separation_colors_outparams
39928: (0)
_mupdf.ll_fz_convert_separation_colors_outparams_swigregister(ll_fz_convert_separation_colors_outp
arams)
39929: (0)          def ll_fz_convert_separation_colors_outparams_fn(src_cs, src_color, dst_seps,
dst_cs, color_params, outparams):
39930: (4)              return _mupdf.ll_fz_convert_separation_colors_outparams_fn(src_cs,
src_color, dst_seps, dst_cs, color_params, outparams)
39931: (0)          class ll_fz_decomp_image_from_stream_outparams(object):
39932: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39933: (4)              __repr__ = _swig_repr
39934: (4)              l2extra =
property(_mupdf.ll_fz_decomp_image_from_stream_outparams_l2extra_get,
_mupdf.ll_fz_decomp_image_from_stream_outparams_l2extra_set)
39935: (4)              def __init__(self):
39936: (8)                  _mupdf.ll_fz_decomp_image_from_stream_outparams_swiginit(self,
_mupdf.new_ll_fz_decomp_image_from_stream_outparams())
39937: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_decomp_image_from_stream_outparams
39938: (0)
_mupdf.ll_fz_decomp_image_from_stream_outparams_swigregister(ll_fz_decomp_image_from_stream_outpar
ams)
39939: (0)          def ll_fz_decomp_image_from_stream_outparams_fn(stm, image, subarea, indexed,
l2factor, outparams):
39940: (4)              return _mupdf.ll_fz_decomp_image_from_stream_outparams_fn(stm, image,
subarea, indexed, l2factor, outparams)
39941: (0)          class ll_fz_deflate_outparams(object):
39942: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39943: (4)              __repr__ = _swig_repr
39944: (4)              compressed_length =
property(_mupdf.ll_fz_deflate_outparams_compressed_length_get,
_mupdf.ll_fz_deflate_outparams_compressed_length_set)
39945: (4)              def __init__(self):
39946: (8)                  _mupdf.ll_fz_deflate_outparams_swiginit(self,
_mupdf.new_ll_fz_deflate_outparams())
39947: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_deflate_outparams
39948: (0)          _mupdf.ll_fz_deflate_outparams_swigregister(ll_fz_deflate_outparams)
39949: (0)          def ll_fz_deflate_outparams_fn(dest, source, source_length, level, outparams):
39950: (4)              return _mupdf.ll_fz_deflate_outparams_fn(dest, source, source_length,
level, outparams)
39951: (0)          class ll_fz_dom_get_attribute_outparams(object):
39952: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39953: (4)              __repr__ = _swig_repr
39954: (4)              att = property(_mupdf.ll_fz_dom_get_attribute_outparams_att_get,
_mupdf.ll_fz_dom_get_attribute_outparams_att_set)
39955: (4)              def __init__(self):
39956: (8)                  _mupdf.ll_fz_dom_get_attribute_outparams_swiginit(self,
_mupdf.new_ll_fz_dom_get_attribute_outparams())
39957: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_dom_get_attribute_outparams
39958: (0)
_mupdf.ll_fz_dom_get_attribute_outparams_swigregister(ll_fz_dom_get_attribute_outparams)
39959: (0)          def ll_fz_dom_get_attribute_outparams_fn(elt, i, outparams):
39960: (4)              return _mupdf.ll_fz_dom_get_attribute_outparams_fn(elt, i, outparams)
39961: (0)          class ll_fz_drop_imp_outparams(object):
39962: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39963: (4)              __repr__ = _swig_repr
39964: (4)              refs = property(_mupdf.ll_fz_drop_imp_outparams_refs_get,
_mupdf.ll_fz_drop_imp_outparams_refs_set)
39965: (4)              def __init__(self):
39966: (8)                  _mupdf.ll_fz_drop_imp_outparams_swiginit(self,
_mupdf.new_ll_fz_drop_imp_outparams())
39967: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_drop_imp_outparams
39968: (0)          _mupdf.ll_fz_drop_imp_outparams_swigregister(ll_fz_drop_imp_outparams)
```

```
39969: (0)          def ll_fz_drop_imp_outparams_fn(p, outparams):
39970: (4)              return _mupdf.ll_fz_drop_imp_outparams_fn(p, outparams)
39971: (0)          class ll_fz_drop_imp16_outparams(object):
39972: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39973: (4)              __repr__ = _swig_repr
39974: (4)              refs = property(_mupdf.ll_fz_drop_imp16_outparams_refs_get,
_mupdf.ll_fz_drop_imp16_outparams_refs_set)
39975: (4)              def __init__(self):
39976: (8)                  _mupdf.ll_fz_drop_imp16_outparams_swiginit(self,
_mupdf.new_ll_fz_drop_imp16_outparams())
39977: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_drop_imp16_outparams
39978: (0)          _mupdf.ll_fz_drop_imp16_outparams_swigregister(ll_fz_drop_imp16_outparams)
39979: (0)          def ll_fz_drop_imp16_outparams_fn(p, outparams):
39980: (4)              return _mupdf.ll_fz_drop_imp16_outparams_fn(p, outparams)
39981: (0)          class ll_fz_encode_character_with_fallback_outparams(object):
39982: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39983: (4)              __repr__ = _swig_repr
39984: (4)              out_font =
property(_mupdf.ll_fz_encode_character_with_fallback_outparams_out_font_get,
_mupdf.ll_fz_encode_character_with_fallback_outparams_out_font_set)
39985: (4)              def __init__(self):
39986: (8)                  _mupdf.ll_fz_encode_character_with_fallback_outparams_swiginit(self,
_mupdf.new_ll_fz_encode_character_with_fallback_outparams())
39987: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_encode_character_with_fallback_outparams
39988: (0)
_mupdf.ll_fz_encode_character_with_fallback_outparams_swigregister(ll_fz_encode_character_with_fal
lback_outparams)
39989: (0)          def ll_fz_encode_character_with_fallback_outparams_fn(font, unicode, script,
language, outparams):
39990: (4)              return _mupdf.ll_fz_encode_character_with_fallback_outparams_fn(font,
unicode, script, language, outparams)
39991: (0)          class ll_fz_error_callback_outparams(object):
39992: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
39993: (4)              __repr__ = _swig_repr
39994: (4)              user = property(_mupdf.ll_fz_error_callback_outparams_user_get,
_mupdf.ll_fz_error_callback_outparams_user_set)
39995: (4)              def __init__(self):
39996: (8)                  _mupdf.ll_fz_error_callback_outparams_swiginit(self,
_mupdf.new_ll_fz_error_callback_outparams())
39997: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_error_callback_outparams
39998: (0)
_mupdf.ll_fz_error_callback_outparams_swigregister(ll_fz_error_callback_outparams)
39999: (0)          def ll_fz_error_callback_outparams_fn(outparams):
40000: (4)              return _mupdf.ll_fz_error_callback_outparams_fn(outparams)
40001: (0)          class ll_fz_eval_function_outparams(object):
40002: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40003: (4)              __repr__ = _swig_repr
40004: (4)              out = property(_mupdf.ll_fz_eval_function_outparams_out_get,
_mupdf.ll_fz_eval_function_outparams_out_set)
40005: (4)              def __init__(self):
40006: (8)                  _mupdf.ll_fz_eval_function_outparams_swiginit(self,
_mupdf.new_ll_fz_eval_function_outparams())
40007: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_eval_function_outparams
40008: (0)
_mupdf.ll_fz_eval_function_outparams_swigregister(ll_fz_eval_function_outparams)
40009: (0)          def ll_fz_eval_function_outparams_fn(func, _in, inlen, outlen, outparams):
40010: (4)              return _mupdf.ll_fz_eval_function_outparams_fn(func, _in, inlen, outlen,
outparams)
40011: (0)          class ll_fz_fill_pixmap_with_color_outparams(object):
40012: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40013: (4)              __repr__ = _swig_repr
40014: (4)              color = property(_mupdf.ll_fz_fill_pixmap_with_color_outparams_color_get,
_mupdf.ll_fz_fill_pixmap_with_color_outparams_color_set)
```

```
40015: (4)              def __init__(self):
40016: (8)                  _mupdf.ll_fz_fill_pixmap_with_color_outparams_swiginit(self,
_mupdf.new_ll_fz_fill_pixmap_with_color_outparams())
40017: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_fill_pixmap_with_color_outparams
40018: (0)
_mupdf.ll_fz_fill_pixmap_with_color_outparams_swigregister(ll_fz_fill_pixmap_with_color_outparams)
40019: (0)          def ll_fz_fill_pixmap_with_color_outparams_fn(pix, colorspace, color_params,
outparams):
40020: (4)              return _mupdf.ll_fz_fill_pixmap_with_color_outparams_fn(pix, colorspace,
color_params, outparams)
40021: (0)          class ll_fz_get_pixmap_from_image_outparams(object):
40022: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40023: (4)              __repr__ = _swig_repr
40024: (4)              w = property(_mupdf.ll_fz_get_pixmap_from_image_outparams_w_get,
_mupdf.ll_fz_get_pixmap_from_image_outparams_w_set)
40025: (4)              h = property(_mupdf.ll_fz_get_pixmap_from_image_outparams_h_get,
_mupdf.ll_fz_get_pixmap_from_image_outparams_h_set)
40026: (4)              def __init__(self):
40027: (8)                  _mupdf.ll_fz_get_pixmap_from_image_outparams_swiginit(self,
_mupdf.new_ll_fz_get_pixmap_from_image_outparams())
40028: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_get_pixmap_from_image_outparams
40029: (0)
_mupdf.ll_fz_get_pixmap_from_image_outparams_swigregister(ll_fz_get_pixmap_from_image_outparams)
40030: (0)          def ll_fz_get_pixmap_from_image_outparams_fn(image, subarea, ctm, outparams):
40031: (4)              return _mupdf.ll_fz_get_pixmap_from_image_outparams_fn(image, subarea,
ctm, outparams)
40032: (0)          class ll_fz_getopt_outparams(object):
40033: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40034: (4)              __repr__ = _swig_repr
40035: (4)              nargv = property(_mupdf.ll_fz_getopt_outparams_nargv_get,
_mupdf.ll_fz_getopt_outparams_nargv_set)
40036: (4)              def __init__(self):
40037: (8)                  _mupdf.ll_fz_getopt_outparams_swiginit(self,
_mupdf.new_ll_fz_getopt_outparams())
40038: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_getopt_outparams
40039: (0)          _mupdf.ll_fz_getopt_outparams_swigregister(ll_fz_getopt_outparams)
40040: (0)          def ll_fz_getopt_outparams_fn(nargc, ostr, outparams):
40041: (4)              return _mupdf.ll_fz_getopt_outparams_fn(nargc, ostr, outparams)
40042: (0)          class ll_fz_getopt_long_outparams(object):
40043: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40044: (4)              __repr__ = _swig_repr
40045: (4)              nargv = property(_mupdf.ll_fz_getopt_long_outparams_nargv_get,
_mupdf.ll_fz_getopt_long_outparams_nargv_set)
40046: (4)              def __init__(self):
40047: (8)                  _mupdf.ll_fz_getopt_long_outparams_swiginit(self,
_mupdf.new_ll_fz_getopt_long_outparams())
40048: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_getopt_long_outparams
40049: (0)          _mupdf.ll_fz_getopt_long_outparams_swigregister(ll_fz_getopt_long_outparams)
40050: (0)          def ll_fz_getopt_long_outparams_fn(nargc, ostr, longopts, outparams):
40051: (4)              return _mupdf.ll_fz_getopt_long_outparams_fn(nargc, ostr, longopts,
outparams)
40052: (0)          class ll_fz_grisu_outparams(object):
40053: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40054: (4)              __repr__ = _swig_repr
40055: (4)              exp = property(_mupdf.ll_fz_grisu_outparams_exp_get,
_mupdf.ll_fz_grisu_outparams_exp_set)
40056: (4)              def __init__(self):
40057: (8)                  _mupdf.ll_fz_grisu_outparams_swiginit(self,
_mupdf.new_ll_fz_grisu_outparams())
40058: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_grisu_outparams
40059: (0)          _mupdf.ll_fz_grisu_outparams_swigregister(ll_fz_grisu_outparams)
40060: (0)          def ll_fz_grisu_outparams_fn(f, s, outparams):
40061: (4)              return _mupdf.ll_fz_grisu_outparams_fn(f, s, outparams)
40062: (0)          class ll_fz_has_option_outparams(object):
40063: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
40064: (4)              __repr__ = _swig_repr
40065: (4)              val = property(_mupdf.ll_fz_has_option_outparams_val_get,
_mupdf.ll_fz_has_option_outparams_val_set)
40066: (4)              def __init__(self):
40067: (8)                  _mupdf.ll_fz_has_option_outparams_swiginit(self,
_mupdf.new_ll_fz_has_option_outparams())
40068: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_has_option_outparams
40069: (0)          _mupdf.ll_fz_has_option_outparams_swigregister(ll_fz_has_option_outparams)
40070: (0)          def ll_fz_has_option_outparams_fn(opts, key, outparams):
40071: (4)              return _mupdf.ll_fz_has_option_outparams_fn(opts, key, outparams)
40072: (0)          class ll_fz_image_resolution_outparams(object):
40073: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40074: (4)              __repr__ = _swig_repr
40075: (4)              xres = property(_mupdf.ll_fz_image_resolution_outparams_xres_get,
_mupdf.ll_fz_image_resolution_outparams_xres_set)
40076: (4)              yres = property(_mupdf.ll_fz_image_resolution_outparams_yres_get,
_mupdf.ll_fz_image_resolution_outparams_yres_set)
40077: (4)              def __init__(self):
40078: (8)                  _mupdf.ll_fz_image_resolution_outparams_swiginit(self,
_mupdf.new_ll_fz_image_resolution_outparams())
40079: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_image_resolution_outparams
40080: (0)
_mupdf.ll_fz_image_resolution_outparams_swigregister(ll_fz_image_resolution_outparams)
40081: (0)          def ll_fz_image_resolution_outparams_fn(image, outparams):
40082: (4)              return _mupdf.ll_fz_image_resolution_outparams_fn(image, outparams)
40083: (0)          class ll_fz_keep_imp_outparams(object):
40084: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40085: (4)              __repr__ = _swig_repr
40086: (4)              refs = property(_mupdf.ll_fz_keep_imp_outparams_refs_get,
_mupdf.ll_fz_keep_imp_outparams_refs_set)
40087: (4)              def __init__(self):
40088: (8)                  _mupdf.ll_fz_keep_imp_outparams_swiginit(self,
_mupdf.new_ll_fz_keep_imp_outparams())
40089: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_keep_imp_outparams
40090: (0)          _mupdf.ll_fz_keep_imp_outparams_swigregister(ll_fz_keep_imp_outparams)
40091: (0)          def ll_fz_keep_imp_outparams_fn(p, outparams):
40092: (4)              return _mupdf.ll_fz_keep_imp_outparams_fn(p, outparams)
40093: (0)          class ll_fz_keep_imp16_outparams(object):
40094: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40095: (4)              __repr__ = _swig_repr
40096: (4)              refs = property(_mupdf.ll_fz_keep_imp16_outparams_refs_get,
_mupdf.ll_fz_keep_imp16_outparams_refs_set)
40097: (4)              def __init__(self):
40098: (8)                  _mupdf.ll_fz_keep_imp16_outparams_swiginit(self,
_mupdf.new_ll_fz_keep_imp16_outparams())
40099: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_keep_imp16_outparams
40100: (0)          _mupdf.ll_fz_keep_imp16_outparams_swigregister(ll_fz_keep_imp16_outparams)
40101: (0)          def ll_fz_keep_imp16_outparams_fn(p, outparams):
40102: (4)              return _mupdf.ll_fz_keep_imp16_outparams_fn(p, outparams)
40103: (0)          class ll_fz_keep_imp_locked_outparams(object):
40104: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40105: (4)              __repr__ = _swig_repr
40106: (4)              refs = property(_mupdf.ll_fz_keep_imp_locked_outparams_refs_get,
_mupdf.ll_fz_keep_imp_locked_outparams_refs_set)
40107: (4)              def __init__(self):
40108: (8)                  _mupdf.ll_fz_keep_imp_locked_outparams_swiginit(self,
_mupdf.new_ll_fz_keep_imp_locked_outparams())
40109: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_keep_imp_locked_outparams
40110: (0)
_mupdf.ll_fz_keep_imp_locked_outparams_swigregister(ll_fz_keep_imp_locked_outparams)
40111: (0)          def ll_fz_keep_imp_locked_outparams_fn(p, outparams):
40112: (4)              return _mupdf.ll_fz_keep_imp_locked_outparams_fn(p, outparams)
40113: (0)          class ll_fz_lookup_base14_font_outparams(object):
40114: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
```

```
doc="The membership flag")
40115: (4)              __repr__ = _swig_repr
40116: (4)              len = property(_mupdf.ll_fz_lookup_base14_font_outparams_len_get,
_mupdf.ll_fz_lookup_base14_font_outparams_len_set)
40117: (4)              def __init__(self):
40118: (8)                  _mupdf.ll_fz_lookup_base14_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_base14_font_outparams())
40119: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_base14_font_outparams
40120: (0)
_mupdf.ll_fz_lookup_base14_font_outparams_swigregister(ll_fz_lookup_base14_font_outparams)
40121: (0)          def ll_fz_lookup_base14_font_outparams_fn(name, outparams):
40122: (4)              return _mupdf.ll_fz_lookup_base14_font_outparams_fn(name, outparams)
40123: (0)          class ll_fz_lookup_builtin_font_outparams(object):
40124: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40125: (4)              __repr__ = _swig_repr
40126: (4)              len = property(_mupdf.ll_fz_lookup_builtin_font_outparams_len_get,
_mupdf.ll_fz_lookup_builtin_font_outparams_len_set)
40127: (4)              def __init__(self):
40128: (8)                  _mupdf.ll_fz_lookup_builtin_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_builtin_font_outparams())
40129: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_builtin_font_outparams
40130: (0)
_mupdf.ll_fz_lookup_builtin_font_outparams_swigregister(ll_fz_lookup_builtin_font_outparams)
40131: (0)          def ll_fz_lookup_builtin_font_outparams_fn(name, bold, italic, outparams):
40132: (4)              return _mupdf.ll_fz_lookup_builtin_font_outparams_fn(name, bold, italic,
outparams)
40133: (0)          class ll_fz_lookup_cjk_font_outparams(object):
40134: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40135: (4)              __repr__ = _swig_repr
40136: (4)              len = property(_mupdf.ll_fz_lookup_cjk_font_outparams_len_get,
_mupdf.ll_fz_lookup_cjk_font_outparams_len_set)
40137: (4)              index = property(_mupdf.ll_fz_lookup_cjk_font_outparams_index_get,
_mupdf.ll_fz_lookup_cjk_font_outparams_index_set)
40138: (4)              def __init__(self):
40139: (8)                  _mupdf.ll_fz_lookup_cjk_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_cjk_font_outparams())
40140: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_cjk_font_outparams
40141: (0)
_mupdf.ll_fz_lookup_cjk_font_outparams_swigregister(ll_fz_lookup_cjk_font_outparams)
40142: (0)          def ll_fz_lookup_cjk_font_outparams_fn(ordering, outparams):
40143: (4)              return _mupdf.ll_fz_lookup_cjk_font_outparams_fn(ordering, outparams)
40144: (0)          class ll_fz_lookup_cjk_font_by_language_outparams(object):
40145: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40146: (4)              __repr__ = _swig_repr
40147: (4)              len = property(_mupdf.ll_fz_lookup_cjk_font_by_language_outparams_len_get,
_mupdf.ll_fz_lookup_cjk_font_by_language_outparams_len_set)
40148: (4)              subfont =
property(_mupdf.ll_fz_lookup_cjk_font_by_language_outparams_subfont_get,
_mupdf.ll_fz_lookup_cjk_font_by_language_outparams_subfont_set)
40149: (4)              def __init__(self):
40150: (8)                  _mupdf.ll_fz_lookup_cjk_font_by_language_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_cjk_font_by_language_outparams())
40151: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_lookup_cjk_font_by_language_outparams
40152: (0)
_mupdf.ll_fz_lookup_cjk_font_by_language_outparams_swigregister(ll_fz_lookup_cjk_font_by_language_
outparams)
40153: (0)          def ll_fz_lookup_cjk_font_by_language_outparams_fn(lang, outparams):
40154: (4)              return _mupdf.ll_fz_lookup_cjk_font_by_language_outparams_fn(lang,
outparams)
40155: (0)          class ll_fz_lookup_noto_boxes_font_outparams(object):
40156: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40157: (4)              __repr__ = _swig_repr
40158: (4)              len = property(_mupdf.ll_fz_lookup_noto_boxes_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_boxes_font_outparams_len_set)
```

```
40159: (4)              def __init__(self):
40160: (8)                  _mupdf.ll_fz_lookup_noto_boxes_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_boxes_font_outparams())
40161: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_boxes_font_outparams
40162: (0)
_mupdf.ll_fz_lookup_noto_boxes_font_outparams_swigregister(ll_fz_lookup_noto_boxes_font_outparams)
40163: (0)          def ll_fz_lookup_noto_boxes_font_outparams_fn(outparams):
40164: (4)              return _mupdf.ll_fz_lookup_noto_boxes_font_outparams_fn(outparams)
40165: (0)          class ll_fz_lookup_noto_emoji_font_outparams(object):
40166: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40167: (4)              __repr__ = _swig_repr
40168: (4)              len = property(_mupdf.ll_fz_lookup_noto_emoji_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_emoji_font_outparams_len_set)
40169: (4)              def __init__(self):
40170: (8)                  _mupdf.ll_fz_lookup_noto_emoji_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_emoji_font_outparams())
40171: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_emoji_font_outparams
40172: (0)
_mupdf.ll_fz_lookup_noto_emoji_font_outparams_swigregister(ll_fz_lookup_noto_emoji_font_outparams)
40173: (0)          def ll_fz_lookup_noto_emoji_font_outparams_fn(outparams):
40174: (4)              return _mupdf.ll_fz_lookup_noto_emoji_font_outparams_fn(outparams)
40175: (0)          class ll_fz_lookup_noto_font_outparams(object):
40176: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40177: (4)              __repr__ = _swig_repr
40178: (4)              len = property(_mupdf.ll_fz_lookup_noto_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_font_outparams_len_set)
40179: (4)              subfont = property(_mupdf.ll_fz_lookup_noto_font_outparams_subfont_get,
_mupdf.ll_fz_lookup_noto_font_outparams_subfont_set)
40180: (4)              def __init__(self):
40181: (8)                  _mupdf.ll_fz_lookup_noto_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_font_outparams())
40182: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_font_outparams
40183: (0)
_mupdf.ll_fz_lookup_noto_font_outparams_swigregister(ll_fz_lookup_noto_font_outparams)
40184: (0)          def ll_fz_lookup_noto_font_outparams_fn(script, lang, outparams):
40185: (4)              return _mupdf.ll_fz_lookup_noto_font_outparams_fn(script, lang, outparams)
40186: (0)          class ll_fz_lookup_noto_math_font_outparams(object):
40187: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40188: (4)              __repr__ = _swig_repr
40189: (4)              len = property(_mupdf.ll_fz_lookup_noto_math_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_math_font_outparams_len_set)
40190: (4)              def __init__(self):
40191: (8)                  _mupdf.ll_fz_lookup_noto_math_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_math_font_outparams())
40192: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_math_font_outparams
40193: (0)
_mupdf.ll_fz_lookup_noto_math_font_outparams_swigregister(ll_fz_lookup_noto_math_font_outparams)
40194: (0)          def ll_fz_lookup_noto_math_font_outparams_fn(outparams):
40195: (4)              return _mupdf.ll_fz_lookup_noto_math_font_outparams_fn(outparams)
40196: (0)          class ll_fz_lookup_noto_music_font_outparams(object):
40197: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40198: (4)              __repr__ = _swig_repr
40199: (4)              len = property(_mupdf.ll_fz_lookup_noto_music_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_music_font_outparams_len_set)
40200: (4)              def __init__(self):
40201: (8)                  _mupdf.ll_fz_lookup_noto_music_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_music_font_outparams())
40202: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_music_font_outparams
40203: (0)
_mupdf.ll_fz_lookup_noto_music_font_outparams_swigregister(ll_fz_lookup_noto_music_font_outparams)
40204: (0)          def ll_fz_lookup_noto_music_font_outparams_fn(outparams):
40205: (4)              return _mupdf.ll_fz_lookup_noto_music_font_outparams_fn(outparams)
40206: (0)          class ll_fz_lookup_noto_symbol1_font_outparams(object):
40207: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
40208: (4)              __repr__ = _swig_repr
40209: (4)              len = property(_mupdf.ll_fz_lookup_noto_symbol1_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_symbol1_font_outparams_len_set)
40210: (4)              def __init__(self):
40211: (8)                  _mupdf.ll_fz_lookup_noto_symbol1_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_symbol1_font_outparams())
40212: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_symbol1_font_outparams
40213: (0)
_mupdf.ll_fz_lookup_noto_symbol1_font_outparams_swigregister(ll_fz_lookup_noto_symbol1_font_outpar
ams)
40214: (0)          def ll_fz_lookup_noto_symbol1_font_outparams_fn(outparams):
40215: (4)              return _mupdf.ll_fz_lookup_noto_symbol1_font_outparams_fn(outparams)
40216: (0)          class ll_fz_lookup_noto_symbol2_font_outparams(object):
40217: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40218: (4)              __repr__ = _swig_repr
40219: (4)              len = property(_mupdf.ll_fz_lookup_noto_symbol2_font_outparams_len_get,
_mupdf.ll_fz_lookup_noto_symbol2_font_outparams_len_set)
40220: (4)              def __init__(self):
40221: (8)                  _mupdf.ll_fz_lookup_noto_symbol2_font_outparams_swiginit(self,
_mupdf.new_ll_fz_lookup_noto_symbol2_font_outparams())
40222: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_lookup_noto_symbol2_font_outparams
40223: (0)
_mupdf.ll_fz_lookup_noto_symbol2_font_outparams_swigregister(ll_fz_lookup_noto_symbol2_font_outpar
ams)
40224: (0)          def ll_fz_lookup_noto_symbol2_font_outparams_fn(outparams):
40225: (4)              return _mupdf.ll_fz_lookup_noto_symbol2_font_outparams_fn(outparams)
40226: (0)          class ll_fz_new_deflated_data_outparams(object):
40227: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40228: (4)              __repr__ = _swig_repr
40229: (4)              compressed_length =
property(_mupdf.ll_fz_new_deflated_data_outparams_compressed_length_get,
_mupdf.ll_fz_new_deflated_data_outparams_compressed_length_set)
40230: (4)              def __init__(self):
40231: (8)                  _mupdf.ll_fz_new_deflated_data_outparams_swiginit(self,
_mupdf.new_ll_fz_new_deflated_data_outparams())
40232: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_new_deflated_data_outparams
40233: (0)
_mupdf.ll_fz_new_deflated_data_outparams_swigregister(ll_fz_new_deflated_data_outparams)
40234: (0)          def ll_fz_new_deflated_data_outparams_fn(source, source_length, level,
outparams):
40235: (4)              return _mupdf.ll_fz_new_deflated_data_outparams_fn(source, source_length,
level, outparams)
40236: (0)          class ll_fz_new_deflated_data_from_buffer_outparams(object):
40237: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40238: (4)              __repr__ = _swig_repr
40239: (4)              compressed_length =
property(_mupdf.ll_fz_new_deflated_data_from_buffer_outparams_compressed_length_get,
_mupdf.ll_fz_new_deflated_data_from_buffer_outparams_compressed_length_set)
40240: (4)              def __init__(self):
40241: (8)                  _mupdf.ll_fz_new_deflated_data_from_buffer_outparams_swiginit(self,
_mupdf.new_ll_fz_new_deflated_data_from_buffer_outparams())
40242: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_new_deflated_data_from_buffer_outparams
40243: (0)
_mupdf.ll_fz_new_deflated_data_from_buffer_outparams_swigregister(ll_fz_new_deflated_data_from_buf
fer_outparams)
40244: (0)          def ll_fz_new_deflated_data_from_buffer_outparams_fn(buffer, level,
outparams):
40245: (4)              return _mupdf.ll_fz_new_deflated_data_from_buffer_outparams_fn(buffer,
level, outparams)
40246: (0)          class ll_fz_new_display_list_from_svg_outparams(object):
40247: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40248: (4)              __repr__ = _swig_repr
40249: (4)              w = property(_mupdf.ll_fz_new_display_list_from_svg_outparams_w_get,
_mupdf.ll_fz_new_display_list_from_svg_outparams_w_set)
```

```
40250: (4)              h = property(_mupdf.ll_fz_new_display_list_from_svg_outparams_h_get,
_mupdf.ll_fz_new_display_list_from_svg_outparams_h_set)
40251: (4)              def __init__(self):
40252: (8)                  _mupdf.ll_fz_new_display_list_from_svg_outparams_swiginit(self,
_mupdf.new_ll_fz_new_display_list_from_svg_outparams())
40253: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_new_display_list_from_svg_outparams
40254: (0)
_mupdf.ll_fz_new_display_list_from_svg_outparams_swigregister(ll_fz_new_display_list_from_svg_outp
arams)
40255: (0)          def ll_fz_new_display_list_from_svg_outparams_fn(buf, base_uri, dir,
outparams):
40256: (4)              return _mupdf.ll_fz_new_display_list_from_svg_outparams_fn(buf, base_uri,
dir, outparams)
40257: (0)          class ll_fz_new_display_list_from_svg_xml_outparams(object):
40258: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40259: (4)              __repr__ = _swig_repr
40260: (4)              w = property(_mupdf.ll_fz_new_display_list_from_svg_xml_outparams_w_get,
_mupdf.ll_fz_new_display_list_from_svg_xml_outparams_w_set)
40261: (4)              h = property(_mupdf.ll_fz_new_display_list_from_svg_xml_outparams_h_get,
_mupdf.ll_fz_new_display_list_from_svg_xml_outparams_h_set)
40262: (4)              def __init__(self):
40263: (8)                  _mupdf.ll_fz_new_display_list_from_svg_xml_outparams_swiginit(self,
_mupdf.new_ll_fz_new_display_list_from_svg_xml_outparams())
40264: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_new_display_list_from_svg_xml_outparams
40265: (0)
_mupdf.ll_fz_new_display_list_from_svg_xml_outparams_swigregister(ll_fz_new_display_list_from_svg_
xml_outparams)
40266: (0)          def ll_fz_new_display_list_from_svg_xml_outparams_fn(xmldoc, xml, base_uri,
dir, outparams):
40267: (4)              return _mupdf.ll_fz_new_display_list_from_svg_xml_outparams_fn(xmldoc,
xml, base_uri, dir, outparams)
40268: (0)          class ll_fz_new_draw_device_with_options_outparams(object):
40269: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40270: (4)              __repr__ = _swig_repr
40271: (4)              pixmap =
property(_mupdf.ll_fz_new_draw_device_with_options_outparams_pixmap_get,
_mupdf.ll_fz_new_draw_device_with_options_outparams_pixmap_set)
40272: (4)              def __init__(self):
40273: (8)                  _mupdf.ll_fz_new_draw_device_with_options_outparams_swiginit(self,
_mupdf.new_ll_fz_new_draw_device_with_options_outparams())
40274: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_new_draw_device_with_options_outparams
40275: (0)
_mupdf.ll_fz_new_draw_device_with_options_outparams_swigregister(ll_fz_new_draw_device_with_option
s_outparams)
40276: (0)          def ll_fz_new_draw_device_with_options_outparams_fn(options, mediabox,
outparams):
40277: (4)              return _mupdf.ll_fz_new_draw_device_with_options_outparams_fn(options,
mediabox, outparams)
40278: (0)          class ll_fz_new_svg_device_with_id_outparams(object):
40279: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40280: (4)              __repr__ = _swig_repr
40281: (4)              id = property(_mupdf.ll_fz_new_svg_device_with_id_outparams_id_get,
_mupdf.ll_fz_new_svg_device_with_id_outparams_id_set)
40282: (4)              def __init__(self):
40283: (8)                  _mupdf.ll_fz_new_svg_device_with_id_outparams_swiginit(self,
_mupdf.new_ll_fz_new_svg_device_with_id_outparams())
40284: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_new_svg_device_with_id_outparams
40285: (0)
_mupdf.ll_fz_new_svg_device_with_id_outparams_swigregister(ll_fz_new_svg_device_with_id_outparams)
40286: (0)          def ll_fz_new_svg_device_with_id_outparams_fn(out, page_width, page_height,
text_format, reuse_images, outparams):
40287: (4)              return _mupdf.ll_fz_new_svg_device_with_id_outparams_fn(out, page_width,
page_height, text_format, reuse_images, outparams)
40288: (0)          class ll_fz_new_test_device_outparams(object):
```

```
40289: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40290: (4)              __repr__ = _swig_repr
40291: (4)              is_color = property(_mupdf.ll_fz_new_test_device_outparams_is_color_get,
_mupdf.ll_fz_new_test_device_outparams_is_color_set)
40292: (4)              def __init__(self):
40293: (8)                  _mupdf.ll_fz_new_test_device_outparams_swiginit(self,
_mupdf.new_ll_fz_new_test_device_outparams())
40294: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_new_test_device_outparams
40295: (0)
_mupdf.ll_fz_new_test_device_outparams_swigregister(ll_fz_new_test_device_outparams)
40296: (0)          def ll_fz_new_test_device_outparams_fn(threshold, options, passthrough,
outparams):
40297: (4)              return _mupdf.ll_fz_new_test_device_outparams_fn(threshold, options,
passthrough, outparams)
40298: (0)          class ll_fz_open_image_decomp_stream_outparams(object):
40299: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40300: (4)              __repr__ = _swig_repr
40301: (4)              l2factor =
property(_mupdf.ll_fz_open_image_decomp_stream_outparams_l2factor_get,
_mupdf.ll_fz_open_image_decomp_stream_outparams_l2factor_set)
40302: (4)              def __init__(self):
40303: (8)                  _mupdf.ll_fz_open_image_decomp_stream_outparams_swiginit(self,
_mupdf.new_ll_fz_open_image_decomp_stream_outparams())
40304: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_open_image_decomp_stream_outparams
40305: (0)
_mupdf.ll_fz_open_image_decomp_stream_outparams_swigregister(ll_fz_open_image_decomp_stream_outpar
ams)
40306: (0)          def ll_fz_open_image_decomp_stream_outparams_fn(arg_0, arg_1, outparams):
40307: (4)              return _mupdf.ll_fz_open_image_decomp_stream_outparams_fn(arg_0, arg_1,
outparams)
40308: (0)          class ll_fz_open_image_decomp_stream_from_buffer_outparams(object):
40309: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40310: (4)              __repr__ = _swig_repr
40311: (4)              l2factor =
property(_mupdf.ll_fz_open_image_decomp_stream_from_buffer_outparams_l2factor_get,
_mupdf.ll_fz_open_image_decomp_stream_from_buffer_outparams_l2factor_set)
40312: (4)              def __init__(self):
40313: (8)
_mupdf.ll_fz_open_image_decomp_stream_from_buffer_outparams_swiginit(self,
_mupdf.new_ll_fz_open_image_decomp_stream_from_buffer_outparams())
40314: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_open_image_decomp_stream_from_buffer_outparams
40315: (0)
_mupdf.ll_fz_open_image_decomp_stream_from_buffer_outparams_swigregister(ll_fz_open_image_decomp_s
tream_from_buffer_outparams)
40316: (0)          def ll_fz_open_image_decomp_stream_from_buffer_outparams_fn(arg_0, outparams):
40317: (4)              return
_mupdf.ll_fz_open_image_decomp_stream_from_buffer_outparams_fn(arg_0, outparams)
40318: (0)          class ll_fz_page_presentation_outparams(object):
40319: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40320: (4)              __repr__ = _swig_repr
40321: (4)              duration = property(_mupdf.ll_fz_page_presentation_outparams_duration_get,
_mupdf.ll_fz_page_presentation_outparams_duration_set)
40322: (4)              def __init__(self):
40323: (8)                  _mupdf.ll_fz_page_presentation_outparams_swiginit(self,
_mupdf.new_ll_fz_page_presentation_outparams())
40324: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_page_presentation_outparams
40325: (0)
_mupdf.ll_fz_page_presentation_outparams_swigregister(ll_fz_page_presentation_outparams)
40326: (0)          def ll_fz_page_presentation_outparams_fn(page, transition, outparams):
40327: (4)              return _mupdf.ll_fz_page_presentation_outparams_fn(page, transition,
outparams)
40328: (0)          class ll_fz_paint_shade_outparams(object):
40329: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
```

```
40330: (4)              __repr__ = _swig_repr
40331: (4)              cache = property(_mupdf.ll_fz_paint_shade_outparams_cache_get,
_mupdf.ll_fz_paint_shade_outparams_cache_set)
40332: (4)              def __init__(self):
40333: (8)                  _mupdf.ll_fz_paint_shade_outparams_swiginit(self,
_mupdf.new_ll_fz_paint_shade_outparams())
40334: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_paint_shade_outparams
40335: (0)          _mupdf.ll_fz_paint_shade_outparams_swigregister(ll_fz_paint_shade_outparams)
40336: (0)          def ll_fz_paint_shade_outparams_fn(shade, override_cs, ctm, dest,
color_params, bbox, eop, outparams):
40337: (4)              return _mupdf.ll_fz_paint_shade_outparams_fn(shade, override_cs, ctm,
dest, color_params, bbox, eop, outparams)
40338: (0)          class ll_fz_parse_page_range_outparams(object):
40339: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40340: (4)              __repr__ = _swig_repr
40341: (4)              a = property(_mupdf.ll_fz_parse_page_range_outparams_a_get,
_mupdf.ll_fz_parse_page_range_outparams_a_set)
40342: (4)              b = property(_mupdf.ll_fz_parse_page_range_outparams_b_get,
_mupdf.ll_fz_parse_page_range_outparams_b_set)
40343: (4)              def __init__(self):
40344: (8)                  _mupdf.ll_fz_parse_page_range_outparams_swiginit(self,
_mupdf.new_ll_fz_parse_page_range_outparams())
40345: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_parse_page_range_outparams
40346: (0)
_mupdf.ll_fz_parse_page_range_outparams_swigregister(ll_fz_parse_page_range_outparams)
40347: (0)          def ll_fz_parse_page_range_outparams_fn(s, n, outparams):
40348: (4)              return _mupdf.ll_fz_parse_page_range_outparams_fn(s, n, outparams)
40349: (0)          class ll_fz_read_best_outparams(object):
40350: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40351: (4)              __repr__ = _swig_repr
40352: (4)              truncated = property(_mupdf.ll_fz_read_best_outparams_truncated_get,
_mupdf.ll_fz_read_best_outparams_truncated_set)
40353: (4)              def __init__(self):
40354: (8)                  _mupdf.ll_fz_read_best_outparams_swiginit(self,
_mupdf.new_ll_fz_read_best_outparams())
40355: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_read_best_outparams
40356: (0)          _mupdf.ll_fz_read_best_outparams_swigregister(ll_fz_read_best_outparams)
40357: (0)          def ll_fz_read_best_outparams_fn(stm, initial, worst_case, outparams):
40358: (4)              return _mupdf.ll_fz_read_best_outparams_fn(stm, initial, worst_case,
outparams)
40359: (0)          class ll_fz_resolve_link_outparams(object):
40360: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40361: (4)              __repr__ = _swig_repr
40362: (4)              xp = property(_mupdf.ll_fz_resolve_link_outparams_xp_get,
_mupdf.ll_fz_resolve_link_outparams_xp_set)
40363: (4)              yp = property(_mupdf.ll_fz_resolve_link_outparams_yp_get,
_mupdf.ll_fz_resolve_link_outparams_yp_set)
40364: (4)              def __init__(self):
40365: (8)                  _mupdf.ll_fz_resolve_link_outparams_swiginit(self,
_mupdf.new_ll_fz_resolve_link_outparams())
40366: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_resolve_link_outparams
40367: (0)          _mupdf.ll_fz_resolve_link_outparams_swigregister(ll_fz_resolve_link_outparams)
40368: (0)          def ll_fz_resolve_link_outparams_fn(doc, uri, outparams):
40369: (4)              return _mupdf.ll_fz_resolve_link_outparams_fn(doc, uri, outparams)
40370: (0)          class ll_fz_search_chapter_page_number_outparams(object):
40371: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40372: (4)              __repr__ = _swig_repr
40373: (4)              hit_mark =
property(_mupdf.ll_fz_search_chapter_page_number_outparams_hit_mark_get,
_mupdf.ll_fz_search_chapter_page_number_outparams_hit_mark_set)
40374: (4)              def __init__(self):
40375: (8)                  _mupdf.ll_fz_search_chapter_page_number_outparams_swiginit(self,
_mupdf.new_ll_fz_search_chapter_page_number_outparams())
40376: (4)              __swig_destroy__ =
_mupdf.delete_ll_fz_search_chapter_page_number_outparams
```

```
40377: (0)
_mupdf.ll_fz_search_chapter_page_number_outparams_swigregister(ll_fz_search_chapter_page_number_ou
tparams)
40378: (0)          def ll_fz_search_chapter_page_number_outparams_fn(doc, chapter, page, needle,
hit_bbox, hit_max, outparams):
40379: (4)              return _mupdf.ll_fz_search_chapter_page_number_outparams_fn(doc, chapter,
page, needle, hit_bbox, hit_max, outparams)
40380: (0)          class ll_fz_search_display_list_outparams(object):
40381: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40382: (4)              __repr__ = _swig_repr
40383: (4)              hit_mark =
property(_mupdf.ll_fz_search_display_list_outparams_hit_mark_get,
_mupdf.ll_fz_search_display_list_outparams_hit_mark_set)
40384: (4)              def __init__(self):
40385: (8)                  _mupdf.ll_fz_search_display_list_outparams_swiginit(self,
_mupdf.new_ll_fz_search_display_list_outparams())
40386: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_search_display_list_outparams
40387: (0)
_mupdf.ll_fz_search_display_list_outparams_swigregister(ll_fz_search_display_list_outparams)
40388: (0)          def ll_fz_search_display_list_outparams_fn(list, needle, hit_bbox, hit_max,
outparams):
40389: (4)              return _mupdf.ll_fz_search_display_list_outparams_fn(list, needle,
hit_bbox, hit_max, outparams)
40390: (0)          class ll_fz_search_page_outparams(object):
40391: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40392: (4)              __repr__ = _swig_repr
40393: (4)              hit_mark = property(_mupdf.ll_fz_search_page_outparams_hit_mark_get,
_mupdf.ll_fz_search_page_outparams_hit_mark_set)
40394: (4)              def __init__(self):
40395: (8)                  _mupdf.ll_fz_search_page_outparams_swiginit(self,
_mupdf.new_ll_fz_search_page_outparams())
40396: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_search_page_outparams
40397: (0)          _mupdf.ll_fz_search_page_outparams_swigregister(ll_fz_search_page_outparams)
40398: (0)          def ll_fz_search_page_outparams_fn(page, needle, hit_bbox, hit_max,
outparams):
40399: (4)              return _mupdf.ll_fz_search_page_outparams_fn(page, needle, hit_bbox,
hit_max, outparams)
40400: (0)          class ll_fz_search_page_number_outparams(object):
40401: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40402: (4)              __repr__ = _swig_repr
40403: (4)              hit_mark =
property(_mupdf.ll_fz_search_page_number_outparams_hit_mark_get,
_mupdf.ll_fz_search_page_number_outparams_hit_mark_set)
40404: (4)              def __init__(self):
40405: (8)                  _mupdf.ll_fz_search_page_number_outparams_swiginit(self,
_mupdf.new_ll_fz_search_page_number_outparams())
40406: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_search_page_number_outparams
40407: (0)
_mupdf.ll_fz_search_page_number_outparams_swigregister(ll_fz_search_page_number_outparams)
40408: (0)          def ll_fz_search_page_number_outparams_fn(doc, number, needle, hit_bbox,
hit_max, outparams):
40409: (4)              return _mupdf.ll_fz_search_page_number_outparams_fn(doc, number, needle,
hit_bbox, hit_max, outparams)
40410: (0)          class ll_fz_search_stext_page_outparams(object):
40411: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40412: (4)              __repr__ = _swig_repr
40413: (4)              hit_mark = property(_mupdf.ll_fz_search_stext_page_outparams_hit_mark_get,
_mupdf.ll_fz_search_stext_page_outparams_hit_mark_set)
40414: (4)              def __init__(self):
40415: (8)                  _mupdf.ll_fz_search_stext_page_outparams_swiginit(self,
_mupdf.new_ll_fz_search_stext_page_outparams())
40416: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_search_stext_page_outparams
40417: (0)
_mupdf.ll_fz_search_stext_page_outparams_swigregister(ll_fz_search_stext_page_outparams)
40418: (0)          def ll_fz_search_stext_page_outparams_fn(text, needle, hit_bbox, hit_max,
```

```
outparams):
40419: (4)              return _mupdf.ll_fz_search_stext_page_outparams_fn(text, needle, hit_bbox,
hit_max, outparams)
40420: (0)          class ll_fz_separation_equivalent_outparams(object):
40421: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40422: (4)              __repr__ = _swig_repr
40423: (4)              dst_color =
property(_mupdf.ll_fz_separation_equivalent_outparams_dst_color_get,
_mupdf.ll_fz_separation_equivalent_outparams_dst_color_set)
40424: (4)              def __init__(self):
40425: (8)                  _mupdf.ll_fz_separation_equivalent_outparams_swiginit(self,
_mupdf.new_ll_fz_separation_equivalent_outparams())
40426: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_separation_equivalent_outparams
40427: (0)
_mupdf.ll_fz_separation_equivalent_outparams_swigregister(ll_fz_separation_equivalent_outparams)
40428: (0)          def ll_fz_separation_equivalent_outparams_fn(seps, idx, dst_cs, prf,
color_params, outparams):
40429: (4)              return _mupdf.ll_fz_separation_equivalent_outparams_fn(seps, idx, dst_cs,
prf, color_params, outparams)
40430: (0)          class ll_fz_store_scavenge_outparams(object):
40431: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40432: (4)              __repr__ = _swig_repr
40433: (4)              phase = property(_mupdf.ll_fz_store_scavenge_outparams_phase_get,
_mupdf.ll_fz_store_scavenge_outparams_phase_set)
40434: (4)              def __init__(self):
40435: (8)                  _mupdf.ll_fz_store_scavenge_outparams_swiginit(self,
_mupdf.new_ll_fz_store_scavenge_outparams())
40436: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_store_scavenge_outparams
40437: (0)
_mupdf.ll_fz_store_scavenge_outparams_swigregister(ll_fz_store_scavenge_outparams)
40438: (0)          def ll_fz_store_scavenge_outparams_fn(size, outparams):
40439: (4)              return _mupdf.ll_fz_store_scavenge_outparams_fn(size, outparams)
40440: (0)          class ll_fz_store_scavenge_external_outparams(object):
40441: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40442: (4)              __repr__ = _swig_repr
40443: (4)              phase = property(_mupdf.ll_fz_store_scavenge_external_outparams_phase_get,
_mupdf.ll_fz_store_scavenge_external_outparams_phase_set)
40444: (4)              def __init__(self):
40445: (8)                  _mupdf.ll_fz_store_scavenge_external_outparams_swiginit(self,
_mupdf.new_ll_fz_store_scavenge_external_outparams())
40446: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_store_scavenge_external_outparams
40447: (0)
_mupdf.ll_fz_store_scavenge_external_outparams_swigregister(ll_fz_store_scavenge_external_outparam
s)
40448: (0)          def ll_fz_store_scavenge_external_outparams_fn(size, outparams):
40449: (4)              return _mupdf.ll_fz_store_scavenge_external_outparams_fn(size, outparams)
40450: (0)          class ll_fz_strsep_outparams(object):
40451: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40452: (4)              __repr__ = _swig_repr
40453: (4)              stringp = property(_mupdf.ll_fz_strsep_outparams_stringp_get,
_mupdf.ll_fz_strsep_outparams_stringp_set)
40454: (4)              def __init__(self):
40455: (8)                  _mupdf.ll_fz_strsep_outparams_swiginit(self,
_mupdf.new_ll_fz_strsep_outparams())
40456: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_strsep_outparams
40457: (0)          _mupdf.ll_fz_strsep_outparams_swigregister(ll_fz_strsep_outparams)
40458: (0)          def ll_fz_strsep_outparams_fn(delim, outparams):
40459: (4)              return _mupdf.ll_fz_strsep_outparams_fn(delim, outparams)
40460: (0)          class ll_fz_strtof_outparams(object):
40461: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40462: (4)              __repr__ = _swig_repr
40463: (4)              es = property(_mupdf.ll_fz_strtof_outparams_es_get,
_mupdf.ll_fz_strtof_outparams_es_set)
40464: (4)              def __init__(self):
```

```
40465: (8)                  _mupdf.ll_fz_strtof_outparams_swiginit(self,
_mupdf.new_ll_fz_strtof_outparams())
40466: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_strtof_outparams
40467: (0)          _mupdf.ll_fz_strtof_outparams_swigregister(ll_fz_strtof_outparams)
40468: (0)          def ll_fz_strtof_outparams_fn(s, outparams):
40469: (4)              return _mupdf.ll_fz_strtof_outparams_fn(s, outparams)
40470: (0)          class ll_fz_subset_cff_for_gids_outparams(object):
40471: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40472: (4)              __repr__ = _swig_repr
40473: (4)              gids = property(_mupdf.ll_fz_subset_cff_for_gids_outparams_gids_get,
_mupdf.ll_fz_subset_cff_for_gids_outparams_gids_set)
40474: (4)              def __init__(self):
40475: (8)                  _mupdf.ll_fz_subset_cff_for_gids_outparams_swiginit(self,
_mupdf.new_ll_fz_subset_cff_for_gids_outparams())
40476: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_subset_cff_for_gids_outparams
40477: (0)
_mupdf.ll_fz_subset_cff_for_gids_outparams_swigregister(ll_fz_subset_cff_for_gids_outparams)
40478: (0)          def ll_fz_subset_cff_for_gids_outparams_fn(orig, num_gids, symbolic, cidfont,
outparams):
40479: (4)              return _mupdf.ll_fz_subset_cff_for_gids_outparams_fn(orig, num_gids,
symbolic, cidfont, outparams)
40480: (0)          class ll_fz_subset_ttf_for_gids_outparams(object):
40481: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40482: (4)              __repr__ = _swig_repr
40483: (4)              gids = property(_mupdf.ll_fz_subset_ttf_for_gids_outparams_gids_get,
_mupdf.ll_fz_subset_ttf_for_gids_outparams_gids_set)
40484: (4)              def __init__(self):
40485: (8)                  _mupdf.ll_fz_subset_ttf_for_gids_outparams_swiginit(self,
_mupdf.new_ll_fz_subset_ttf_for_gids_outparams())
40486: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_subset_ttf_for_gids_outparams
40487: (0)
_mupdf.ll_fz_subset_ttf_for_gids_outparams_swigregister(ll_fz_subset_ttf_for_gids_outparams)
40488: (0)          def ll_fz_subset_ttf_for_gids_outparams_fn(orig, num_gids, symbolic, cidfont,
outparams):
40489: (4)              return _mupdf.ll_fz_subset_ttf_for_gids_outparams_fn(orig, num_gids,
symbolic, cidfont, outparams)
40490: (0)          class ll_fz_warning_callback_outparams(object):
40491: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40492: (4)              __repr__ = _swig_repr
40493: (4)              user = property(_mupdf.ll_fz_warning_callback_outparams_user_get,
_mupdf.ll_fz_warning_callback_outparams_user_set)
40494: (4)              def __init__(self):
40495: (8)                  _mupdf.ll_fz_warning_callback_outparams_swiginit(self,
_mupdf.new_ll_fz_warning_callback_outparams())
40496: (4)              __swig_destroy__ = _mupdf.delete_ll_fz_warning_callback_outparams
40497: (0)
_mupdf.ll_fz_warning_callback_outparams_swigregister(ll_fz_warning_callback_outparams)
40498: (0)          def ll_fz_warning_callback_outparams_fn(outparams):
40499: (4)              return _mupdf.ll_fz_warning_callback_outparams_fn(outparams)
40500: (0)          class ll_pdf_annot_MK_BC_outparams(object):
40501: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40502: (4)              __repr__ = _swig_repr
40503: (4)              n = property(_mupdf.ll_pdf_annot_MK_BC_outparams_n_get,
_mupdf.ll_pdf_annot_MK_BC_outparams_n_set)
40504: (4)              def __init__(self):
40505: (8)                  _mupdf.ll_pdf_annot_MK_BC_outparams_swiginit(self,
_mupdf.new_ll_pdf_annot_MK_BC_outparams())
40506: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_MK_BC_outparams
40507: (0)          _mupdf.ll_pdf_annot_MK_BC_outparams_swigregister(ll_pdf_annot_MK_BC_outparams)
40508: (0)          def ll_pdf_annot_MK_BC_outparams_fn(annot, color, outparams):
40509: (4)              return _mupdf.ll_pdf_annot_MK_BC_outparams_fn(annot, color, outparams)
40510: (0)          class ll_pdf_annot_MK_BG_outparams(object):
40511: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40512: (4)              __repr__ = _swig_repr
```

```
40513: (4)              n = property(_mupdf.ll_pdf_annot_MK_BG_outparams_n_get,
_mupdf.ll_pdf_annot_MK_BG_outparams_n_set)
40514: (4)              def __init__(self):
40515: (8)                  _mupdf.ll_pdf_annot_MK_BG_outparams_swiginit(self,
_mupdf.new_ll_pdf_annot_MK_BG_outparams())
40516: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_MK_BG_outparams
40517: (0)          _mupdf.ll_pdf_annot_MK_BG_outparams_swigregister(ll_pdf_annot_MK_BG_outparams)
40518: (0)          def ll_pdf_annot_MK_BG_outparams_fn(annot, color, outparams):
40519: (4)              return _mupdf.ll_pdf_annot_MK_BG_outparams_fn(annot, color, outparams)
40520: (0)          class ll_pdf_annot_color_outparams(object):
40521: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40522: (4)              __repr__ = _swig_repr
40523: (4)              n = property(_mupdf.ll_pdf_annot_color_outparams_n_get,
_mupdf.ll_pdf_annot_color_outparams_n_set)
40524: (4)              def __init__(self):
40525: (8)                  _mupdf.ll_pdf_annot_color_outparams_swiginit(self,
_mupdf.new_ll_pdf_annot_color_outparams())
40526: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_color_outparams
40527: (0)          _mupdf.ll_pdf_annot_color_outparams_swigregister(ll_pdf_annot_color_outparams)
40528: (0)          def ll_pdf_annot_color_outparams_fn(annot, color, outparams):
40529: (4)              return _mupdf.ll_pdf_annot_color_outparams_fn(annot, color, outparams)
40530: (0)          class ll_pdf_annot_default_appearance_outparams(object):
40531: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40532: (4)              __repr__ = _swig_repr
40533: (4)              font = property(_mupdf.ll_pdf_annot_default_appearance_outparams_font_get,
_mupdf.ll_pdf_annot_default_appearance_outparams_font_set)
40534: (4)              size = property(_mupdf.ll_pdf_annot_default_appearance_outparams_size_get,
_mupdf.ll_pdf_annot_default_appearance_outparams_size_set)
40535: (4)              n = property(_mupdf.ll_pdf_annot_default_appearance_outparams_n_get,
_mupdf.ll_pdf_annot_default_appearance_outparams_n_set)
40536: (4)              def __init__(self):
40537: (8)                  _mupdf.ll_pdf_annot_default_appearance_outparams_swiginit(self,
_mupdf.new_ll_pdf_annot_default_appearance_outparams())
40538: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_default_appearance_outparams
40539: (0)
_mupdf.ll_pdf_annot_default_appearance_outparams_swigregister(ll_pdf_annot_default_appearance_outp
arams)
40540: (0)          def ll_pdf_annot_default_appearance_outparams_fn(annot, color, outparams):
40541: (4)              return _mupdf.ll_pdf_annot_default_appearance_outparams_fn(annot, color,
outparams)
40542: (0)          class ll_pdf_annot_interior_color_outparams(object):
40543: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40544: (4)              __repr__ = _swig_repr
40545: (4)              n = property(_mupdf.ll_pdf_annot_interior_color_outparams_n_get,
_mupdf.ll_pdf_annot_interior_color_outparams_n_set)
40546: (4)              def __init__(self):
40547: (8)                  _mupdf.ll_pdf_annot_interior_color_outparams_swiginit(self,
_mupdf.new_ll_pdf_annot_interior_color_outparams())
40548: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_interior_color_outparams
40549: (0)
_mupdf.ll_pdf_annot_interior_color_outparams_swigregister(ll_pdf_annot_interior_color_outparams)
40550: (0)          def ll_pdf_annot_interior_color_outparams_fn(annot, color, outparams):
40551: (4)              return _mupdf.ll_pdf_annot_interior_color_outparams_fn(annot, color,
outparams)
40552: (0)          class ll_pdf_annot_line_ending_styles_outparams(object):
40553: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40554: (4)              __repr__ = _swig_repr
40555: (4)              start_style =
property(_mupdf.ll_pdf_annot_line_ending_styles_outparams_start_style_get,
_mupdf.ll_pdf_annot_line_ending_styles_outparams_start_style_set)
40556: (4)              end_style =
property(_mupdf.ll_pdf_annot_line_ending_styles_outparams_end_style_get,
_mupdf.ll_pdf_annot_line_ending_styles_outparams_end_style_set)
40557: (4)              def __init__(self):
40558: (8)                  _mupdf.ll_pdf_annot_line_ending_styles_outparams_swiginit(self,
```

```
_mupdf.new_ll_pdf_annot_line_ending_styles_outparams())
40559: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_annot_line_ending_styles_outparams
40560: (0)
_mupdf.ll_pdf_annot_line_ending_styles_outparams_swigregister(ll_pdf_annot_line_ending_styles_outp
arams)
40561: (0)          def ll_pdf_annot_line_ending_styles_outparams_fn(annot, outparams):
40562: (4)              return _mupdf.ll_pdf_annot_line_ending_styles_outparams_fn(annot,
outparams)
40563: (0)          class ll_pdf_array_get_string_outparams(object):
40564: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40565: (4)              __repr__ = _swig_repr
40566: (4)              sizep = property(_mupdf.ll_pdf_array_get_string_outparams_sizep_get,
_mupdf.ll_pdf_array_get_string_outparams_sizep_set)
40567: (4)              def __init__(self):
40568: (8)                  _mupdf.ll_pdf_array_get_string_outparams_swiginit(self,
_mupdf.new_ll_pdf_array_get_string_outparams())
40569: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_array_get_string_outparams
40570: (0)
_mupdf.ll_pdf_array_get_string_outparams_swigregister(ll_pdf_array_get_string_outparams)
40571: (0)          def ll_pdf_array_get_string_outparams_fn(array, index, outparams):
40572: (4)              return _mupdf.ll_pdf_array_get_string_outparams_fn(array, index,
outparams)
40573: (0)          class ll_pdf_count_q_balance_outparams(object):
40574: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40575: (4)              __repr__ = _swig_repr
40576: (4)              prepend = property(_mupdf.ll_pdf_count_q_balance_outparams_prepend_get,
_mupdf.ll_pdf_count_q_balance_outparams_prepend_set)
40577: (4)              append = property(_mupdf.ll_pdf_count_q_balance_outparams_append_get,
_mupdf.ll_pdf_count_q_balance_outparams_append_set)
40578: (4)              def __init__(self):
40579: (8)                  _mupdf.ll_pdf_count_q_balance_outparams_swiginit(self,
_mupdf.new_ll_pdf_count_q_balance_outparams())
40580: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_count_q_balance_outparams
40581: (0)
_mupdf.ll_pdf_count_q_balance_outparams_swigregister(ll_pdf_count_q_balance_outparams)
40582: (0)          def ll_pdf_count_q_balance_outparams_fn(doc, res, stm, outparams):
40583: (4)              return _mupdf.ll_pdf_count_q_balance_outparams_fn(doc, res, stm,
outparams)
40584: (0)          class ll_pdf_decode_cmap_outparams(object):
40585: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40586: (4)              __repr__ = _swig_repr
40587: (4)              cpt = property(_mupdf.ll_pdf_decode_cmap_outparams_cpt_get,
_mupdf.ll_pdf_decode_cmap_outparams_cpt_set)
40588: (4)              def __init__(self):
40589: (8)                  _mupdf.ll_pdf_decode_cmap_outparams_swiginit(self,
_mupdf.new_ll_pdf_decode_cmap_outparams())
40590: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_decode_cmap_outparams
40591: (0)          _mupdf.ll_pdf_decode_cmap_outparams_swigregister(ll_pdf_decode_cmap_outparams)
40592: (0)          def ll_pdf_decode_cmap_outparams_fn(cmap, s, e, outparams):
40593: (4)              return _mupdf.ll_pdf_decode_cmap_outparams_fn(cmap, s, e, outparams)
40594: (0)          class ll_pdf_dict_get_inheritable_string_outparams(object):
40595: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40596: (4)              __repr__ = _swig_repr
40597: (4)              sizep =
property(_mupdf.ll_pdf_dict_get_inheritable_string_outparams_sizep_get,
_mupdf.ll_pdf_dict_get_inheritable_string_outparams_sizep_set)
40598: (4)              def __init__(self):
40599: (8)                  _mupdf.ll_pdf_dict_get_inheritable_string_outparams_swiginit(self,
_mupdf.new_ll_pdf_dict_get_inheritable_string_outparams())
40600: (4)              __swig_destroy__ =
_mupdf.delete_ll_pdf_dict_get_inheritable_string_outparams
40601: (0)
_mupdf.ll_pdf_dict_get_inheritable_string_outparams_swigregister(ll_pdf_dict_get_inheritable_strin
g_outparams)
40602: (0)          def ll_pdf_dict_get_inheritable_string_outparams_fn(dict, key, outparams):
```

```
40603: (4)              return _mupdf.ll_pdf_dict_get_inheritable_string_outparams_fn(dict, key,
outparams)
40604: (0)          class ll_pdf_dict_get_put_drop_outparams(object):
40605: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40606: (4)              __repr__ = _swig_repr
40607: (4)              old_val = property(_mupdf.ll_pdf_dict_get_put_drop_outparams_old_val_get,
_mupdf.ll_pdf_dict_get_put_drop_outparams_old_val_set)
40608: (4)              def __init__(self):
40609: (8)                  _mupdf.ll_pdf_dict_get_put_drop_outparams_swiginit(self,
_mupdf.new_ll_pdf_dict_get_put_drop_outparams())
40610: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_dict_get_put_drop_outparams
40611: (0)
_mupdf.ll_pdf_dict_get_put_drop_outparams_swigregister(ll_pdf_dict_get_put_drop_outparams)
40612: (0)          def ll_pdf_dict_get_put_drop_outparams_fn(dict, key, val, outparams):
40613: (4)              return _mupdf.ll_pdf_dict_get_put_drop_outparams_fn(dict, key, val,
outparams)
40614: (0)          class ll_pdf_dict_get_string_outparams(object):
40615: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40616: (4)              __repr__ = _swig_repr
40617: (4)              sizep = property(_mupdf.ll_pdf_dict_get_string_outparams_sizep_get,
_mupdf.ll_pdf_dict_get_string_outparams_sizep_set)
40618: (4)              def __init__(self):
40619: (8)                  _mupdf.ll_pdf_dict_get_string_outparams_swiginit(self,
_mupdf.new_ll_pdf_dict_get_string_outparams())
40620: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_dict_get_string_outparams
40621: (0)
_mupdf.ll_pdf_dict_get_string_outparams_swigregister(ll_pdf_dict_get_string_outparams)
40622: (0)          def ll_pdf_dict_get_string_outparams_fn(dict, key, outparams):
40623: (4)              return _mupdf.ll_pdf_dict_get_string_outparams_fn(dict, key, outparams)
40624: (0)          class ll_pdf_edit_text_field_value_outparams(object):
40625: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40626: (4)              __repr__ = _swig_repr
40627: (4)              selStart =
property(_mupdf.ll_pdf_edit_text_field_value_outparams_selStart_get,
_mupdf.ll_pdf_edit_text_field_value_outparams_selStart_set)
40628: (4)              selEnd =
property(_mupdf.ll_pdf_edit_text_field_value_outparams_selEnd_get,
_mupdf.ll_pdf_edit_text_field_value_outparams_selEnd_set)
40629: (4)              newvalue =
property(_mupdf.ll_pdf_edit_text_field_value_outparams_newvalue_get,
_mupdf.ll_pdf_edit_text_field_value_outparams_newvalue_set)
40630: (4)              def __init__(self):
40631: (8)                  _mupdf.ll_pdf_edit_text_field_value_outparams_swiginit(self,
_mupdf.new_ll_pdf_edit_text_field_value_outparams())
40632: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_edit_text_field_value_outparams
40633: (0)
_mupdf.ll_pdf_edit_text_field_value_outparams_swigregister(ll_pdf_edit_text_field_value_outparams)
40634: (0)          def ll_pdf_edit_text_field_value_outparams_fn(widget, value, change,
outparams):
40635: (4)              return _mupdf.ll_pdf_edit_text_field_value_outparams_fn(widget, value,
change, outparams)
40636: (0)          class ll_pdf_eval_function_outparams(object):
40637: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40638: (4)              __repr__ = _swig_repr
40639: (4)              out = property(_mupdf.ll_pdf_eval_function_outparams_out_get,
_mupdf.ll_pdf_eval_function_outparams_out_set)
40640: (4)              def __init__(self):
40641: (8)                  _mupdf.ll_pdf_eval_function_outparams_swiginit(self,
_mupdf.new_ll_pdf_eval_function_outparams())
40642: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_eval_function_outparams
40643: (0)
_mupdf.ll_pdf_eval_function_outparams_swigregister(ll_pdf_eval_function_outparams)
40644: (0)          def ll_pdf_eval_function_outparams_fn(func, _in, inlen, outlen, outparams):
40645: (4)              return _mupdf.ll_pdf_eval_function_outparams_fn(func, _in, inlen, outlen,
outparams)
```

```
40646: (0)          class ll_pdf_field_event_validate_outparams(object):
40647: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40648: (4)              __repr__ = _swig_repr
40649: (4)              newvalue =
property(_mupdf.ll_pdf_field_event_validate_outparams_newvalue_get,
_mupdf.ll_pdf_field_event_validate_outparams_newvalue_set)
40650: (4)              def __init__(self):
40651: (8)                  _mupdf.ll_pdf_field_event_validate_outparams_swiginit(self,
_mupdf.new_ll_pdf_field_event_validate_outparams())
40652: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_field_event_validate_outparams
40653: (0)
_mupdf.ll_pdf_field_event_validate_outparams_swigregister(ll_pdf_field_event_validate_outparams)
40654: (0)          def ll_pdf_field_event_validate_outparams_fn(doc, field, value, outparams):
40655: (4)              return _mupdf.ll_pdf_field_event_validate_outparams_fn(doc, field, value,
outparams)
40656: (0)          class ll_pdf_js_event_result_validate_outparams(object):
40657: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40658: (4)              __repr__ = _swig_repr
40659: (4)              newvalue =
property(_mupdf.ll_pdf_js_event_result_validate_outparams_newvalue_get,
_mupdf.ll_pdf_js_event_result_validate_outparams_newvalue_set)
40660: (4)              def __init__(self):
40661: (8)                  _mupdf.ll_pdf_js_event_result_validate_outparams_swiginit(self,
_mupdf.new_ll_pdf_js_event_result_validate_outparams())
40662: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_js_event_result_validate_outparams
40663: (0)
_mupdf.ll_pdf_js_event_result_validate_outparams_swigregister(ll_pdf_js_event_result_validate_outp
arams)
40664: (0)          def ll_pdf_js_event_result_validate_outparams_fn(js, outparams):
40665: (4)              return _mupdf.ll_pdf_js_event_result_validate_outparams_fn(js, outparams)
40666: (0)          class ll_pdf_js_execute_outparams(object):
40667: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40668: (4)              __repr__ = _swig_repr
40669: (4)              result = property(_mupdf.ll_pdf_js_execute_outparams_result_get,
_mupdf.ll_pdf_js_execute_outparams_result_set)
40670: (4)              def __init__(self):
40671: (8)                  _mupdf.ll_pdf_js_execute_outparams_swiginit(self,
_mupdf.new_ll_pdf_js_execute_outparams())
40672: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_js_execute_outparams
40673: (0)          _mupdf.ll_pdf_js_execute_outparams_swigregister(ll_pdf_js_execute_outparams)
40674: (0)          def ll_pdf_js_execute_outparams_fn(js, name, code, outparams):
40675: (4)              return _mupdf.ll_pdf_js_execute_outparams_fn(js, name, code, outparams)
40676: (0)          class ll_pdf_load_encoding_outparams(object):
40677: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40678: (4)              __repr__ = _swig_repr
40679: (4)              estrings = property(_mupdf.ll_pdf_load_encoding_outparams_estrings_get,
_mupdf.ll_pdf_load_encoding_outparams_estrings_set)
40680: (4)              def __init__(self):
40681: (8)                  _mupdf.ll_pdf_load_encoding_outparams_swiginit(self,
_mupdf.new_ll_pdf_load_encoding_outparams())
40682: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_load_encoding_outparams
40683: (0)
_mupdf.ll_pdf_load_encoding_outparams_swigregister(ll_pdf_load_encoding_outparams)
40684: (0)          def ll_pdf_load_encoding_outparams_fn(encoding, outparams):
40685: (4)              return _mupdf.ll_pdf_load_encoding_outparams_fn(encoding, outparams)
40686: (0)          class ll_pdf_load_to_unicode_outparams(object):
40687: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40688: (4)              __repr__ = _swig_repr
40689: (4)              strings = property(_mupdf.ll_pdf_load_to_unicode_outparams_strings_get,
_mupdf.ll_pdf_load_to_unicode_outparams_strings_set)
40690: (4)              def __init__(self):
40691: (8)                  _mupdf.ll_pdf_load_to_unicode_outparams_swiginit(self,
_mupdf.new_ll_pdf_load_to_unicode_outparams())
40692: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_load_to_unicode_outparams
```

```
40693: (0)
_mupdf.ll_pdf_load_to_unicode_outparams_swigregister(ll_pdf_load_to_unicode_outparams)
40694: (0)          def ll_pdf_load_to_unicode_outparams_fn(doc, font, collection, cmapstm,
outparams):
40695: (4)              return _mupdf.ll_pdf_load_to_unicode_outparams_fn(doc, font, collection,
cmapstm, outparams)
40696: (0)          class ll_pdf_lookup_cmap_full_outparams(object):
40697: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40698: (4)              __repr__ = _swig_repr
40699: (4)              out = property(_mupdf.ll_pdf_lookup_cmap_full_outparams_out_get,
_mupdf.ll_pdf_lookup_cmap_full_outparams_out_set)
40700: (4)              def __init__(self):
40701: (8)                  _mupdf.ll_pdf_lookup_cmap_full_outparams_swiginit(self,
_mupdf.new_ll_pdf_lookup_cmap_full_outparams())
40702: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_lookup_cmap_full_outparams
40703: (0)
_mupdf.ll_pdf_lookup_cmap_full_outparams_swigregister(ll_pdf_lookup_cmap_full_outparams)
40704: (0)          def ll_pdf_lookup_cmap_full_outparams_fn(cmap, cpt, outparams):
40705: (4)              return _mupdf.ll_pdf_lookup_cmap_full_outparams_fn(cmap, cpt, outparams)
40706: (0)          class ll_pdf_lookup_page_loc_outparams(object):
40707: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40708: (4)              __repr__ = _swig_repr
40709: (4)              parentp = property(_mupdf.ll_pdf_lookup_page_loc_outparams_parentp_get,
_mupdf.ll_pdf_lookup_page_loc_outparams_parentp_set)
40710: (4)              indexp = property(_mupdf.ll_pdf_lookup_page_loc_outparams_indexp_get,
_mupdf.ll_pdf_lookup_page_loc_outparams_indexp_set)
40711: (4)              def __init__(self):
40712: (8)                  _mupdf.ll_pdf_lookup_page_loc_outparams_swiginit(self,
_mupdf.new_ll_pdf_lookup_page_loc_outparams())
40713: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_lookup_page_loc_outparams
40714: (0)
_mupdf.ll_pdf_lookup_page_loc_outparams_swigregister(ll_pdf_lookup_page_loc_outparams)
40715: (0)          def ll_pdf_lookup_page_loc_outparams_fn(doc, needle, outparams):
40716: (4)              return _mupdf.ll_pdf_lookup_page_loc_outparams_fn(doc, needle, outparams)
40717: (0)          class ll_pdf_lookup_substitute_font_outparams(object):
40718: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40719: (4)              __repr__ = _swig_repr
40720: (4)              len = property(_mupdf.ll_pdf_lookup_substitute_font_outparams_len_get,
_mupdf.ll_pdf_lookup_substitute_font_outparams_len_set)
40721: (4)              def __init__(self):
40722: (8)                  _mupdf.ll_pdf_lookup_substitute_font_outparams_swiginit(self,
_mupdf.new_ll_pdf_lookup_substitute_font_outparams())
40723: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_lookup_substitute_font_outparams
40724: (0)
_mupdf.ll_pdf_lookup_substitute_font_outparams_swigregister(ll_pdf_lookup_substitute_font_outparam
s)
40725: (0)          def ll_pdf_lookup_substitute_font_outparams_fn(mono, serif, bold, italic,
outparams):
40726: (4)              return _mupdf.ll_pdf_lookup_substitute_font_outparams_fn(mono, serif,
bold, italic, outparams)
40727: (0)          class ll_pdf_map_one_to_many_outparams(object):
40728: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40729: (4)              __repr__ = _swig_repr
40730: (4)              many = property(_mupdf.ll_pdf_map_one_to_many_outparams_many_get,
_mupdf.ll_pdf_map_one_to_many_outparams_many_set)
40731: (4)              def __init__(self):
40732: (8)                  _mupdf.ll_pdf_map_one_to_many_outparams_swiginit(self,
_mupdf.new_ll_pdf_map_one_to_many_outparams())
40733: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_map_one_to_many_outparams
40734: (0)
_mupdf.ll_pdf_map_one_to_many_outparams_swigregister(ll_pdf_map_one_to_many_outparams)
40735: (0)          def ll_pdf_map_one_to_many_outparams_fn(cmap, one, len, outparams):
40736: (4)              return _mupdf.ll_pdf_map_one_to_many_outparams_fn(cmap, one, len,
outparams)
40737: (0)          class ll_pdf_obj_memo_outparams(object):
```

```
40738: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40739: (4)              __repr__ = _swig_repr
40740: (4)              memo = property(_mupdf.ll_pdf_obj_memo_outparams_memo_get,
_mupdf.ll_pdf_obj_memo_outparams_memo_set)
40741: (4)              def __init__(self):
40742: (8)                  _mupdf.ll_pdf_obj_memo_outparams_swiginit(self,
_mupdf.new_ll_pdf_obj_memo_outparams())
40743: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_obj_memo_outparams
40744: (0)          _mupdf.ll_pdf_obj_memo_outparams_swigregister(ll_pdf_obj_memo_outparams)
40745: (0)          def ll_pdf_obj_memo_outparams_fn(obj, bit, outparams):
40746: (4)              return _mupdf.ll_pdf_obj_memo_outparams_fn(obj, bit, outparams)
40747: (0)          class ll_pdf_page_presentation_outparams(object):
40748: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40749: (4)              __repr__ = _swig_repr
40750: (4)              duration =
property(_mupdf.ll_pdf_page_presentation_outparams_duration_get,
_mupdf.ll_pdf_page_presentation_outparams_duration_set)
40751: (4)              def __init__(self):
40752: (8)                  _mupdf.ll_pdf_page_presentation_outparams_swiginit(self,
_mupdf.new_ll_pdf_page_presentation_outparams())
40753: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_page_presentation_outparams
40754: (0)
_mupdf.ll_pdf_page_presentation_outparams_swigregister(ll_pdf_page_presentation_outparams)
40755: (0)          def ll_pdf_page_presentation_outparams_fn(page, transition, outparams):
40756: (4)              return _mupdf.ll_pdf_page_presentation_outparams_fn(page, transition,
outparams)
40757: (0)          class ll_pdf_page_write_outparams(object):
40758: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40759: (4)              __repr__ = _swig_repr
40760: (4)              presources = property(_mupdf.ll_pdf_page_write_outparams_presources_get,
_mupdf.ll_pdf_page_write_outparams_presources_set)
40761: (4)              pcontents = property(_mupdf.ll_pdf_page_write_outparams_pcontents_get,
_mupdf.ll_pdf_page_write_outparams_pcontents_set)
40762: (4)              def __init__(self):
40763: (8)                  _mupdf.ll_pdf_page_write_outparams_swiginit(self,
_mupdf.new_ll_pdf_page_write_outparams())
40764: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_page_write_outparams
40765: (0)          _mupdf.ll_pdf_page_write_outparams_swigregister(ll_pdf_page_write_outparams)
40766: (0)          def ll_pdf_page_write_outparams_fn(doc, mediabox, outparams):
40767: (4)              return _mupdf.ll_pdf_page_write_outparams_fn(doc, mediabox, outparams)
40768: (0)          class ll_pdf_parse_default_appearance_outparams(object):
40769: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40770: (4)              __repr__ = _swig_repr
40771: (4)              font = property(_mupdf.ll_pdf_parse_default_appearance_outparams_font_get,
_mupdf.ll_pdf_parse_default_appearance_outparams_font_set)
40772: (4)              size = property(_mupdf.ll_pdf_parse_default_appearance_outparams_size_get,
_mupdf.ll_pdf_parse_default_appearance_outparams_size_set)
40773: (4)              n = property(_mupdf.ll_pdf_parse_default_appearance_outparams_n_get,
_mupdf.ll_pdf_parse_default_appearance_outparams_n_set)
40774: (4)              def __init__(self):
40775: (8)                  _mupdf.ll_pdf_parse_default_appearance_outparams_swiginit(self,
_mupdf.new_ll_pdf_parse_default_appearance_outparams())
40776: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_parse_default_appearance_outparams
40777: (0)
_mupdf.ll_pdf_parse_default_appearance_outparams_swigregister(ll_pdf_parse_default_appearance_outp
arams)
40778: (0)          def ll_pdf_parse_default_appearance_outparams_fn(da, color, outparams):
40779: (4)              return _mupdf.ll_pdf_parse_default_appearance_outparams_fn(da, color,
outparams)
40780: (0)          class ll_pdf_parse_ind_obj_outparams(object):
40781: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40782: (4)              __repr__ = _swig_repr
40783: (4)              num = property(_mupdf.ll_pdf_parse_ind_obj_outparams_num_get,
_mupdf.ll_pdf_parse_ind_obj_outparams_num_set)
```

```
40784: (4)              gen = property(_mupdf.ll_pdf_parse_ind_obj_outparams_gen_get,
_mupdf.ll_pdf_parse_ind_obj_outparams_gen_set)
40785: (4)              stm_ofs = property(_mupdf.ll_pdf_parse_ind_obj_outparams_stm_ofs_get,
_mupdf.ll_pdf_parse_ind_obj_outparams_stm_ofs_set)
40786: (4)              try_repair =
property(_mupdf.ll_pdf_parse_ind_obj_outparams_try_repair_get,
_mupdf.ll_pdf_parse_ind_obj_outparams_try_repair_set)
40787: (4)              def __init__(self):
40788: (8)                  _mupdf.ll_pdf_parse_ind_obj_outparams_swiginit(self,
_mupdf.new_ll_pdf_parse_ind_obj_outparams())
40789: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_parse_ind_obj_outparams
40790: (0)
_mupdf.ll_pdf_parse_ind_obj_outparams_swigregister(ll_pdf_parse_ind_obj_outparams)
40791: (0)          def ll_pdf_parse_ind_obj_outparams_fn(doc, f, outparams):
40792: (4)              return _mupdf.ll_pdf_parse_ind_obj_outparams_fn(doc, f, outparams)
40793: (0)          class ll_pdf_parse_journal_obj_outparams(object):
40794: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40795: (4)              __repr__ = _swig_repr
40796: (4)              onum = property(_mupdf.ll_pdf_parse_journal_obj_outparams_onum_get,
_mupdf.ll_pdf_parse_journal_obj_outparams_onum_set)
40797: (4)              ostm = property(_mupdf.ll_pdf_parse_journal_obj_outparams_ostm_get,
_mupdf.ll_pdf_parse_journal_obj_outparams_ostm_set)
40798: (4)              newobj = property(_mupdf.ll_pdf_parse_journal_obj_outparams_newobj_get,
_mupdf.ll_pdf_parse_journal_obj_outparams_newobj_set)
40799: (4)              def __init__(self):
40800: (8)                  _mupdf.ll_pdf_parse_journal_obj_outparams_swiginit(self,
_mupdf.new_ll_pdf_parse_journal_obj_outparams())
40801: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_parse_journal_obj_outparams
40802: (0)
_mupdf.ll_pdf_parse_journal_obj_outparams_swigregister(ll_pdf_parse_journal_obj_outparams)
40803: (0)          def ll_pdf_parse_journal_obj_outparams_fn(doc, stm, outparams):
40804: (4)              return _mupdf.ll_pdf_parse_journal_obj_outparams_fn(doc, stm, outparams)
40805: (0)          class ll_pdf_print_encrypted_obj_outparams(object):
40806: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40807: (4)              __repr__ = _swig_repr
40808: (4)              sep = property(_mupdf.ll_pdf_print_encrypted_obj_outparams_sep_get,
_mupdf.ll_pdf_print_encrypted_obj_outparams_sep_set)
40809: (4)              def __init__(self):
40810: (8)                  _mupdf.ll_pdf_print_encrypted_obj_outparams_swiginit(self,
_mupdf.new_ll_pdf_print_encrypted_obj_outparams())
40811: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_print_encrypted_obj_outparams
40812: (0)
_mupdf.ll_pdf_print_encrypted_obj_outparams_swigregister(ll_pdf_print_encrypted_obj_outparams)
40813: (0)          def ll_pdf_print_encrypted_obj_outparams_fn(out, obj, tight, ascii, crypt,
num, gen, outparams):
40814: (4)              return _mupdf.ll_pdf_print_encrypted_obj_outparams_fn(out, obj, tight,
ascii, crypt, num, gen, outparams)
40815: (0)          class ll_pdf_process_contents_outparams(object):
40816: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40817: (4)              __repr__ = _swig_repr
40818: (4)              out_res = property(_mupdf.ll_pdf_process_contents_outparams_out_res_get,
_mupdf.ll_pdf_process_contents_outparams_out_res_set)
40819: (4)              def __init__(self):
40820: (8)                  _mupdf.ll_pdf_process_contents_outparams_swiginit(self,
_mupdf.new_ll_pdf_process_contents_outparams())
40821: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_process_contents_outparams
40822: (0)
_mupdf.ll_pdf_process_contents_outparams_swigregister(ll_pdf_process_contents_outparams)
40823: (0)          def ll_pdf_process_contents_outparams_fn(proc, doc, res, stm, cookie,
outparams):
40824: (4)              return _mupdf.ll_pdf_process_contents_outparams_fn(proc, doc, res, stm,
cookie, outparams)
40825: (0)          class ll_pdf_repair_obj_outparams(object):
40826: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40827: (4)              __repr__ = _swig_repr
```

```
40828: (4)              stmofsp = property(_mupdf.ll_pdf_repair_obj_outparams_stmofsp_get,
_mupdf.ll_pdf_repair_obj_outparams_stmofsp_set)
40829: (4)              stmlenp = property(_mupdf.ll_pdf_repair_obj_outparams_stmlenp_get,
_mupdf.ll_pdf_repair_obj_outparams_stmlenp_set)
40830: (4)              encrypt = property(_mupdf.ll_pdf_repair_obj_outparams_encrypt_get,
_mupdf.ll_pdf_repair_obj_outparams_encrypt_set)
40831: (4)              id = property(_mupdf.ll_pdf_repair_obj_outparams_id_get,
_mupdf.ll_pdf_repair_obj_outparams_id_set)
40832: (4)              page = property(_mupdf.ll_pdf_repair_obj_outparams_page_get,
_mupdf.ll_pdf_repair_obj_outparams_page_set)
40833: (4)              tmpofs = property(_mupdf.ll_pdf_repair_obj_outparams_tmpofs_get,
_mupdf.ll_pdf_repair_obj_outparams_tmpofs_set)
40834: (4)              root = property(_mupdf.ll_pdf_repair_obj_outparams_root_get,
_mupdf.ll_pdf_repair_obj_outparams_root_set)
40835: (4)              def __init__(self):
40836: (8)                  _mupdf.ll_pdf_repair_obj_outparams_swiginit(self,
_mupdf.new_ll_pdf_repair_obj_outparams())
40837: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_repair_obj_outparams
40838: (0)          _mupdf.ll_pdf_repair_obj_outparams_swigregister(ll_pdf_repair_obj_outparams)
40839: (0)          def ll_pdf_repair_obj_outparams_fn(doc, buf, outparams):
40840: (4)              return _mupdf.ll_pdf_repair_obj_outparams_fn(doc, buf, outparams)
40841: (0)          class ll_pdf_resolve_link_outparams(object):
40842: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40843: (4)              __repr__ = _swig_repr
40844: (4)              xp = property(_mupdf.ll_pdf_resolve_link_outparams_xp_get,
_mupdf.ll_pdf_resolve_link_outparams_xp_set)
40845: (4)              yp = property(_mupdf.ll_pdf_resolve_link_outparams_yp_get,
_mupdf.ll_pdf_resolve_link_outparams_yp_set)
40846: (4)              def __init__(self):
40847: (8)                  _mupdf.ll_pdf_resolve_link_outparams_swiginit(self,
_mupdf.new_ll_pdf_resolve_link_outparams())
40848: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_resolve_link_outparams
40849: (0)
_mupdf.ll_pdf_resolve_link_outparams_swigregister(ll_pdf_resolve_link_outparams)
40850: (0)          def ll_pdf_resolve_link_outparams_fn(doc, uri, outparams):
40851: (4)              return _mupdf.ll_pdf_resolve_link_outparams_fn(doc, uri, outparams)
40852: (0)          class ll_pdf_sample_shade_function_outparams(object):
40853: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40854: (4)              __repr__ = _swig_repr
40855: (4)              samples =
property(_mupdf.ll_pdf_sample_shade_function_outparams_samples_get,
_mupdf.ll_pdf_sample_shade_function_outparams_samples_set)
40856: (4)              func = property(_mupdf.ll_pdf_sample_shade_function_outparams_func_get,
_mupdf.ll_pdf_sample_shade_function_outparams_func_set)
40857: (4)              def __init__(self):
40858: (8)                  _mupdf.ll_pdf_sample_shade_function_outparams_swiginit(self,
_mupdf.new_ll_pdf_sample_shade_function_outparams())
40859: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_sample_shade_function_outparams
40860: (0)
_mupdf.ll_pdf_sample_shade_function_outparams_swigregister(ll_pdf_sample_shade_function_outparams)
40861: (0)          def ll_pdf_sample_shade_function_outparams_fn(n, funcs, t0, t1, outparams):
40862: (4)              return _mupdf.ll_pdf_sample_shade_function_outparams_fn(n, funcs, t0, t1,
outparams)
40863: (0)          class ll_pdf_signature_contents_outparams(object):
40864: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40865: (4)              __repr__ = _swig_repr
40866: (4)              contents =
property(_mupdf.ll_pdf_signature_contents_outparams_contents_get,
_mupdf.ll_pdf_signature_contents_outparams_contents_set)
40867: (4)              def __init__(self):
40868: (8)                  _mupdf.ll_pdf_signature_contents_outparams_swiginit(self,
_mupdf.new_ll_pdf_signature_contents_outparams())
40869: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_signature_contents_outparams
40870: (0)
_mupdf.ll_pdf_signature_contents_outparams_swigregister(ll_pdf_signature_contents_outparams)
40871: (0)          def ll_pdf_signature_contents_outparams_fn(doc, signature, outparams):
```

```
40872: (4)              return _mupdf.ll_pdf_signature_contents_outparams_fn(doc, signature,
outparams)
40873: (0)          class ll_pdf_sprint_obj_outparams(object):
40874: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40875: (4)              __repr__ = _swig_repr
40876: (4)              len = property(_mupdf.ll_pdf_sprint_obj_outparams_len_get,
_mupdf.ll_pdf_sprint_obj_outparams_len_set)
40877: (4)              def __init__(self):
40878: (8)                  _mupdf.ll_pdf_sprint_obj_outparams_swiginit(self,
_mupdf.new_ll_pdf_sprint_obj_outparams())
40879: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_sprint_obj_outparams
40880: (0)          _mupdf.ll_pdf_sprint_obj_outparams_swigregister(ll_pdf_sprint_obj_outparams)
40881: (0)          def ll_pdf_sprint_obj_outparams_fn(buf, cap, obj, tight, ascii, outparams):
40882: (4)              return _mupdf.ll_pdf_sprint_obj_outparams_fn(buf, cap, obj, tight, ascii,
outparams)
40883: (0)          class ll_pdf_to_string_outparams(object):
40884: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40885: (4)              __repr__ = _swig_repr
40886: (4)              sizep = property(_mupdf.ll_pdf_to_string_outparams_sizep_get,
_mupdf.ll_pdf_to_string_outparams_sizep_set)
40887: (4)              def __init__(self):
40888: (8)                  _mupdf.ll_pdf_to_string_outparams_swiginit(self,
_mupdf.new_ll_pdf_to_string_outparams())
40889: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_to_string_outparams
40890: (0)          _mupdf.ll_pdf_to_string_outparams_swigregister(ll_pdf_to_string_outparams)
40891: (0)          def ll_pdf_to_string_outparams_fn(obj, outparams):
40892: (4)              return _mupdf.ll_pdf_to_string_outparams_fn(obj, outparams)
40893: (0)          class ll_pdf_undoredo_state_outparams(object):
40894: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40895: (4)              __repr__ = _swig_repr
40896: (4)              steps = property(_mupdf.ll_pdf_undoredo_state_outparams_steps_get,
_mupdf.ll_pdf_undoredo_state_outparams_steps_set)
40897: (4)              def __init__(self):
40898: (8)                  _mupdf.ll_pdf_undoredo_state_outparams_swiginit(self,
_mupdf.new_ll_pdf_undoredo_state_outparams())
40899: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_undoredo_state_outparams
40900: (0)
_mupdf.ll_pdf_undoredo_state_outparams_swigregister(ll_pdf_undoredo_state_outparams)
40901: (0)          def ll_pdf_undoredo_state_outparams_fn(doc, outparams):
40902: (4)              return _mupdf.ll_pdf_undoredo_state_outparams_fn(doc, outparams)
40903: (0)          class ll_pdf_walk_tree_outparams(object):
40904: (4)              thisown = property(lambda x: x.this.own(), lambda x, v: x.this.own(v),
doc="The membership flag")
40905: (4)              __repr__ = _swig_repr
40906: (4)              names = property(_mupdf.ll_pdf_walk_tree_outparams_names_get,
_mupdf.ll_pdf_walk_tree_outparams_names_set)
40907: (4)              values = property(_mupdf.ll_pdf_walk_tree_outparams_values_get,
_mupdf.ll_pdf_walk_tree_outparams_values_set)
40908: (4)              def __init__(self):
40909: (8)                  _mupdf.ll_pdf_walk_tree_outparams_swiginit(self,
_mupdf.new_ll_pdf_walk_tree_outparams())
40910: (4)              __swig_destroy__ = _mupdf.delete_ll_pdf_walk_tree_outparams
40911: (0)          _mupdf.ll_pdf_walk_tree_outparams_swigregister(ll_pdf_walk_tree_outparams)
40912: (0)          def ll_pdf_walk_tree_outparams_fn(tree, kid_name, arrive, leave, arg,
outparams):
40913: (4)              return _mupdf.ll_pdf_walk_tree_outparams_fn(tree, kid_name, arrive, leave,
arg, outparams)
40914: (0)          UCDN_EAST_ASIAN_F = _mupdf.UCDN_EAST_ASIAN_F
40915: (0)          UCDN_EAST_ASIAN_H = _mupdf.UCDN_EAST_ASIAN_H
40916: (0)          UCDN_EAST_ASIAN_W = _mupdf.UCDN_EAST_ASIAN_W
40917: (0)          UCDN_EAST_ASIAN_NA = _mupdf.UCDN_EAST_ASIAN_NA
40918: (0)          UCDN_EAST_ASIAN_A = _mupdf.UCDN_EAST_ASIAN_A
40919: (0)          UCDN_EAST_ASIAN_N = _mupdf.UCDN_EAST_ASIAN_N
40920: (0)          UCDN_SCRIPT_COMMON = _mupdf.UCDN_SCRIPT_COMMON
40921: (0)          UCDN_SCRIPT_LATIN = _mupdf.UCDN_SCRIPT_LATIN
40922: (0)          UCDN_SCRIPT_GREEK = _mupdf.UCDN_SCRIPT_GREEK
```

```
40923: (0)          UCDN_SCRIPT_CYRILLIC = _mupdf.UCDN_SCRIPT_CYRILLIC
40924: (0)          UCDN_SCRIPT_ARMENIAN = _mupdf.UCDN_SCRIPT_ARMENIAN
40925: (0)          UCDN_SCRIPT_HEBREW = _mupdf.UCDN_SCRIPT_HEBREW
40926: (0)          UCDN_SCRIPT_ARABIC = _mupdf.UCDN_SCRIPT_ARABIC
40927: (0)          UCDN_SCRIPT_SYRIAC = _mupdf.UCDN_SCRIPT_SYRIAC
40928: (0)          UCDN_SCRIPT_THAANA = _mupdf.UCDN_SCRIPT_THAANA
40929: (0)          UCDN_SCRIPT_DEVANAGARI = _mupdf.UCDN_SCRIPT_DEVANAGARI
40930: (0)          UCDN_SCRIPT_BENGALI = _mupdf.UCDN_SCRIPT_BENGALI
40931: (0)          UCDN_SCRIPT_GURMUKHI = _mupdf.UCDN_SCRIPT_GURMUKHI
40932: (0)          UCDN_SCRIPT_GUJARATI = _mupdf.UCDN_SCRIPT_GUJARATI
40933: (0)          UCDN_SCRIPT_ORIYA = _mupdf.UCDN_SCRIPT_ORIYA
40934: (0)          UCDN_SCRIPT_TAMIL = _mupdf.UCDN_SCRIPT_TAMIL
40935: (0)          UCDN_SCRIPT_TELUGU = _mupdf.UCDN_SCRIPT_TELUGU
40936: (0)          UCDN_SCRIPT_KANNADA = _mupdf.UCDN_SCRIPT_KANNADA
40937: (0)          UCDN_SCRIPT_MALAYALAM = _mupdf.UCDN_SCRIPT_MALAYALAM
40938: (0)          UCDN_SCRIPT_SINHALA = _mupdf.UCDN_SCRIPT_SINHALA
40939: (0)          UCDN_SCRIPT_THAI = _mupdf.UCDN_SCRIPT_THAI
40940: (0)          UCDN_SCRIPT_LAO = _mupdf.UCDN_SCRIPT_LAO
40941: (0)          UCDN_SCRIPT_TIBETAN = _mupdf.UCDN_SCRIPT_TIBETAN
40942: (0)          UCDN_SCRIPT_MYANMAR = _mupdf.UCDN_SCRIPT_MYANMAR
40943: (0)          UCDN_SCRIPT_GEORGIAN = _mupdf.UCDN_SCRIPT_GEORGIAN
40944: (0)          UCDN_SCRIPT_HANGUL = _mupdf.UCDN_SCRIPT_HANGUL
40945: (0)          UCDN_SCRIPT_ETHIOPIC = _mupdf.UCDN_SCRIPT_ETHIOPIC
40946: (0)          UCDN_SCRIPT_CHEROKEE = _mupdf.UCDN_SCRIPT_CHEROKEE
40947: (0)          UCDN_SCRIPT_CANADIAN_ABORIGINAL = _mupdf.UCDN_SCRIPT_CANADIAN_ABORIGINAL
40948: (0)          UCDN_SCRIPT_OGHAM = _mupdf.UCDN_SCRIPT_OGHAM
40949: (0)          UCDN_SCRIPT_RUNIC = _mupdf.UCDN_SCRIPT_RUNIC
40950: (0)          UCDN_SCRIPT_KHMER = _mupdf.UCDN_SCRIPT_KHMER
40951: (0)          UCDN_SCRIPT_MONGOLIAN = _mupdf.UCDN_SCRIPT_MONGOLIAN
40952: (0)          UCDN_SCRIPT_HIRAGANA = _mupdf.UCDN_SCRIPT_HIRAGANA
40953: (0)          UCDN_SCRIPT_KATAKANA = _mupdf.UCDN_SCRIPT_KATAKANA
40954: (0)          UCDN_SCRIPT_BOPOMOFO = _mupdf.UCDN_SCRIPT_BOPOMOFO
40955: (0)          UCDN_SCRIPT_HAN = _mupdf.UCDN_SCRIPT_HAN
40956: (0)          UCDN_SCRIPT_YI = _mupdf.UCDN_SCRIPT_YI
40957: (0)          UCDN_SCRIPT_OLD_ITALIC = _mupdf.UCDN_SCRIPT_OLD_ITALIC
40958: (0)          UCDN_SCRIPT_GOTHIC = _mupdf.UCDN_SCRIPT_GOTHIC
40959: (0)          UCDN_SCRIPT_DESERET = _mupdf.UCDN_SCRIPT_DESERET
40960: (0)          UCDN_SCRIPT_INHERITED = _mupdf.UCDN_SCRIPT_INHERITED
40961: (0)          UCDN_SCRIPT_TAGALOG = _mupdf.UCDN_SCRIPT_TAGALOG
40962: (0)          UCDN_SCRIPT_HANUNOO = _mupdf.UCDN_SCRIPT_HANUNOO
40963: (0)          UCDN_SCRIPT_BUHID = _mupdf.UCDN_SCRIPT_BUHID
40964: (0)          UCDN_SCRIPT_TAGBANWA = _mupdf.UCDN_SCRIPT_TAGBANWA
40965: (0)          UCDN_SCRIPT_LIMBU = _mupdf.UCDN_SCRIPT_LIMBU
40966: (0)          UCDN_SCRIPT_TAI_LE = _mupdf.UCDN_SCRIPT_TAI_LE
40967: (0)          UCDN_SCRIPT_LINEAR_B = _mupdf.UCDN_SCRIPT_LINEAR_B
40968: (0)          UCDN_SCRIPT_UGARITIC = _mupdf.UCDN_SCRIPT_UGARITIC
40969: (0)          UCDN_SCRIPT_SHAVIAN = _mupdf.UCDN_SCRIPT_SHAVIAN
40970: (0)          UCDN_SCRIPT_OSMANYA = _mupdf.UCDN_SCRIPT_OSMANYA
40971: (0)          UCDN_SCRIPT_CYPRIOT = _mupdf.UCDN_SCRIPT_CYPRIOT
40972: (0)          UCDN_SCRIPT_BRAILLE = _mupdf.UCDN_SCRIPT_BRAILLE
40973: (0)          UCDN_SCRIPT_BUGINESE = _mupdf.UCDN_SCRIPT_BUGINESE
40974: (0)          UCDN_SCRIPT_COPTIC = _mupdf.UCDN_SCRIPT_COPTIC
40975: (0)          UCDN_SCRIPT_NEW_TAI_LUE = _mupdf.UCDN_SCRIPT_NEW_TAI_LUE
40976: (0)          UCDN_SCRIPT_GLAGOLITIC = _mupdf.UCDN_SCRIPT_GLAGOLITIC
40977: (0)          UCDN_SCRIPT_TIFINAGH = _mupdf.UCDN_SCRIPT_TIFINAGH
40978: (0)          UCDN_SCRIPT_SYLOTI_NAGRI = _mupdf.UCDN_SCRIPT_SYLOTI_NAGRI
40979: (0)          UCDN_SCRIPT_OLD_PERSIAN = _mupdf.UCDN_SCRIPT_OLD_PERSIAN
40980: (0)          UCDN_SCRIPT_KHAROSHTHI = _mupdf.UCDN_SCRIPT_KHAROSHTHI
40981: (0)          UCDN_SCRIPT_BALINESE = _mupdf.UCDN_SCRIPT_BALINESE
40982: (0)          UCDN_SCRIPT_CUNEIFORM = _mupdf.UCDN_SCRIPT_CUNEIFORM
40983: (0)          UCDN_SCRIPT_PHOENICIAN = _mupdf.UCDN_SCRIPT_PHOENICIAN
40984: (0)          UCDN_SCRIPT_PHAGS_PA = _mupdf.UCDN_SCRIPT_PHAGS_PA
40985: (0)          UCDN_SCRIPT_NKO = _mupdf.UCDN_SCRIPT_NKO
40986: (0)          UCDN_SCRIPT_SUNDANESE = _mupdf.UCDN_SCRIPT_SUNDANESE
40987: (0)          UCDN_SCRIPT_LEPCHA = _mupdf.UCDN_SCRIPT_LEPCHA
40988: (0)          UCDN_SCRIPT_OL_CHIKI = _mupdf.UCDN_SCRIPT_OL_CHIKI
40989: (0)          UCDN_SCRIPT_VAI = _mupdf.UCDN_SCRIPT_VAI
40990: (0)          UCDN_SCRIPT_SAURASHTRA = _mupdf.UCDN_SCRIPT_SAURASHTRA
40991: (0)          UCDN_SCRIPT_KAYAH_LI = _mupdf.UCDN_SCRIPT_KAYAH_LI
```

```
40992: (0)          UCDN_SCRIPT_REJANG = _mupdf.UCDN_SCRIPT_REJANG
40993: (0)          UCDN_SCRIPT_LYCIAN = _mupdf.UCDN_SCRIPT_LYCIAN
40994: (0)          UCDN_SCRIPT_CARIAN = _mupdf.UCDN_SCRIPT_CARIAN
40995: (0)          UCDN_SCRIPT_LYDIAN = _mupdf.UCDN_SCRIPT_LYDIAN
40996: (0)          UCDN_SCRIPT_CHAM = _mupdf.UCDN_SCRIPT_CHAM
40997: (0)          UCDN_SCRIPT_TAI_THAM = _mupdf.UCDN_SCRIPT_TAI_THAM
40998: (0)          UCDN_SCRIPT_TAI_VIET = _mupdf.UCDN_SCRIPT_TAI_VIET
40999: (0)          UCDN_SCRIPT_AVESTAN = _mupdf.UCDN_SCRIPT_AVESTAN
41000: (0)          UCDN_SCRIPT_EGYPTIAN_HIEROGLYPHS = _mupdf.UCDN_SCRIPT_EGYPTIAN_HIEROGLYPHS
41001: (0)          UCDN_SCRIPT_SAMARITAN = _mupdf.UCDN_SCRIPT_SAMARITAN
41002: (0)          UCDN_SCRIPT_LISU = _mupdf.UCDN_SCRIPT_LISU
41003: (0)          UCDN_SCRIPT_BAMUM = _mupdf.UCDN_SCRIPT_BAMUM
41004: (0)          UCDN_SCRIPT_JAVANESE = _mupdf.UCDN_SCRIPT_JAVANESE
41005: (0)          UCDN_SCRIPT_MEETEI_MAYEK = _mupdf.UCDN_SCRIPT_MEETEI_MAYEK
41006: (0)          UCDN_SCRIPT_IMPERIAL_ARAMAIC = _mupdf.UCDN_SCRIPT_IMPERIAL_ARAMAIC
41007: (0)          UCDN_SCRIPT_OLD_SOUTH_ARABIAN = _mupdf.UCDN_SCRIPT_OLD_SOUTH_ARABIAN
41008: (0)          UCDN_SCRIPT_INSCRIPTIONAL_PARTHIAN = _mupdf.UCDN_SCRIPT_INSCRIPTIONAL_PARTHIAN
41009: (0)          UCDN_SCRIPT_INSCRIPTIONAL_PAHLAVI = _mupdf.UCDN_SCRIPT_INSCRIPTIONAL_PAHLAVI
41010: (0)          UCDN_SCRIPT_OLD_TURKIC = _mupdf.UCDN_SCRIPT_OLD_TURKIC
41011: (0)          UCDN_SCRIPT_KAITHI = _mupdf.UCDN_SCRIPT_KAITHI
41012: (0)          UCDN_SCRIPT_BATAK = _mupdf.UCDN_SCRIPT_BATAK
41013: (0)          UCDN_SCRIPT_BRAHMI = _mupdf.UCDN_SCRIPT_BRAHMI
41014: (0)          UCDN_SCRIPT_MANDAIC = _mupdf.UCDN_SCRIPT_MANDAIC
41015: (0)          UCDN_SCRIPT_CHAKMA = _mupdf.UCDN_SCRIPT_CHAKMA
41016: (0)          UCDN_SCRIPT_MEROITIC_CURSIVE = _mupdf.UCDN_SCRIPT_MEROITIC_CURSIVE
41017: (0)          UCDN_SCRIPT_MEROITIC_HIEROGLYPHS = _mupdf.UCDN_SCRIPT_MEROITIC_HIEROGLYPHS
41018: (0)          UCDN_SCRIPT_MIAO = _mupdf.UCDN_SCRIPT_MIAO
41019: (0)          UCDN_SCRIPT_SHARADA = _mupdf.UCDN_SCRIPT_SHARADA
41020: (0)          UCDN_SCRIPT_SORA_SOMPENG = _mupdf.UCDN_SCRIPT_SORA_SOMPENG
41021: (0)          UCDN_SCRIPT_TAKRI = _mupdf.UCDN_SCRIPT_TAKRI
41022: (0)          UCDN_SCRIPT_UNKNOWN = _mupdf.UCDN_SCRIPT_UNKNOWN
41023: (0)          UCDN_SCRIPT_BASSA_VAH = _mupdf.UCDN_SCRIPT_BASSA_VAH
41024: (0)          UCDN_SCRIPT_CAUCASIAN_ALBANIAN = _mupdf.UCDN_SCRIPT_CAUCASIAN_ALBANIAN
41025: (0)          UCDN_SCRIPT_DUPLOYAN = _mupdf.UCDN_SCRIPT_DUPLOYAN
41026: (0)          UCDN_SCRIPT_ELBASAN = _mupdf.UCDN_SCRIPT_ELBASAN
41027: (0)          UCDN_SCRIPT_GRANTHA = _mupdf.UCDN_SCRIPT_GRANTHA
41028: (0)          UCDN_SCRIPT_KHOJKI = _mupdf.UCDN_SCRIPT_KHOJKI
41029: (0)          UCDN_SCRIPT_KHUDAWADI = _mupdf.UCDN_SCRIPT_KHUDAWADI
41030: (0)          UCDN_SCRIPT_LINEAR_A = _mupdf.UCDN_SCRIPT_LINEAR_A
41031: (0)          UCDN_SCRIPT_MAHAJANI = _mupdf.UCDN_SCRIPT_MAHAJANI
41032: (0)          UCDN_SCRIPT_MANICHAEAN = _mupdf.UCDN_SCRIPT_MANICHAEAN
41033: (0)          UCDN_SCRIPT_MENDE_KIKAKUI = _mupdf.UCDN_SCRIPT_MENDE_KIKAKUI
41034: (0)          UCDN_SCRIPT_MODI = _mupdf.UCDN_SCRIPT_MODI
41035: (0)          UCDN_SCRIPT_MRO = _mupdf.UCDN_SCRIPT_MRO
41036: (0)          UCDN_SCRIPT_NABATAEAN = _mupdf.UCDN_SCRIPT_NABATAEAN
41037: (0)          UCDN_SCRIPT_OLD_NORTH_ARABIAN = _mupdf.UCDN_SCRIPT_OLD_NORTH_ARABIAN
41038: (0)          UCDN_SCRIPT_OLD_PERMIC = _mupdf.UCDN_SCRIPT_OLD_PERMIC
41039: (0)          UCDN_SCRIPT_PAHAWH_HMONG = _mupdf.UCDN_SCRIPT_PAHAWH_HMONG
41040: (0)          UCDN_SCRIPT_PALMYRENE = _mupdf.UCDN_SCRIPT_PALMYRENE
41041: (0)          UCDN_SCRIPT_PAU_CIN_HAU = _mupdf.UCDN_SCRIPT_PAU_CIN_HAU
41042: (0)          UCDN_SCRIPT_PSALTER_PAHLAVI = _mupdf.UCDN_SCRIPT_PSALTER_PAHLAVI
41043: (0)          UCDN_SCRIPT_SIDDHAM = _mupdf.UCDN_SCRIPT_SIDDHAM
41044: (0)          UCDN_SCRIPT_TIRHUTA = _mupdf.UCDN_SCRIPT_TIRHUTA
41045: (0)          UCDN_SCRIPT_WARANG_CITI = _mupdf.UCDN_SCRIPT_WARANG_CITI
41046: (0)          UCDN_SCRIPT_AHOM = _mupdf.UCDN_SCRIPT_AHOM
41047: (0)          UCDN_SCRIPT_ANATOLIAN_HIEROGLYPHS = _mupdf.UCDN_SCRIPT_ANATOLIAN_HIEROGLYPHS
41048: (0)          UCDN_SCRIPT_HATRAN = _mupdf.UCDN_SCRIPT_HATRAN
41049: (0)          UCDN_SCRIPT_MULTANI = _mupdf.UCDN_SCRIPT_MULTANI
41050: (0)          UCDN_SCRIPT_OLD_HUNGARIAN = _mupdf.UCDN_SCRIPT_OLD_HUNGARIAN
41051: (0)          UCDN_SCRIPT_SIGNWRITING = _mupdf.UCDN_SCRIPT_SIGNWRITING
41052: (0)          UCDN_SCRIPT_ADLAM = _mupdf.UCDN_SCRIPT_ADLAM
41053: (0)          UCDN_SCRIPT_BHAIKSUKI = _mupdf.UCDN_SCRIPT_BHAIKSUKI
41054: (0)          UCDN_SCRIPT_MARCHEN = _mupdf.UCDN_SCRIPT_MARCHEN
41055: (0)          UCDN_SCRIPT_NEWA = _mupdf.UCDN_SCRIPT_NEWA
41056: (0)          UCDN_SCRIPT_OSAGE = _mupdf.UCDN_SCRIPT_OSAGE
41057: (0)          UCDN_SCRIPT_TANGUT = _mupdf.UCDN_SCRIPT_TANGUT
41058: (0)          UCDN_SCRIPT_MASARAM_GONDI = _mupdf.UCDN_SCRIPT_MASARAM_GONDI
41059: (0)          UCDN_SCRIPT_NUSHU = _mupdf.UCDN_SCRIPT_NUSHU
41060: (0)          UCDN_SCRIPT_SOYOMBO = _mupdf.UCDN_SCRIPT_SOYOMBO
```

```
41061: (0)          UCDN_SCRIPT_ZANABAZAR_SQUARE = _mupdf.UCDN_SCRIPT_ZANABAZAR_SQUARE
41062: (0)          UCDN_SCRIPT_DOGRA = _mupdf.UCDN_SCRIPT_DOGRA
41063: (0)          UCDN_SCRIPT_GUNJALA_GONDI = _mupdf.UCDN_SCRIPT_GUNJALA_GONDI
41064: (0)          UCDN_SCRIPT_HANIFI_ROHINGYA = _mupdf.UCDN_SCRIPT_HANIFI_ROHINGYA
41065: (0)          UCDN_SCRIPT_MAKASAR = _mupdf.UCDN_SCRIPT_MAKASAR
41066: (0)          UCDN_SCRIPT_MEDEFAIDRIN = _mupdf.UCDN_SCRIPT_MEDEFAIDRIN
41067: (0)          UCDN_SCRIPT_OLD_SOGDIAN = _mupdf.UCDN_SCRIPT_OLD_SOGDIAN
41068: (0)          UCDN_SCRIPT_SOGDIAN = _mupdf.UCDN_SCRIPT_SOGDIAN
41069: (0)          UCDN_SCRIPT_ELYMAIC = _mupdf.UCDN_SCRIPT_ELYMAIC
41070: (0)          UCDN_SCRIPT_NANDINAGARI = _mupdf.UCDN_SCRIPT_NANDINAGARI
41071: (0)          UCDN_SCRIPT_NYIAKENG_PUACHUE_HMONG = _mupdf.UCDN_SCRIPT_NYIAKENG_PUACHUE_HMONG
41072: (0)          UCDN_SCRIPT_WANCHO = _mupdf.UCDN_SCRIPT_WANCHO
41073: (0)          UCDN_SCRIPT_CHORASMIAN = _mupdf.UCDN_SCRIPT_CHORASMIAN
41074: (0)          UCDN_SCRIPT_DIVES_AKURU = _mupdf.UCDN_SCRIPT_DIVES_AKURU
41075: (0)          UCDN_SCRIPT_KHITAN_SMALL_SCRIPT = _mupdf.UCDN_SCRIPT_KHITAN_SMALL_SCRIPT
41076: (0)          UCDN_SCRIPT_YEZIDI = _mupdf.UCDN_SCRIPT_YEZIDI
41077: (0)          UCDN_SCRIPT_VITHKUQI = _mupdf.UCDN_SCRIPT_VITHKUQI
41078: (0)          UCDN_SCRIPT_OLD_UYGHUR = _mupdf.UCDN_SCRIPT_OLD_UYGHUR
41079: (0)          UCDN_SCRIPT_CYPRO_MINOAN = _mupdf.UCDN_SCRIPT_CYPRO_MINOAN
41080: (0)          UCDN_SCRIPT_TANGSA = _mupdf.UCDN_SCRIPT_TANGSA
41081: (0)          UCDN_SCRIPT_TOTO = _mupdf.UCDN_SCRIPT_TOTO
41082: (0)          UCDN_SCRIPT_KAWI = _mupdf.UCDN_SCRIPT_KAWI
41083: (0)          UCDN_SCRIPT_NAG_MUNDARI = _mupdf.UCDN_SCRIPT_NAG_MUNDARI
41084: (0)          UCDN_LAST_SCRIPT = _mupdf.UCDN_LAST_SCRIPT
41085: (0)          UCDN_LINEBREAK_CLASS_OP = _mupdf.UCDN_LINEBREAK_CLASS_OP
41086: (0)          UCDN_LINEBREAK_CLASS_CL = _mupdf.UCDN_LINEBREAK_CLASS_CL
41087: (0)          UCDN_LINEBREAK_CLASS_CP = _mupdf.UCDN_LINEBREAK_CLASS_CP
41088: (0)          UCDN_LINEBREAK_CLASS_QU = _mupdf.UCDN_LINEBREAK_CLASS_QU
41089: (0)          UCDN_LINEBREAK_CLASS_GL = _mupdf.UCDN_LINEBREAK_CLASS_GL
41090: (0)          UCDN_LINEBREAK_CLASS_NS = _mupdf.UCDN_LINEBREAK_CLASS_NS
41091: (0)          UCDN_LINEBREAK_CLASS_EX = _mupdf.UCDN_LINEBREAK_CLASS_EX
41092: (0)          UCDN_LINEBREAK_CLASS_SY = _mupdf.UCDN_LINEBREAK_CLASS_SY
41093: (0)          UCDN_LINEBREAK_CLASS_IS = _mupdf.UCDN_LINEBREAK_CLASS_IS
41094: (0)          UCDN_LINEBREAK_CLASS_PR = _mupdf.UCDN_LINEBREAK_CLASS_PR
41095: (0)          UCDN_LINEBREAK_CLASS_PO = _mupdf.UCDN_LINEBREAK_CLASS_PO
41096: (0)          UCDN_LINEBREAK_CLASS_NU = _mupdf.UCDN_LINEBREAK_CLASS_NU
41097: (0)          UCDN_LINEBREAK_CLASS_AL = _mupdf.UCDN_LINEBREAK_CLASS_AL
41098: (0)          UCDN_LINEBREAK_CLASS_HL = _mupdf.UCDN_LINEBREAK_CLASS_HL
41099: (0)          UCDN_LINEBREAK_CLASS_ID = _mupdf.UCDN_LINEBREAK_CLASS_ID
41100: (0)          UCDN_LINEBREAK_CLASS_IN = _mupdf.UCDN_LINEBREAK_CLASS_IN
41101: (0)          UCDN_LINEBREAK_CLASS_HY = _mupdf.UCDN_LINEBREAK_CLASS_HY
41102: (0)          UCDN_LINEBREAK_CLASS_BA = _mupdf.UCDN_LINEBREAK_CLASS_BA
41103: (0)          UCDN_LINEBREAK_CLASS_BB = _mupdf.UCDN_LINEBREAK_CLASS_BB
41104: (0)          UCDN_LINEBREAK_CLASS_B2 = _mupdf.UCDN_LINEBREAK_CLASS_B2
41105: (0)          UCDN_LINEBREAK_CLASS_ZW = _mupdf.UCDN_LINEBREAK_CLASS_ZW
41106: (0)          UCDN_LINEBREAK_CLASS_CM = _mupdf.UCDN_LINEBREAK_CLASS_CM
41107: (0)          UCDN_LINEBREAK_CLASS_WJ = _mupdf.UCDN_LINEBREAK_CLASS_WJ
41108: (0)          UCDN_LINEBREAK_CLASS_H2 = _mupdf.UCDN_LINEBREAK_CLASS_H2
41109: (0)          UCDN_LINEBREAK_CLASS_H3 = _mupdf.UCDN_LINEBREAK_CLASS_H3
41110: (0)          UCDN_LINEBREAK_CLASS_JL = _mupdf.UCDN_LINEBREAK_CLASS_JL
41111: (0)          UCDN_LINEBREAK_CLASS_JV = _mupdf.UCDN_LINEBREAK_CLASS_JV
41112: (0)          UCDN_LINEBREAK_CLASS_JT = _mupdf.UCDN_LINEBREAK_CLASS_JT
41113: (0)          UCDN_LINEBREAK_CLASS_RI = _mupdf.UCDN_LINEBREAK_CLASS_RI
41114: (0)          UCDN_LINEBREAK_CLASS_AI = _mupdf.UCDN_LINEBREAK_CLASS_AI
41115: (0)          UCDN_LINEBREAK_CLASS_BK = _mupdf.UCDN_LINEBREAK_CLASS_BK
41116: (0)          UCDN_LINEBREAK_CLASS_CB = _mupdf.UCDN_LINEBREAK_CLASS_CB
41117: (0)          UCDN_LINEBREAK_CLASS_CJ = _mupdf.UCDN_LINEBREAK_CLASS_CJ
41118: (0)          UCDN_LINEBREAK_CLASS_CR = _mupdf.UCDN_LINEBREAK_CLASS_CR
41119: (0)          UCDN_LINEBREAK_CLASS_LF = _mupdf.UCDN_LINEBREAK_CLASS_LF
41120: (0)          UCDN_LINEBREAK_CLASS_NL = _mupdf.UCDN_LINEBREAK_CLASS_NL
41121: (0)          UCDN_LINEBREAK_CLASS_SA = _mupdf.UCDN_LINEBREAK_CLASS_SA
41122: (0)          UCDN_LINEBREAK_CLASS_SG = _mupdf.UCDN_LINEBREAK_CLASS_SG
41123: (0)          UCDN_LINEBREAK_CLASS_SP = _mupdf.UCDN_LINEBREAK_CLASS_SP
41124: (0)          UCDN_LINEBREAK_CLASS_XX = _mupdf.UCDN_LINEBREAK_CLASS_XX
41125: (0)          UCDN_LINEBREAK_CLASS_ZWJ = _mupdf.UCDN_LINEBREAK_CLASS_ZWJ
41126: (0)          UCDN_LINEBREAK_CLASS_EB = _mupdf.UCDN_LINEBREAK_CLASS_EB
41127: (0)          UCDN_LINEBREAK_CLASS_EM = _mupdf.UCDN_LINEBREAK_CLASS_EM
41128: (0)          UCDN_GENERAL_CATEGORY_CC = _mupdf.UCDN_GENERAL_CATEGORY_CC
41129: (0)          UCDN_GENERAL_CATEGORY_CF = _mupdf.UCDN_GENERAL_CATEGORY_CF
```

```
41130: (0)          UCDN_GENERAL_CATEGORY_CN = _mupdf.UCDN_GENERAL_CATEGORY_CN
41131: (0)          UCDN_GENERAL_CATEGORY_CO = _mupdf.UCDN_GENERAL_CATEGORY_CO
41132: (0)          UCDN_GENERAL_CATEGORY_CS = _mupdf.UCDN_GENERAL_CATEGORY_CS
41133: (0)          UCDN_GENERAL_CATEGORY_LL = _mupdf.UCDN_GENERAL_CATEGORY_LL
41134: (0)          UCDN_GENERAL_CATEGORY_LM = _mupdf.UCDN_GENERAL_CATEGORY_LM
41135: (0)          UCDN_GENERAL_CATEGORY_LO = _mupdf.UCDN_GENERAL_CATEGORY_LO
41136: (0)          UCDN_GENERAL_CATEGORY_LT = _mupdf.UCDN_GENERAL_CATEGORY_LT
41137: (0)          UCDN_GENERAL_CATEGORY_LU = _mupdf.UCDN_GENERAL_CATEGORY_LU
41138: (0)          UCDN_GENERAL_CATEGORY_MC = _mupdf.UCDN_GENERAL_CATEGORY_MC
41139: (0)          UCDN_GENERAL_CATEGORY_ME = _mupdf.UCDN_GENERAL_CATEGORY_ME
41140: (0)          UCDN_GENERAL_CATEGORY_MN = _mupdf.UCDN_GENERAL_CATEGORY_MN
41141: (0)          UCDN_GENERAL_CATEGORY_ND = _mupdf.UCDN_GENERAL_CATEGORY_ND
41142: (0)          UCDN_GENERAL_CATEGORY_NL = _mupdf.UCDN_GENERAL_CATEGORY_NL
41143: (0)          UCDN_GENERAL_CATEGORY_NO = _mupdf.UCDN_GENERAL_CATEGORY_NO
41144: (0)          UCDN_GENERAL_CATEGORY_PC = _mupdf.UCDN_GENERAL_CATEGORY_PC
41145: (0)          UCDN_GENERAL_CATEGORY_PD = _mupdf.UCDN_GENERAL_CATEGORY_PD
41146: (0)          UCDN_GENERAL_CATEGORY_PE = _mupdf.UCDN_GENERAL_CATEGORY_PE
41147: (0)          UCDN_GENERAL_CATEGORY_PF = _mupdf.UCDN_GENERAL_CATEGORY_PF
41148: (0)          UCDN_GENERAL_CATEGORY_PI = _mupdf.UCDN_GENERAL_CATEGORY_PI
41149: (0)          UCDN_GENERAL_CATEGORY_PO = _mupdf.UCDN_GENERAL_CATEGORY_PO
41150: (0)          UCDN_GENERAL_CATEGORY_PS = _mupdf.UCDN_GENERAL_CATEGORY_PS
41151: (0)          UCDN_GENERAL_CATEGORY_SC = _mupdf.UCDN_GENERAL_CATEGORY_SC
41152: (0)          UCDN_GENERAL_CATEGORY_SK = _mupdf.UCDN_GENERAL_CATEGORY_SK
41153: (0)          UCDN_GENERAL_CATEGORY_SM = _mupdf.UCDN_GENERAL_CATEGORY_SM
41154: (0)          UCDN_GENERAL_CATEGORY_SO = _mupdf.UCDN_GENERAL_CATEGORY_SO
41155: (0)          UCDN_GENERAL_CATEGORY_ZL = _mupdf.UCDN_GENERAL_CATEGORY_ZL
41156: (0)          UCDN_GENERAL_CATEGORY_ZP = _mupdf.UCDN_GENERAL_CATEGORY_ZP
41157: (0)          UCDN_GENERAL_CATEGORY_ZS = _mupdf.UCDN_GENERAL_CATEGORY_ZS
41158: (0)          UCDN_BIDI_CLASS_L = _mupdf.UCDN_BIDI_CLASS_L
41159: (0)          UCDN_BIDI_CLASS_LRE = _mupdf.UCDN_BIDI_CLASS_LRE
41160: (0)          UCDN_BIDI_CLASS_LRO = _mupdf.UCDN_BIDI_CLASS_LRO
41161: (0)          UCDN_BIDI_CLASS_R = _mupdf.UCDN_BIDI_CLASS_R
41162: (0)          UCDN_BIDI_CLASS_AL = _mupdf.UCDN_BIDI_CLASS_AL
41163: (0)          UCDN_BIDI_CLASS_RLE = _mupdf.UCDN_BIDI_CLASS_RLE
41164: (0)          UCDN_BIDI_CLASS_RLO = _mupdf.UCDN_BIDI_CLASS_RLO
41165: (0)          UCDN_BIDI_CLASS_PDF = _mupdf.UCDN_BIDI_CLASS_PDF
41166: (0)          UCDN_BIDI_CLASS_EN = _mupdf.UCDN_BIDI_CLASS_EN
41167: (0)          UCDN_BIDI_CLASS_ES = _mupdf.UCDN_BIDI_CLASS_ES
41168: (0)          UCDN_BIDI_CLASS_ET = _mupdf.UCDN_BIDI_CLASS_ET
41169: (0)          UCDN_BIDI_CLASS_AN = _mupdf.UCDN_BIDI_CLASS_AN
41170: (0)          UCDN_BIDI_CLASS_CS = _mupdf.UCDN_BIDI_CLASS_CS
41171: (0)          UCDN_BIDI_CLASS_NSM = _mupdf.UCDN_BIDI_CLASS_NSM
41172: (0)          UCDN_BIDI_CLASS_BN = _mupdf.UCDN_BIDI_CLASS_BN
41173: (0)          UCDN_BIDI_CLASS_B = _mupdf.UCDN_BIDI_CLASS_B
41174: (0)          UCDN_BIDI_CLASS_S = _mupdf.UCDN_BIDI_CLASS_S
41175: (0)          UCDN_BIDI_CLASS_WS = _mupdf.UCDN_BIDI_CLASS_WS
41176: (0)          UCDN_BIDI_CLASS_ON = _mupdf.UCDN_BIDI_CLASS_ON
41177: (0)          UCDN_BIDI_CLASS_LRI = _mupdf.UCDN_BIDI_CLASS_LRI
41178: (0)          UCDN_BIDI_CLASS_RLI = _mupdf.UCDN_BIDI_CLASS_RLI
41179: (0)          UCDN_BIDI_CLASS_FSI = _mupdf.UCDN_BIDI_CLASS_FSI
41180: (0)          UCDN_BIDI_CLASS_PDI = _mupdf.UCDN_BIDI_CLASS_PDI
41181: (0)          UCDN_BIDI_PAIRED_BRACKET_TYPE_OPEN = _mupdf.UCDN_BIDI_PAIRED_BRACKET_TYPE_OPEN
41182: (0)          UCDN_BIDI_PAIRED_BRACKET_TYPE_CLOSE =
_mupdf.UCDN_BIDI_PAIRED_BRACKET_TYPE_CLOSE
41183: (0)          UCDN_BIDI_PAIRED_BRACKET_TYPE_NONE = _mupdf.UCDN_BIDI_PAIRED_BRACKET_TYPE_NONE
41184: (0)          def new_pint():
41185: (4)              return _mupdf.new_pint()
41186: (0)          def copy_pint(value):
41187: (4)              return _mupdf.copy_pint(value)
41188: (0)          def delete_pint(obj):
41189: (4)              return _mupdf.delete_pint(obj)
41190: (0)          def pint_assign(obj, value):
41191: (4)              return _mupdf.pint_assign(obj, value)
41192: (0)          def pint_value(obj):
41193: (4)              return _mupdf.pint_value(obj)
41194: (0)          import inspect
41195: (0)          import os
41196: (0)          import re
41197: (0)          import sys
```

```
41198: (0)          import traceback
41199: (0)          def log( text):
41200: (4)              print( text, file=sys.stderr)
41201: (0)          g_mupdf_trace_director = (os.environ.get('MUPDF_trace_director') == '1')
41202: (0)          def fz_lookup_metadata(document, key):
41203: (4)              """
41204: (4)              Like fz_lookup_metadata2() but returns None on error
41205: (4)              instead of raising exception.
41206: (4)              """
41207: (4)              try:
41208: (8)                  return fz_lookup_metadata2(document, key)
41209: (4)              except Exception:
41210: (8)                  return
41211: (0)          FzDocument.fz_lookup_metadata                        = fz_lookup_metadata
41212: (0)          def pdf_lookup_metadata(document, key):
41213: (4)              """
41214: (4)              Likepsd_lookup_metadata2() but returns None on error
41215: (4)              instead of raising exception.
41216: (4)              """
41217: (4)              try:
41218: (8)                  return pdf_lookup_metadata2(document, key)
41219: (4)              except Exception:
41220: (8)                  return
41221: (0)          PdfDocument.pdf_lookup_metadata                        = pdf_lookup_metadata
41222: (0)          import inspect
41223: (0)          import io
41224: (0)          import os
41225: (0)          import sys
41226: (0)          import traceback
41227: (0)          import types
41228: (0)          def exception_info(
41229: (8)                  exception_or_traceback=None,
41230: (8)                  limit=None,
41231: (8)                  file=None,
41232: (8)                  chain=True,
41233: (8)                  outer=True,
41234: (8)                  show_exception_type=True,
41235: (8)                  _filelinefn=True,
41236: (8)                  ):
41237: (4)              '''
41238: (4)              Shows an exception and/or backtrace.
41239: (4)              Alternative to `traceback.*` functions that print/return information about
41240: (4)              exceptions and backtraces, such as:
41241: (8)                  * `traceback.format_exc()`
41242: (8)                  * `traceback.format_exception()`
41243: (8)                  * `traceback.print_exc()`
41244: (8)                  * `traceback.print_exception()`
41245: (4)              Install as system default with:
41246: (8)                  `sys.excepthook = lambda type_, exception, traceback:
jlib.exception_info( exception)`
41247: (4)              Returns `None`, or the generated text if `file` is 'return'.
41248: (4)              Args:
41249: (8)                  exception_or_traceback:
41250: (12)                     `None`, a `BaseException`, a `types.TracebackType` (typically from
41251: (12)                     an exception's `.__traceback__` member) or an `inspect.FrameInfo`.
41252: (12)                     If `None` we use current exception from `sys.exc_info()` if set,
41253: (12)                     otherwise the current backtrace from `inspect.stack()`.
41254: (8)                  limit:
41255: (12)                     As in `traceback.*` functions: `None` to show all frames, positive
41256: (12)                     to show last `limit` frames, negative to exclude outermost `-
limit`
41257: (12)                     frames. Zero to not show any backtraces.
41258: (8)                  file:
41259: (12)                     As in `traceback.*` functions: file-like object to which we write
41260: (12)                     output, or `sys.stderr` if `None`. Special value 'return' makes us
41261: (12)                     return our output as a string.
41262: (8)                  chain:
41263: (12)                     As in `traceback.*` functions: if true (the default) we show
41264: (12)                     chained exceptions as described in PEP-3134. Special value
```

```
(12)                    'because' reverses the usual ordering, showing higher-level
(12)                    exceptions first and joining with 'Because:' text.
(8)                 outer:
(12)                    If true (the default) we also show an exception's outer frames
(12)                    above the `catch` block (see next section for details). We
(12)                    use `outer=false` internally for chained exceptions to avoid
(12)                    duplication.
(8)                 show_exception_type:
(12)                    Controls whether exception text is prefixed by
(12)                    `f'{type(exception)}: '`. If callable we only include this prefix
(12)                    if `show_exception_type(exception)` is true. Otherwise if true
(the
(12)                    default) we include the prefix for all exceptions (this mimcs the
(12)                    behaviour of `traceback.*` functions). Otherwise we exclude the
(12)                    prefix for all exceptions.
(8)                 _filelinefn:
(12)                    Internal only; makes us omit file:line: information to allow
simple
(12)                    doctest comparison with expected output.
(4)             Differences from `traceback.*` functions:
(8)                 Frames are displayed as one line in the form::
(12)                    <file>:<line>:<function>: <text>
(8)                 Filenames are displayed as relative to the current directory if
(8)                 applicable.
(8)                 Inclusion of outer frames:
(12)                    Unlike `traceback.*` functions, stack traces for exceptions
include
(12)                    outer stack frames above the point at which an exception was
caught
(12)                    - i.e. frames from the top-level <module> or thread creation to
the
(12)                    catch block. [Search for 'sys.exc_info backtrace incomplete' for
(12)                    more details.]
(12)                    We separate the two parts of the backtrace using a marker line
(12)                    '^except raise:' where '^except' points upwards to the frame that
(12)                    caught the exception and 'raise:' refers downwards to the frame
(12)                    that raised the exception.
(12)                    So the backtrace for an exception looks like this::
(16)                        <file>:<line>:<fn>: <text>  [in root module.]
(16)                        ...                         [... other frames]
(16)                        <file>:<line>:<fn>: <text>  [in except: block where exception
was caught.]
(16)                        ^except raise:              [marker line]
(16)                        <file>:<line>:<fn>: <text>  [in try: block.]
(16)                        ...                         [... other frames]
(16)                        <file>:<line>:<fn>: <text>  [where the exception was raised.]
(4)             Examples:
(8)                 In these examples we use `file=sys.stdout` so we can check the output
(8)                 with `doctest`, and set `_filelinefn=0` so that the output can be
(8)                 matched easily. We also use `+ELLIPSIS` and `...` to match arbitrary
(8)                 outer frames from the doctest code itself.
(8)                 Basic handling of an exception:
(12)                    >>> def c():
(12)                    ...     raise Exception( 'c() failed')
(12)                    >>> def b():
(12)                    ...     try:
(12)                    ...         c()
(12)                    ...     except Exception as e:
(12)                    ...         exception_info( e, file=sys.stdout, _filelinefn=0)
(12)                    >>> def a():
(12)                    ...     b()
(12)                    >>> a() # doctest: +REPORT_UDIFF +ELLIPSIS
(12)                    Traceback (most recent call last):
(16)                        ...
(16)                        a(): b()
(16)                        b(): exception_info( e, file=sys.stdout, _filelinefn=0)
(16)                        ^except raise:
(16)                        b(): c()
(16)                        c(): raise Exception( 'c() failed')
```

```
    Exception: c() failed
Handling of chained exceptions:
    >>> def e():
    ...     raise Exception( 'e(): deliberate error')
    >>> def d():
    ...     e()
    >>> def c():
    ...     try:
    ...         d()
    ...     except Exception as e:
    ...         raise Exception( 'c: d() failed') from e
    >>> def b():
    ...     try:
    ...         c()
    ...     except Exception as e:
    ...         exception_info( file=sys.stdout, chain=g_chain, _filelinefn=0)
    >>> def a():
    ...     b()
    With `chain=True` (the default), we output low-level exceptions
    first, matching the behaviour of `traceback.*` functions:
        >>> g_chain = True
        >>> a() # doctest: +REPORT_UDIFF +ELLIPSIS
        Traceback (most recent call last):
            c(): d()
            d(): e()
            e(): raise Exception( 'e(): deliberate error')
        Exception: e(): deliberate error
        <BLANKLINE>
        The above exception was the direct cause of the following exception:

        Traceback (most recent call last):
            ...
            <module>(): a() # doctest: +REPORT_UDIFF +ELLIPSIS
            a(): b()
            b(): exception_info( file=sys.stdout, chain=g_chain, _filelinefn=0)

            ^except raise:
            b(): c()
            c(): raise Exception( 'c: d() failed') from e
        Exception: c: d() failed
    With `chain='because'`, we output high-level exceptions first:
        >>> g_chain = 'because'
        >>> a() # doctest: +REPORT_UDIFF +ELLIPSIS
        Traceback (most recent call last):
            ...
            <module>(): a() # doctest: +REPORT_UDIFF +ELLIPSIS
            a(): b()
            b(): exception_info( file=sys.stdout, chain=g_chain, _filelinefn=0)

            ^except raise:
            b(): c()
            c(): raise Exception( 'c: d() failed') from e
        Exception: c: d() failed
        <BLANKLINE>
        Because:
        Traceback (most recent call last):
            c(): d()
            d(): e()
            e(): raise Exception( 'e(): deliberate error')
        Exception: e(): deliberate error
Show current backtrace by passing `exception_or_traceback=None`:
    >>> def c():
    ...     exception_info( None, file=sys.stdout, _filelinefn=0)
    >>> def b():
    ...     return c()
    >>> def a():
    ...     return b()
    >>> a() # doctest: +REPORT_UDIFF +ELLIPSIS
```

```
41393: (12)                     Traceback (most recent call last):
41394: (16)                         ...
41395: (16)                         <module>(): a() # doctest: +REPORT_UDIFF +ELLIPSIS
41396: (16)                         a(): return b()
41397: (16)                         b(): return c()
41398: (16)                         c(): exception_info( None, file=sys.stdout, _filelinefn=0)
41399: (8)                  Show an exception's `.__traceback__` backtrace:
41400: (12)                     >>> def c():
41401: (12)                     ...     raise Exception( 'foo') # raise
41402: (12)                     >>> def b():
41403: (12)                     ...     return c()  # call c
41404: (12)                     >>> def a():
41405: (12)                     ...     try:
41406: (12)                     ...         b() # call b
41407: (12)                     ...     except Exception as e:
41408: (12)                     ...         exception_info( e.__traceback__, file=sys.stdout,
_filelinefn=0)
41409: (12)                     >>> a() # doctest: +REPORT_UDIFF +ELLIPSIS
41410: (12)                     Traceback (most recent call last):
41411: (16)                         ...
41412: (16)                         a(): b() # call b
41413: (16)                         b(): return c()  # call c
41414: (16)                         c(): raise Exception( 'foo') # raise
41415: (4)              '''
41416: (4)              if isinstance( exception_or_traceback, (types.TracebackType,
inspect.FrameInfo)):
41417: (8)                  exception = None
41418: (8)                  tb = exception_or_traceback
41419: (4)              elif isinstance( exception_or_traceback, BaseException):
41420: (8)                  exception = exception_or_traceback
41421: (8)                  tb = None
41422: (4)              elif exception_or_traceback is None:
41423: (8)                  _, exception, tb = sys.exc_info()
41424: (8)                  tb = None if exception else inspect.stack()[1:]
41425: (4)              else:
41426: (8)                  assert 0, f'Unrecognised exception_or_traceback type:
{type(exception_or_traceback)}'
41427: (4)              if file == 'return':
41428: (8)                  out = io.StringIO()
41429: (4)              else:
41430: (8)                  out = file if file else sys.stderr
41431: (4)              def do_chain( exception):
41432: (8)                  exception_info(
41433: (16)                         exception,
41434: (16)                         limit,
41435: (16)                         out,
41436: (16)                         chain,
41437: (16)                         outer=False,
41438: (16)                         show_exception_type=show_exception_type,
41439: (16)                         _filelinefn=_filelinefn,
41440: (16)                         )
41441: (4)              if exception and chain and chain != 'because' and chain != 'because-
compact':
41442: (8)                  if exception.__cause__:
41443: (12)                     do_chain( exception.__cause__)
41444: (12)                     out.write( '\nThe above exception was the direct cause of the
following exception:\n')
41445: (8)                  elif exception.__context__:
41446: (12)                     do_chain( exception.__context__)
41447: (12)                     out.write( '\nDuring handling of the above exception, another
exception occurred:\n')
41448: (4)              cwd = os.getcwd() + os.sep
41449: (4)              def output_frames( frames, reverse, limit):
41450: (8)                  if limit == 0:
41451: (12)                     return
41452: (8)                  if reverse:
41453: (12)                     assert isinstance( frames, list)
41454: (12)                     frames = reversed( frames)
41455: (8)                  if limit is not None:
```

```
41456: (12)                     frames = list( frames)
41457: (12)                     frames = frames[ -limit:]
41458: (8)                  for frame in frames:
41459: (12)                     f, filename, line, fnname, text, index = frame
41460: (12)                     text = text[0].strip() if text else ''
41461: (12)                     if filename.startswith( cwd):
41462: (16)                         filename = filename[ len(cwd):]
41463: (12)                     if filename.startswith( f'.{os.sep}'):
41464: (16)                         filename = filename[ 2:]
41465: (12)                     if _filelinefn:
41466: (16)                         out.write( f'    {filename}:{line}:{fnname}(): {text}\n')
41467: (12)                     else:
41468: (16)                         out.write( f'    {fnname}(): {text}\n')
41469: (4)              if limit != 0:
41470: (8)                  out.write( 'Traceback (most recent call last):\n')
41471: (8)                  if exception:
41472: (12)                     tb = exception.__traceback__
41473: (12)                     assert tb
41474: (12)                     if outer:
41475: (16)                         output_frames( inspect.getouterframes( tb.tb_frame),
reverse=True, limit=limit)
41476: (16)                         out.write( '    ^except raise:\n')
41477: (12)                     limit2 = 0 if limit == 0 else None
41478: (12)                     output_frames( inspect.getinnerframes( tb), reverse=False,
limit=limit2)
41479: (8)                  else:
41480: (12)                     if not isinstance( tb, list):
41481: (16)                         inner = inspect.getinnerframes(tb)
41482: (16)                         outer = inspect.getouterframes(tb.tb_frame)
41483: (16)                         tb = outer + inner
41484: (16)                         tb.reverse()
41485: (12)                     output_frames( tb, reverse=True, limit=limit)
41486: (4)              if exception:
41487: (8)                  if callable(show_exception_type):
41488: (12)                     show_exception_type2 = show_exception_type( exception)
41489: (8)                  else:
41490: (12)                     show_exception_type2 = show_exception_type
41491: (8)                  if show_exception_type2:
41492: (12)                     lines = traceback.format_exception_only( type(exception),
exception)
41493: (12)                     for line in lines:
41494: (16)                         out.write( line)
41495: (8)                  else:
41496: (12)                     out.write( str( exception) + '\n')
41497: (4)              if exception and (chain == 'because' or chain == 'because-compact'):
41498: (8)                  pre, post = ('\n', '\n') if chain == 'because' else ('', ' ')
41499: (8)                  if exception.__cause__:
41500: (12)                     out.write( f'{pre}Because:{post}')
41501: (12)                     do_chain( exception.__cause__)
41502: (8)                  elif exception.__context__:
41503: (12)                     out.write( f'{pre}Because: error occurred handling this exception:
{post}')
41504: (12)                     do_chain( exception.__context__)
41505: (4)              if file == 'return':
41506: (8)                  return out.getvalue()
41507: (0)          def ll_fz_bidi_fragment_text(text, textlen, callback, arg, flags):
41508: (4)              """
41509: (4)              Wrapper for out-params of fz_bidi_fragment_text().
41510: (4)              Returns: ::fz_bidi_direction baseDir
41511: (4)              """
41512: (4)              outparams = ll_fz_bidi_fragment_text_outparams()
41513: (4)              ret = ll_fz_bidi_fragment_text_outparams_fn(text, textlen, callback, arg,
flags, outparams)
41514: (4)              return outparams.baseDir
41515: (0)          def fz_bidi_fragment_text_outparams_fn(text, textlen, callback, arg, flags):
41516: (4)              """
41517: (4)              Class-aware helper for out-params of fz_bidi_fragment_text()
[fz_bidi_fragment_text()].
41518: (4)              """
```

```
41519: (4)              baseDir = ll_fz_bidi_fragment_text(text, textlen, callback, arg, flags)
41520: (4)              return baseDir
41521: (0)          fz_bidi_fragment_text = fz_bidi_fragment_text_outparams_fn
41522: (0)          def ll_fz_bitmap_details(bitmap):
41523: (4)              """
41524: (4)              Wrapper for out-params of fz_bitmap_details().
41525: (4)              Returns: int w, int h, int n, int stride
41526: (4)              """
41527: (4)              outparams = ll_fz_bitmap_details_outparams()
41528: (4)              ret = ll_fz_bitmap_details_outparams_fn(bitmap, outparams)
41529: (4)              return outparams.w, outparams.h, outparams.n, outparams.stride
41530: (0)          def fz_bitmap_details_outparams_fn(bitmap):
41531: (4)              """
41532: (4)              Class-aware helper for out-params of fz_bitmap_details()
[fz_bitmap_details()].
41533: (4)              """
41534: (4)              w, h, n, stride = ll_fz_bitmap_details(bitmap.m_internal)
41535: (4)              return w, h, n, stride
41536: (0)          fz_bitmap_details = fz_bitmap_details_outparams_fn
41537: (0)          def ll_fz_buffer_extract(buf):
41538: (4)              """
41539: (4)              Wrapper for out-params of fz_buffer_extract().
41540: (4)              Returns: size_t, unsigned char *data
41541: (4)              """
41542: (4)              outparams = ll_fz_buffer_extract_outparams()
41543: (4)              ret = ll_fz_buffer_extract_outparams_fn(buf, outparams)
41544: (4)              return ret, outparams.data
41545: (0)          def fz_buffer_extract_outparams_fn(buf):
41546: (4)              """
41547: (4)              Class-aware helper for out-params of fz_buffer_extract()
[fz_buffer_extract()].
41548: (4)              """
41549: (4)              ret, data = ll_fz_buffer_extract(buf.m_internal)
41550: (4)              return ret, data
41551: (0)          fz_buffer_extract = fz_buffer_extract_outparams_fn
41552: (0)          def ll_fz_buffer_storage(buf):
41553: (4)              """
41554: (4)              Wrapper for out-params of fz_buffer_storage().
41555: (4)              Returns: size_t, unsigned char *datap
41556: (4)              """
41557: (4)              outparams = ll_fz_buffer_storage_outparams()
41558: (4)              ret = ll_fz_buffer_storage_outparams_fn(buf, outparams)
41559: (4)              return ret, outparams.datap
41560: (0)          def fz_buffer_storage_outparams_fn(buf):
41561: (4)              """
41562: (4)              Class-aware helper for out-params of fz_buffer_storage()
[fz_buffer_storage()].
41563: (4)              """
41564: (4)              ret, datap = ll_fz_buffer_storage(buf.m_internal)
41565: (4)              return ret, datap
41566: (0)          fz_buffer_storage = fz_buffer_storage_outparams_fn
41567: (0)          def ll_fz_chartorune(str):
41568: (4)              """
41569: (4)              Wrapper for out-params of fz_chartorune().
41570: (4)              Returns: int, int rune
41571: (4)              """
41572: (4)              outparams = ll_fz_chartorune_outparams()
41573: (4)              ret = ll_fz_chartorune_outparams_fn(str, outparams)
41574: (4)              return ret, outparams.rune
41575: (0)          def fz_chartorune_outparams_fn(str):
41576: (4)              """
41577: (4)              Class-aware helper for out-params of fz_chartorune() [fz_chartorune()].
41578: (4)              """
41579: (4)              ret, rune = ll_fz_chartorune(str)
41580: (4)              return ret, rune
41581: (0)          fz_chartorune = fz_chartorune_outparams_fn
41582: (0)          def ll_fz_clamp_color(cs, in_):
41583: (4)              """
41584: (4)              Wrapper for out-params of fz_clamp_color().
```

```
41585: (4)              Returns: float out
41586: (4)              """
41587: (4)              outparams = ll_fz_clamp_color_outparams()
41588: (4)              ret = ll_fz_clamp_color_outparams_fn(cs, in_, outparams)
41589: (4)              return outparams.out
41590: (0)          def fz_clamp_color_outparams_fn(cs, in_):
41591: (4)              """
41592: (4)              Class-aware helper for out-params of fz_clamp_color() [fz_clamp_color()].
41593: (4)              """
41594: (4)              out = ll_fz_clamp_color(cs.m_internal, in_)
41595: (4)              return out
41596: (0)          fz_clamp_color = fz_clamp_color_outparams_fn
41597: (0)          def ll_fz_convert_color(ss, sv, ds, is_, params):
41598: (4)              """
41599: (4)              Wrapper for out-params of fz_convert_color().
41600: (4)              Returns: float dv
41601: (4)              """
41602: (4)              outparams = ll_fz_convert_color_outparams()
41603: (4)              ret = ll_fz_convert_color_outparams_fn(ss, sv, ds, is_, params, outparams)
41604: (4)              return outparams.dv
41605: (0)          def fz_convert_color_outparams_fn(ss, sv, ds, is_, params):
41606: (4)              """
41607: (4)              Class-aware helper for out-params of fz_convert_color()
[fz_convert_color()].
41608: (4)              """
41609: (4)              dv = ll_fz_convert_color(ss.m_internal, sv, ds.m_internal, is_.m_internal,
params.internal())
41610: (4)              return dv
41611: (0)          fz_convert_color = fz_convert_color_outparams_fn
41612: (0)          def ll_fz_convert_error():
41613: (4)              """
41614: (4)              Wrapper for out-params of fz_convert_error().
41615: (4)              Returns: const char *, int code
41616: (4)              """
41617: (4)              outparams = ll_fz_convert_error_outparams()
41618: (4)              ret = ll_fz_convert_error_outparams_fn(outparams)
41619: (4)              return ret, outparams.code
41620: (0)          def fz_convert_error_outparams_fn():
41621: (4)              """
41622: (4)              Class-aware helper for out-params of fz_convert_error()
[fz_convert_error()].
41623: (4)              """
41624: (4)              ret, code = ll_fz_convert_error()
41625: (4)              return ret, code
41626: (0)          fz_convert_error = fz_convert_error_outparams_fn
41627: (0)          def ll_fz_convert_separation_colors(src_cs, src_color, dst_seps, dst_cs,
color_params):
41628: (4)              """
41629: (4)              Wrapper for out-params of fz_convert_separation_colors().
41630: (4)              Returns: float dst_color
41631: (4)              """
41632: (4)              outparams = ll_fz_convert_separation_colors_outparams()
41633: (4)              ret = ll_fz_convert_separation_colors_outparams_fn(src_cs, src_color,
dst_seps, dst_cs, color_params, outparams)
41634: (4)              return outparams.dst_color
41635: (0)          def fz_convert_separation_colors_outparams_fn(src_cs, src_color, dst_seps,
dst_cs, color_params):
41636: (4)              """
41637: (4)              Class-aware helper for out-params of fz_convert_separation_colors()
[fz_convert_separation_colors()].
41638: (4)              """
41639: (4)              dst_color = ll_fz_convert_separation_colors(src_cs.m_internal, src_color,
dst_seps.m_internal, dst_cs.m_internal, color_params.internal())
41640: (4)              return dst_color
41641: (0)          fz_convert_separation_colors = fz_convert_separation_colors_outparams_fn
41642: (0)          def ll_fz_decomp_image_from_stream(stm, image, subarea, indexed, l2factor):
41643: (4)              """
41644: (4)              Wrapper for out-params of fz_decomp_image_from_stream().
41645: (4)              Returns: fz_pixmap *, int l2extra
```

```
41646: (4)              """
41647: (4)              outparams = ll_fz_decomp_image_from_stream_outparams()
41648: (4)              ret = ll_fz_decomp_image_from_stream_outparams_fn(stm, image, subarea,
indexed, l2factor, outparams)
41649: (4)              return ret, outparams.l2extra
41650: (0)          def fz_decomp_image_from_stream_outparams_fn(stm, image, subarea, indexed,
l2factor):
41651: (4)              """
41652: (4)              Class-aware helper for out-params of fz_decomp_image_from_stream()
[fz_decomp_image_from_stream()].
41653: (4)              """
41654: (4)              ret, l2extra = ll_fz_decomp_image_from_stream(stm.m_internal,
image.m_internal, subarea.internal(), indexed, l2factor)
41655: (4)              return FzPixmap(ret), l2extra
41656: (0)          fz_decomp_image_from_stream = fz_decomp_image_from_stream_outparams_fn
41657: (0)          def ll_fz_deflate(dest, source, source_length, level):
41658: (4)              """
41659: (4)              Wrapper for out-params of fz_deflate().
41660: (4)              Returns: size_t compressed_length
41661: (4)              """
41662: (4)              outparams = ll_fz_deflate_outparams()
41663: (4)              ret = ll_fz_deflate_outparams_fn(dest, source, source_length, level,
outparams)
41664: (4)              return outparams.compressed_length
41665: (0)          def fz_deflate_outparams_fn(dest, source, source_length, level):
41666: (4)              """
41667: (4)              Class-aware helper for out-params of fz_deflate() [fz_deflate()].
41668: (4)              """
41669: (4)              compressed_length = ll_fz_deflate(dest, source, source_length, level)
41670: (4)              return compressed_length
41671: (0)          fz_deflate = fz_deflate_outparams_fn
41672: (0)          def ll_fz_dom_get_attribute(elt, i):
41673: (4)              """
41674: (4)              Wrapper for out-params of fz_dom_get_attribute().
41675: (4)              Returns: const char *, const char *att
41676: (4)              """
41677: (4)              outparams = ll_fz_dom_get_attribute_outparams()
41678: (4)              ret = ll_fz_dom_get_attribute_outparams_fn(elt, i, outparams)
41679: (4)              return ret, outparams.att
41680: (0)          def fz_dom_get_attribute_outparams_fn(elt, i):
41681: (4)              """
41682: (4)              Class-aware helper for out-params of fz_dom_get_attribute()
[fz_dom_get_attribute()].
41683: (4)              """
41684: (4)              ret, att = ll_fz_dom_get_attribute(elt.m_internal, i)
41685: (4)              return ret, att
41686: (0)          fz_dom_get_attribute = fz_dom_get_attribute_outparams_fn
41687: (0)          def ll_fz_drop_imp(p):
41688: (4)              """
41689: (4)              Wrapper for out-params of fz_drop_imp().
41690: (4)              Returns: int, int refs
41691: (4)              """
41692: (4)              outparams = ll_fz_drop_imp_outparams()
41693: (4)              ret = ll_fz_drop_imp_outparams_fn(p, outparams)
41694: (4)              return ret, outparams.refs
41695: (0)          def ll_fz_drop_imp16(p):
41696: (4)              """
41697: (4)              Wrapper for out-params of fz_drop_imp16().
41698: (4)              Returns: int, int16_t refs
41699: (4)              """
41700: (4)              outparams = ll_fz_drop_imp16_outparams()
41701: (4)              ret = ll_fz_drop_imp16_outparams_fn(p, outparams)
41702: (4)              return ret, outparams.refs
41703: (0)          def ll_fz_encode_character_with_fallback(font, unicode, script, language):
41704: (4)              """
41705: (4)              Wrapper for out-params of fz_encode_character_with_fallback().
41706: (4)              Returns: int, ::fz_font *out_font
41707: (4)              """
41708: (4)              outparams = ll_fz_encode_character_with_fallback_outparams()
```

```
41709: (4)              ret = ll_fz_encode_character_with_fallback_outparams_fn(font, unicode,
script, language, outparams)
41710: (4)              return ret, outparams.out_font
41711: (0)          def fz_encode_character_with_fallback_outparams_fn(font, unicode, script,
language):
41712: (4)              """
41713: (4)              Class-aware helper for out-params of fz_encode_character_with_fallback()
[fz_encode_character_with_fallback()].
41714: (4)              """
41715: (4)              ret, out_font = ll_fz_encode_character_with_fallback(font.m_internal,
unicode, script, language)
41716: (4)              return ret, FzFont(ll_fz_keep_font( out_font))
41717: (0)          fz_encode_character_with_fallback =
fz_encode_character_with_fallback_outparams_fn
41718: (0)          def ll_fz_error_callback():
41719: (4)              """
41720: (4)              Wrapper for out-params of fz_error_callback().
41721: (4)              Returns: fz_error_cb *, void *user
41722: (4)              """
41723: (4)              outparams = ll_fz_error_callback_outparams()
41724: (4)              ret = ll_fz_error_callback_outparams_fn(outparams)
41725: (4)              return ret, outparams.user
41726: (0)          def fz_error_callback_outparams_fn():
41727: (4)              """
41728: (4)              Class-aware helper for out-params of fz_error_callback()
[fz_error_callback()].
41729: (4)              """
41730: (4)              ret, user = ll_fz_error_callback()
41731: (4)              return ret, user
41732: (0)          fz_error_callback = fz_error_callback_outparams_fn
41733: (0)          def ll_fz_eval_function(func, in_, inlen, outlen):
41734: (4)              """
41735: (4)              Wrapper for out-params of fz_eval_function().
41736: (4)              Returns: float out
41737: (4)              """
41738: (4)              outparams = ll_fz_eval_function_outparams()
41739: (4)              ret = ll_fz_eval_function_outparams_fn(func, in_, inlen, outlen,
outparams)
41740: (4)              return outparams.out
41741: (0)          def fz_eval_function_outparams_fn(func, in_, inlen, outlen):
41742: (4)              """
41743: (4)              Class-aware helper for out-params of fz_eval_function()
[fz_eval_function()].
41744: (4)              """
41745: (4)              out = ll_fz_eval_function(func.m_internal, in_, inlen, outlen)
41746: (4)              return out
41747: (0)          fz_eval_function = fz_eval_function_outparams_fn
41748: (0)          def ll_fz_fill_pixmap_with_color(pix, colorspace, color_params):
41749: (4)              """
41750: (4)              Wrapper for out-params of fz_fill_pixmap_with_color().
41751: (4)              Returns: float color
41752: (4)              """
41753: (4)              outparams = ll_fz_fill_pixmap_with_color_outparams()
41754: (4)              ret = ll_fz_fill_pixmap_with_color_outparams_fn(pix, colorspace,
color_params, outparams)
41755: (4)              return outparams.color
41756: (0)          def fz_fill_pixmap_with_color_outparams_fn(pix, colorspace, color_params):
41757: (4)              """
41758: (4)              Class-aware helper for out-params of fz_fill_pixmap_with_color()
[fz_fill_pixmap_with_color()].
41759: (4)              """
41760: (4)              color = ll_fz_fill_pixmap_with_color(pix.m_internal,
colorspace.m_internal, color_params.internal())
41761: (4)              return color
41762: (0)          fz_fill_pixmap_with_color = fz_fill_pixmap_with_color_outparams_fn
41763: (0)          def ll_fz_get_pixmap_from_image(image, subarea, ctm):
41764: (4)              """
41765: (4)              Wrapper for out-params of fz_get_pixmap_from_image().
41766: (4)              Returns: fz_pixmap *, int w, int h
```

```
41767: (4)             """
41768: (4)             outparams = ll_fz_get_pixmap_from_image_outparams()
41769: (4)             ret = ll_fz_get_pixmap_from_image_outparams_fn(image, subarea, ctm,
outparams)
41770: (4)             return ret, outparams.w, outparams.h
41771: (0)         def fz_get_pixmap_from_image_outparams_fn(image, subarea, ctm):
41772: (4)             """
41773: (4)             Class-aware helper for out-params of fz_get_pixmap_from_image()
[fz_get_pixmap_from_image()].
41774: (4)             """
41775: (4)             ret, w, h = ll_fz_get_pixmap_from_image(image.m_internal,
subarea.internal(), ctm.internal())
41776: (4)             return FzPixmap(ret), w, h
41777: (0)         fz_get_pixmap_from_image = fz_get_pixmap_from_image_outparams_fn
41778: (0)         def ll_fz_getopt(nargc, ostr):
41779: (4)             """
41780: (4)             Wrapper for out-params of fz_getopt().
41781: (4)             Returns: int, char *nargv
41782: (4)             """
41783: (4)             outparams = ll_fz_getopt_outparams()
41784: (4)             ret = ll_fz_getopt_outparams_fn(nargc, ostr, outparams)
41785: (4)             return ret, outparams.nargv
41786: (0)         def fz_getopt_outparams_fn(nargc, ostr):
41787: (4)             """
41788: (4)             Class-aware helper for out-params of fz_getopt() [fz_getopt()].
41789: (4)             """
41790: (4)             ret, nargv = ll_fz_getopt(nargc, ostr)
41791: (4)             return ret, nargv
41792: (0)         fz_getopt = fz_getopt_outparams_fn
41793: (0)         def ll_fz_getopt_long(nargc, ostr, longopts):
41794: (4)             """
41795: (4)             Wrapper for out-params of fz_getopt_long().
41796: (4)             Returns: int, char *nargv
41797: (4)             """
41798: (4)             outparams = ll_fz_getopt_long_outparams()
41799: (4)             ret = ll_fz_getopt_long_outparams_fn(nargc, ostr, longopts, outparams)
41800: (4)             return ret, outparams.nargv
41801: (0)         def fz_getopt_long_outparams_fn(nargc, ostr, longopts):
41802: (4)             """
41803: (4)             Class-aware helper for out-params of fz_getopt_long() [fz_getopt_long()].
41804: (4)             """
41805: (4)             ret, nargv = ll_fz_getopt_long(nargc, ostr, longopts.m_internal)
41806: (4)             return ret, nargv
41807: (0)         fz_getopt_long = fz_getopt_long_outparams_fn
41808: (0)         def ll_fz_grisu(f, s):
41809: (4)             """
41810: (4)             Wrapper for out-params of fz_grisu().
41811: (4)             Returns: int, int exp
41812: (4)             """
41813: (4)             outparams = ll_fz_grisu_outparams()
41814: (4)             ret = ll_fz_grisu_outparams_fn(f, s, outparams)
41815: (4)             return ret, outparams.exp
41816: (0)         def fz_grisu_outparams_fn(f, s):
41817: (4)             """
41818: (4)             Class-aware helper for out-params of fz_grisu() [fz_grisu()].
41819: (4)             """
41820: (4)             ret, exp = ll_fz_grisu(f, s)
41821: (4)             return ret, exp
41822: (0)         fz_grisu = fz_grisu_outparams_fn
41823: (0)         def ll_fz_has_option(opts, key):
41824: (4)             """
41825: (4)             Wrapper for out-params of fz_has_option().
41826: (4)             Returns: int, const char *val
41827: (4)             """
41828: (4)             outparams = ll_fz_has_option_outparams()
41829: (4)             ret = ll_fz_has_option_outparams_fn(opts, key, outparams)
41830: (4)             return ret, outparams.val
41831: (0)         def fz_has_option_outparams_fn(opts, key):
41832: (4)             """
```

```
41833: (4)              Class-aware helper for out-params of fz_has_option() [fz_has_option()].
41834: (4)              """
41835: (4)              ret, val = ll_fz_has_option(opts, key)
41836: (4)              return ret, val
41837: (0)          fz_has_option = fz_has_option_outparams_fn
41838: (0)          def ll_fz_image_resolution(image):
41839: (4)              """
41840: (4)              Wrapper for out-params of fz_image_resolution().
41841: (4)              Returns: int xres, int yres
41842: (4)              """
41843: (4)              outparams = ll_fz_image_resolution_outparams()
41844: (4)              ret = ll_fz_image_resolution_outparams_fn(image, outparams)
41845: (4)              return outparams.xres, outparams.yres
41846: (0)          def fz_image_resolution_outparams_fn(image):
41847: (4)              """
41848: (4)              Class-aware helper for out-params of fz_image_resolution()
[fz_image_resolution()].
41849: (4)              """
41850: (4)              xres, yres = ll_fz_image_resolution(image.m_internal)
41851: (4)              return xres, yres
41852: (0)          fz_image_resolution = fz_image_resolution_outparams_fn
41853: (0)          def ll_fz_keep_imp(p):
41854: (4)              """
41855: (4)              Wrapper for out-params of fz_keep_imp().
41856: (4)              Returns: void *, int refs
41857: (4)              """
41858: (4)              outparams = ll_fz_keep_imp_outparams()
41859: (4)              ret = ll_fz_keep_imp_outparams_fn(p, outparams)
41860: (4)              return ret, outparams.refs
41861: (0)          def ll_fz_keep_imp16(p):
41862: (4)              """
41863: (4)              Wrapper for out-params of fz_keep_imp16().
41864: (4)              Returns: void *, int16_t refs
41865: (4)              """
41866: (4)              outparams = ll_fz_keep_imp16_outparams()
41867: (4)              ret = ll_fz_keep_imp16_outparams_fn(p, outparams)
41868: (4)              return ret, outparams.refs
41869: (0)          def ll_fz_keep_imp_locked(p):
41870: (4)              """
41871: (4)              Wrapper for out-params of fz_keep_imp_locked().
41872: (4)              Returns: void *, int refs
41873: (4)              """
41874: (4)              outparams = ll_fz_keep_imp_locked_outparams()
41875: (4)              ret = ll_fz_keep_imp_locked_outparams_fn(p, outparams)
41876: (4)              return ret, outparams.refs
41877: (0)          def ll_fz_lookup_base14_font(name):
41878: (4)              """
41879: (4)              Wrapper for out-params of fz_lookup_base14_font().
41880: (4)              Returns: const unsigned char *, int len
41881: (4)              """
41882: (4)              outparams = ll_fz_lookup_base14_font_outparams()
41883: (4)              ret = ll_fz_lookup_base14_font_outparams_fn(name, outparams)
41884: (4)              return ret, outparams.len
41885: (0)          def fz_lookup_base14_font_outparams_fn(name):
41886: (4)              """
41887: (4)              Class-aware helper for out-params of fz_lookup_base14_font()
[fz_lookup_base14_font()].
41888: (4)              """
41889: (4)              ret, len = ll_fz_lookup_base14_font(name)
41890: (4)              return ret, len
41891: (0)          fz_lookup_base14_font = fz_lookup_base14_font_outparams_fn
41892: (0)          def ll_fz_lookup_builtin_font(name, bold, italic):
41893: (4)              """
41894: (4)              Wrapper for out-params of fz_lookup_builtin_font().
41895: (4)              Returns: const unsigned char *, int len
41896: (4)              """
41897: (4)              outparams = ll_fz_lookup_builtin_font_outparams()
41898: (4)              ret = ll_fz_lookup_builtin_font_outparams_fn(name, bold, italic,
outparams)
```

```
41899: (4)              return ret, outparams.len
41900: (0)          def fz_lookup_builtin_font_outparams_fn(name, bold, italic):
41901: (4)              """
41902: (4)              Class-aware helper for out-params of fz_lookup_builtin_font()
[fz_lookup_builtin_font()].
41903: (4)              """
41904: (4)              ret, len = ll_fz_lookup_builtin_font(name, bold, italic)
41905: (4)              return ret, len
41906: (0)          fz_lookup_builtin_font = fz_lookup_builtin_font_outparams_fn
41907: (0)          def ll_fz_lookup_cjk_font(ordering):
41908: (4)              """
41909: (4)              Wrapper for out-params of fz_lookup_cjk_font().
41910: (4)              Returns: const unsigned char *, int len, int index
41911: (4)              """
41912: (4)              outparams = ll_fz_lookup_cjk_font_outparams()
41913: (4)              ret = ll_fz_lookup_cjk_font_outparams_fn(ordering, outparams)
41914: (4)              return ret, outparams.len, outparams.index
41915: (0)          def fz_lookup_cjk_font_outparams_fn(ordering):
41916: (4)              """
41917: (4)              Class-aware helper for out-params of fz_lookup_cjk_font()
[fz_lookup_cjk_font()].
41918: (4)              """
41919: (4)              ret, len, index = ll_fz_lookup_cjk_font(ordering)
41920: (4)              return ret, len, index
41921: (0)          fz_lookup_cjk_font = fz_lookup_cjk_font_outparams_fn
41922: (0)          def ll_fz_lookup_cjk_font_by_language(lang):
41923: (4)              """
41924: (4)              Wrapper for out-params of fz_lookup_cjk_font_by_language().
41925: (4)              Returns: const unsigned char *, int len, int subfont
41926: (4)              """
41927: (4)              outparams = ll_fz_lookup_cjk_font_by_language_outparams()
41928: (4)              ret = ll_fz_lookup_cjk_font_by_language_outparams_fn(lang, outparams)
41929: (4)              return ret, outparams.len, outparams.subfont
41930: (0)          def fz_lookup_cjk_font_by_language_outparams_fn(lang):
41931: (4)              """
41932: (4)              Class-aware helper for out-params of fz_lookup_cjk_font_by_language()
[fz_lookup_cjk_font_by_language()].
41933: (4)              """
41934: (4)              ret, len, subfont = ll_fz_lookup_cjk_font_by_language(lang)
41935: (4)              return ret, len, subfont
41936: (0)          fz_lookup_cjk_font_by_language = fz_lookup_cjk_font_by_language_outparams_fn
41937: (0)          def ll_fz_lookup_noto_boxes_font():
41938: (4)              """
41939: (4)              Wrapper for out-params of fz_lookup_noto_boxes_font().
41940: (4)              Returns: const unsigned char *, int len
41941: (4)              """
41942: (4)              outparams = ll_fz_lookup_noto_boxes_font_outparams()
41943: (4)              ret = ll_fz_lookup_noto_boxes_font_outparams_fn(outparams)
41944: (4)              return ret, outparams.len
41945: (0)          def fz_lookup_noto_boxes_font_outparams_fn():
41946: (4)              """
41947: (4)              Class-aware helper for out-params of fz_lookup_noto_boxes_font()
[fz_lookup_noto_boxes_font()].
41948: (4)              """
41949: (4)              ret, len = ll_fz_lookup_noto_boxes_font()
41950: (4)              return ret, len
41951: (0)          fz_lookup_noto_boxes_font = fz_lookup_noto_boxes_font_outparams_fn
41952: (0)          def ll_fz_lookup_noto_emoji_font():
41953: (4)              """
41954: (4)              Wrapper for out-params of fz_lookup_noto_emoji_font().
41955: (4)              Returns: const unsigned char *, int len
41956: (4)              """
41957: (4)              outparams = ll_fz_lookup_noto_emoji_font_outparams()
41958: (4)              ret = ll_fz_lookup_noto_emoji_font_outparams_fn(outparams)
41959: (4)              return ret, outparams.len
41960: (0)          def fz_lookup_noto_emoji_font_outparams_fn():
41961: (4)              """
41962: (4)              Class-aware helper for out-params of fz_lookup_noto_emoji_font()
[fz_lookup_noto_emoji_font()].
```

```
41963: (4)              """
41964: (4)              ret, len = ll_fz_lookup_noto_emoji_font()
41965: (4)              return ret, len
41966: (0)          fz_lookup_noto_emoji_font = fz_lookup_noto_emoji_font_outparams_fn
41967: (0)          def ll_fz_lookup_noto_font(script, lang):
41968: (4)              """
41969: (4)              Wrapper for out-params of fz_lookup_noto_font().
41970: (4)              Returns: const unsigned char *, int len, int subfont
41971: (4)              """
41972: (4)              outparams = ll_fz_lookup_noto_font_outparams()
41973: (4)              ret = ll_fz_lookup_noto_font_outparams_fn(script, lang, outparams)
41974: (4)              return ret, outparams.len, outparams.subfont
41975: (0)          def fz_lookup_noto_font_outparams_fn(script, lang):
41976: (4)              """
41977: (4)              Class-aware helper for out-params of fz_lookup_noto_font()
[fz_lookup_noto_font()].
41978: (4)              """
41979: (4)              ret, len, subfont = ll_fz_lookup_noto_font(script, lang)
41980: (4)              return ret, len, subfont
41981: (0)          fz_lookup_noto_font = fz_lookup_noto_font_outparams_fn
41982: (0)          def ll_fz_lookup_noto_math_font():
41983: (4)              """
41984: (4)              Wrapper for out-params of fz_lookup_noto_math_font().
41985: (4)              Returns: const unsigned char *, int len
41986: (4)              """
41987: (4)              outparams = ll_fz_lookup_noto_math_font_outparams()
41988: (4)              ret = ll_fz_lookup_noto_math_font_outparams_fn(outparams)
41989: (4)              return ret, outparams.len
41990: (0)          def fz_lookup_noto_math_font_outparams_fn():
41991: (4)              """
41992: (4)              Class-aware helper for out-params of fz_lookup_noto_math_font()
[fz_lookup_noto_math_font()].
41993: (4)              """
41994: (4)              ret, len = ll_fz_lookup_noto_math_font()
41995: (4)              return ret, len
41996: (0)          fz_lookup_noto_math_font = fz_lookup_noto_math_font_outparams_fn
41997: (0)          def ll_fz_lookup_noto_music_font():
41998: (4)              """
41999: (4)              Wrapper for out-params of fz_lookup_noto_music_font().
42000: (4)              Returns: const unsigned char *, int len
42001: (4)              """
42002: (4)              outparams = ll_fz_lookup_noto_music_font_outparams()
42003: (4)              ret = ll_fz_lookup_noto_music_font_outparams_fn(outparams)
42004: (4)              return ret, outparams.len
42005: (0)          def fz_lookup_noto_music_font_outparams_fn():
42006: (4)              """
42007: (4)              Class-aware helper for out-params of fz_lookup_noto_music_font()
[fz_lookup_noto_music_font()].
42008: (4)              """
42009: (4)              ret, len = ll_fz_lookup_noto_music_font()
42010: (4)              return ret, len
42011: (0)          fz_lookup_noto_music_font = fz_lookup_noto_music_font_outparams_fn
42012: (0)          def ll_fz_lookup_noto_symbol1_font():
42013: (4)              """
42014: (4)              Wrapper for out-params of fz_lookup_noto_symbol1_font().
42015: (4)              Returns: const unsigned char *, int len
42016: (4)              """
42017: (4)              outparams = ll_fz_lookup_noto_symbol1_font_outparams()
42018: (4)              ret = ll_fz_lookup_noto_symbol1_font_outparams_fn(outparams)
42019: (4)              return ret, outparams.len
42020: (0)          def fz_lookup_noto_symbol1_font_outparams_fn():
42021: (4)              """
42022: (4)              Class-aware helper for out-params of fz_lookup_noto_symbol1_font()
[fz_lookup_noto_symbol1_font()].
42023: (4)              """
42024: (4)              ret, len = ll_fz_lookup_noto_symbol1_font()
42025: (4)              return ret, len
42026: (0)          fz_lookup_noto_symbol1_font = fz_lookup_noto_symbol1_font_outparams_fn
42027: (0)          def ll_fz_lookup_noto_symbol2_font():
```

```
42028: (4)              """
42029: (4)              Wrapper for out-params of fz_lookup_noto_symbol2_font().
42030: (4)              Returns: const unsigned char *, int len
42031: (4)              """
42032: (4)              outparams = ll_fz_lookup_noto_symbol2_font_outparams()
42033: (4)              ret = ll_fz_lookup_noto_symbol2_font_outparams_fn(outparams)
42034: (4)              return ret, outparams.len
42035: (0)          def fz_lookup_noto_symbol2_font_outparams_fn():
42036: (4)              """
42037: (4)              Class-aware helper for out-params of fz_lookup_noto_symbol2_font()
[fz_lookup_noto_symbol2_font()].
42038: (4)              """
42039: (4)              ret, len = ll_fz_lookup_noto_symbol2_font()
42040: (4)              return ret, len
42041: (0)          fz_lookup_noto_symbol2_font = fz_lookup_noto_symbol2_font_outparams_fn
42042: (0)          def ll_fz_new_deflated_data(source, source_length, level):
42043: (4)              """
42044: (4)              Wrapper for out-params of fz_new_deflated_data().
42045: (4)              Returns: unsigned char *, size_t compressed_length
42046: (4)              """
42047: (4)              outparams = ll_fz_new_deflated_data_outparams()
42048: (4)              ret = ll_fz_new_deflated_data_outparams_fn(source, source_length, level,
outparams)
42049: (4)              return ret, outparams.compressed_length
42050: (0)          def fz_new_deflated_data_outparams_fn(source, source_length, level):
42051: (4)              """
42052: (4)              Class-aware helper for out-params of fz_new_deflated_data()
[fz_new_deflated_data()].
42053: (4)              """
42054: (4)              ret, compressed_length = ll_fz_new_deflated_data(source, source_length,
level)
42055: (4)              return ret, compressed_length
42056: (0)          fz_new_deflated_data = fz_new_deflated_data_outparams_fn
42057: (0)          def ll_fz_new_deflated_data_from_buffer(buffer, level):
42058: (4)              """
42059: (4)              Wrapper for out-params of fz_new_deflated_data_from_buffer().
42060: (4)              Returns: unsigned char *, size_t compressed_length
42061: (4)              """
42062: (4)              outparams = ll_fz_new_deflated_data_from_buffer_outparams()
42063: (4)              ret = ll_fz_new_deflated_data_from_buffer_outparams_fn(buffer, level,
outparams)
42064: (4)              return ret, outparams.compressed_length
42065: (0)          def fz_new_deflated_data_from_buffer_outparams_fn(buffer, level):
42066: (4)              """
42067: (4)              Class-aware helper for out-params of fz_new_deflated_data_from_buffer()
[fz_new_deflated_data_from_buffer()].
42068: (4)              """
42069: (4)              ret, compressed_length =
ll_fz_new_deflated_data_from_buffer(buffer.m_internal, level)
42070: (4)              return ret, compressed_length
42071: (0)          fz_new_deflated_data_from_buffer =
fz_new_deflated_data_from_buffer_outparams_fn
42072: (0)          def ll_fz_new_display_list_from_svg(buf, base_uri, dir):
42073: (4)              """
42074: (4)              Wrapper for out-params of fz_new_display_list_from_svg().
42075: (4)              Returns: fz_display_list *, float w, float h
42076: (4)              """
42077: (4)              outparams = ll_fz_new_display_list_from_svg_outparams()
42078: (4)              ret = ll_fz_new_display_list_from_svg_outparams_fn(buf, base_uri, dir,
outparams)
42079: (4)              return ret, outparams.w, outparams.h
42080: (0)          def fz_new_display_list_from_svg_outparams_fn(buf, base_uri, dir):
42081: (4)              """
42082: (4)              Class-aware helper for out-params of fz_new_display_list_from_svg()
[fz_new_display_list_from_svg()].
42083: (4)              """
42084: (4)              ret, w, h = ll_fz_new_display_list_from_svg(buf.m_internal, base_uri,
dir.m_internal)
42085: (4)              return FzDisplayList(ret), w, h
```

```
42086: (0)          fz_new_display_list_from_svg = fz_new_display_list_from_svg_outparams_fn
42087: (0)          def ll_fz_new_display_list_from_svg_xml(xmldoc, xml, base_uri, dir):
42088: (4)              """
42089: (4)              Wrapper for out-params of fz_new_display_list_from_svg_xml().
42090: (4)              Returns: fz_display_list *, float w, float h
42091: (4)              """
42092: (4)              outparams = ll_fz_new_display_list_from_svg_xml_outparams()
42093: (4)              ret = ll_fz_new_display_list_from_svg_xml_outparams_fn(xmldoc, xml,
base_uri, dir, outparams)
42094: (4)              return ret, outparams.w, outparams.h
42095: (0)          def fz_new_display_list_from_svg_xml_outparams_fn(xmldoc, xml, base_uri, dir):
42096: (4)              """
42097: (4)              Class-aware helper for out-params of fz_new_display_list_from_svg_xml()
[fz_new_display_list_from_svg_xml()].
42098: (4)              """
42099: (4)              ret, w, h = ll_fz_new_display_list_from_svg_xml(xmldoc.m_internal,
xml.m_internal, base_uri, dir.m_internal)
42100: (4)              return FzDisplayList(ret), w, h
42101: (0)          fz_new_display_list_from_svg_xml =
fz_new_display_list_from_svg_xml_outparams_fn
42102: (0)          def ll_fz_new_draw_device_with_options(options, mediabox):
42103: (4)              """
42104: (4)              Wrapper for out-params of fz_new_draw_device_with_options().
42105: (4)              Returns: fz_device *, ::fz_pixmap *pixmap
42106: (4)              """
42107: (4)              outparams = ll_fz_new_draw_device_with_options_outparams()
42108: (4)              ret = ll_fz_new_draw_device_with_options_outparams_fn(options, mediabox,
outparams)
42109: (4)              return ret, outparams.pixmap
42110: (0)          def fz_new_draw_device_with_options_outparams_fn(options, mediabox):
42111: (4)              """
42112: (4)              Class-aware helper for out-params of fz_new_draw_device_with_options()
[fz_new_draw_device_with_options()].
42113: (4)              """
42114: (4)              ret, pixmap = ll_fz_new_draw_device_with_options(options.internal(),
mediabox.internal())
42115: (4)              return FzDevice(ret), FzPixmap( pixmap)
42116: (0)          fz_new_draw_device_with_options = fz_new_draw_device_with_options_outparams_fn
42117: (0)          def ll_fz_new_svg_device_with_id(out, page_width, page_height, text_format,
reuse_images):
42118: (4)              """
42119: (4)              Wrapper for out-params of fz_new_svg_device_with_id().
42120: (4)              Returns: fz_device *, int id
42121: (4)              """
42122: (4)              outparams = ll_fz_new_svg_device_with_id_outparams()
42123: (4)              ret = ll_fz_new_svg_device_with_id_outparams_fn(out, page_width,
page_height, text_format, reuse_images, outparams)
42124: (4)              return ret, outparams.id
42125: (0)          def fz_new_svg_device_with_id_outparams_fn(out, page_width, page_height,
text_format, reuse_images):
42126: (4)              """
42127: (4)              Class-aware helper for out-params of fz_new_svg_device_with_id()
[fz_new_svg_device_with_id()].
42128: (4)              """
42129: (4)              ret, id = ll_fz_new_svg_device_with_id(out.m_internal, page_width,
page_height, text_format, reuse_images)
42130: (4)              return FzDevice(ret), id
42131: (0)          fz_new_svg_device_with_id = fz_new_svg_device_with_id_outparams_fn
42132: (0)          def ll_fz_new_test_device(threshold, options, passthrough):
42133: (4)              """
42134: (4)              Wrapper for out-params of fz_new_test_device().
42135: (4)              Returns: fz_device *, int is_color
42136: (4)              """
42137: (4)              outparams = ll_fz_new_test_device_outparams()
42138: (4)              ret = ll_fz_new_test_device_outparams_fn(threshold, options, passthrough,
outparams)
42139: (4)              return ret, outparams.is_color
42140: (0)          def fz_new_test_device_outparams_fn(threshold, options, passthrough):
42141: (4)              """
```

```
42142: (4)              Class-aware helper for out-params of fz_new_test_device()
[fz_new_test_device()].
42143: (4)              """
42144: (4)              ret, is_color = ll_fz_new_test_device(threshold, options,
passthrough.m_internal)
42145: (4)              return FzDevice(ret), is_color
42146: (0)          fz_new_test_device = fz_new_test_device_outparams_fn
42147: (0)          def ll_fz_open_image_decomp_stream(arg_0, arg_1):
42148: (4)              """
42149: (4)              Wrapper for out-params of fz_open_image_decomp_stream().
42150: (4)              Returns: fz_stream *, int l2factor
42151: (4)              """
42152: (4)              outparams = ll_fz_open_image_decomp_stream_outparams()
42153: (4)              ret = ll_fz_open_image_decomp_stream_outparams_fn(arg_0, arg_1, outparams)
42154: (4)              return ret, outparams.l2factor
42155: (0)          def fz_open_image_decomp_stream_outparams_fn(arg_0, arg_1):
42156: (4)              """
42157: (4)              Class-aware helper for out-params of fz_open_image_decomp_stream()
[fz_open_image_decomp_stream()].
42158: (4)              """
42159: (4)              ret, l2factor = ll_fz_open_image_decomp_stream(arg_0.m_internal,
arg_1.m_internal)
42160: (4)              return FzStream(ret), l2factor
42161: (0)          fz_open_image_decomp_stream = fz_open_image_decomp_stream_outparams_fn
42162: (0)          def ll_fz_open_image_decomp_stream_from_buffer(arg_0):
42163: (4)              """
42164: (4)              Wrapper for out-params of fz_open_image_decomp_stream_from_buffer().
42165: (4)              Returns: fz_stream *, int l2factor
42166: (4)              """
42167: (4)              outparams = ll_fz_open_image_decomp_stream_from_buffer_outparams()
42168: (4)              ret = ll_fz_open_image_decomp_stream_from_buffer_outparams_fn(arg_0,
outparams)
42169: (4)              return ret, outparams.l2factor
42170: (0)          def fz_open_image_decomp_stream_from_buffer_outparams_fn(arg_0):
42171: (4)              """
42172: (4)              Class-aware helper for out-params of
fz_open_image_decomp_stream_from_buffer() [fz_open_image_decomp_stream_from_buffer()].
42173: (4)              """
42174: (4)              ret, l2factor =
ll_fz_open_image_decomp_stream_from_buffer(arg_0.m_internal)
42175: (4)              return FzStream(ret), l2factor
42176: (0)          fz_open_image_decomp_stream_from_buffer =
fz_open_image_decomp_stream_from_buffer_outparams_fn
42177: (0)          def ll_fz_page_presentation(page, transition):
42178: (4)              """
42179: (4)              Wrapper for out-params of fz_page_presentation().
42180: (4)              Returns: fz_transition *, float duration
42181: (4)              """
42182: (4)              outparams = ll_fz_page_presentation_outparams()
42183: (4)              ret = ll_fz_page_presentation_outparams_fn(page, transition, outparams)
42184: (4)              return ret, outparams.duration
42185: (0)          def fz_page_presentation_outparams_fn(page, transition):
42186: (4)              """
42187: (4)              Class-aware helper for out-params of fz_page_presentation()
[fz_page_presentation()].
42188: (4)              """
42189: (4)              ret, duration = ll_fz_page_presentation(page.m_internal,
transition.internal())
42190: (4)              return FzTransition(ret), duration
42191: (0)          fz_page_presentation = fz_page_presentation_outparams_fn
42192: (0)          def ll_fz_paint_shade(shade, override_cs, ctm, dest, color_params, bbox, eop):
42193: (4)              """
42194: (4)              Wrapper for out-params of fz_paint_shade().
42195: (4)              Returns: ::fz_shade_color_cache *cache
42196: (4)              """
42197: (4)              outparams = ll_fz_paint_shade_outparams()
42198: (4)              ret = ll_fz_paint_shade_outparams_fn(shade, override_cs, ctm, dest,
color_params, bbox, eop, outparams)
42199: (4)              return outparams.cache
```

```
42200: (0)          def fz_paint_shade_outparams_fn(shade, override_cs, ctm, dest, color_params,
bbox, eop):
42201: (4)              """
42202: (4)              Class-aware helper for out-params of fz_paint_shade() [fz_paint_shade()].
42203: (4)              """
42204: (4)              cache = ll_fz_paint_shade(shade.m_internal, override_cs.m_internal,
ctm.internal(), dest.m_internal, color_params.internal(), bbox.internal(), eop.m_internal)
42205: (4)              return FzShadeColorCache(ll_fz_keep_shade_color_cache( cache))
42206: (0)          fz_paint_shade = fz_paint_shade_outparams_fn
42207: (0)          def ll_fz_parse_page_range(s, n):
42208: (4)              """
42209: (4)              Wrapper for out-params of fz_parse_page_range().
42210: (4)              Returns: const char *, int a, int b
42211: (4)              """
42212: (4)              outparams = ll_fz_parse_page_range_outparams()
42213: (4)              ret = ll_fz_parse_page_range_outparams_fn(s, n, outparams)
42214: (4)              return ret, outparams.a, outparams.b
42215: (0)          def fz_parse_page_range_outparams_fn(s, n):
42216: (4)              """
42217: (4)              Class-aware helper for out-params of fz_parse_page_range()
[fz_parse_page_range()].
42218: (4)              """
42219: (4)              ret, a, b = ll_fz_parse_page_range(s, n)
42220: (4)              return ret, a, b
42221: (0)          fz_parse_page_range = fz_parse_page_range_outparams_fn
42222: (0)          def ll_fz_read_best(stm, initial, worst_case):
42223: (4)              """
42224: (4)              Wrapper for out-params of fz_read_best().
42225: (4)              Returns: fz_buffer *, int truncated
42226: (4)              """
42227: (4)              outparams = ll_fz_read_best_outparams()
42228: (4)              ret = ll_fz_read_best_outparams_fn(stm, initial, worst_case, outparams)
42229: (4)              return ret, outparams.truncated
42230: (0)          def fz_read_best_outparams_fn(stm, initial, worst_case):
42231: (4)              """
42232: (4)              Class-aware helper for out-params of fz_read_best() [fz_read_best()].
42233: (4)              """
42234: (4)              ret, truncated = ll_fz_read_best(stm.m_internal, initial, worst_case)
42235: (4)              return FzBuffer(ret), truncated
42236: (0)          fz_read_best = fz_read_best_outparams_fn
42237: (0)          def ll_fz_resolve_link(doc, uri):
42238: (4)              """
42239: (4)              Wrapper for out-params of fz_resolve_link().
42240: (4)              Returns: fz_location, float xp, float yp
42241: (4)              """
42242: (4)              outparams = ll_fz_resolve_link_outparams()
42243: (4)              ret = ll_fz_resolve_link_outparams_fn(doc, uri, outparams)
42244: (4)              return ret, outparams.xp, outparams.yp
42245: (0)          def fz_resolve_link_outparams_fn(doc, uri):
42246: (4)              """
42247: (4)              Class-aware helper for out-params of fz_resolve_link()
[fz_resolve_link()].
42248: (4)              """
42249: (4)              ret, xp, yp = ll_fz_resolve_link(doc.m_internal, uri)
42250: (4)              return FzLocation(ret), xp, yp
42251: (0)          fz_resolve_link = fz_resolve_link_outparams_fn
42252: (0)          def ll_fz_search_chapter_page_number(doc, chapter, page, needle, hit_bbox,
hit_max):
42253: (4)              """
42254: (4)              Wrapper for out-params of fz_search_chapter_page_number().
42255: (4)              Returns: int, int hit_mark
42256: (4)              """
42257: (4)              outparams = ll_fz_search_chapter_page_number_outparams()
42258: (4)              ret = ll_fz_search_chapter_page_number_outparams_fn(doc, chapter, page,
needle, hit_bbox, hit_max, outparams)
42259: (4)              return ret, outparams.hit_mark
42260: (0)          def fz_search_chapter_page_number_outparams_fn(doc, chapter, page, needle,
hit_bbox, hit_max):
42261: (4)              """
```

```
42262: (4)              Class-aware helper for out-params of fz_search_chapter_page_number()
[fz_search_chapter_page_number()].
42263: (4)              """
42264: (4)              ret, hit_mark = ll_fz_search_chapter_page_number(doc.m_internal, chapter,
page, needle, hit_bbox.internal(), hit_max)
42265: (4)              return ret, hit_mark
42266: (0)          fz_search_chapter_page_number = fz_search_chapter_page_number_outparams_fn
42267: (0)          def ll_fz_search_display_list(list, needle, hit_bbox, hit_max):
42268: (4)              """
42269: (4)              Wrapper for out-params of fz_search_display_list().
42270: (4)              Returns: int, int hit_mark
42271: (4)              """
42272: (4)              outparams = ll_fz_search_display_list_outparams()
42273: (4)              ret = ll_fz_search_display_list_outparams_fn(list, needle, hit_bbox,
hit_max, outparams)
42274: (4)              return ret, outparams.hit_mark
42275: (0)          def fz_search_display_list_outparams_fn(list, needle, hit_bbox, hit_max):
42276: (4)              """
42277: (4)              Class-aware helper for out-params of fz_search_display_list()
[fz_search_display_list()].
42278: (4)              """
42279: (4)              ret, hit_mark = ll_fz_search_display_list(list.m_internal, needle,
hit_bbox.internal(), hit_max)
42280: (4)              return ret, hit_mark
42281: (0)          fz_search_display_list = fz_search_display_list_outparams_fn
42282: (0)          def ll_fz_search_page(page, needle, hit_bbox, hit_max):
42283: (4)              """
42284: (4)              Wrapper for out-params of fz_search_page().
42285: (4)              Returns: int, int hit_mark
42286: (4)              """
42287: (4)              outparams = ll_fz_search_page_outparams()
42288: (4)              ret = ll_fz_search_page_outparams_fn(page, needle, hit_bbox, hit_max,
outparams)
42289: (4)              return ret, outparams.hit_mark
42290: (0)          def fz_search_page_outparams_fn(page, needle, hit_bbox, hit_max):
42291: (4)              """
42292: (4)              Class-aware helper for out-params of fz_search_page() [fz_search_page()].
42293: (4)              """
42294: (4)              ret, hit_mark = ll_fz_search_page(page.m_internal, needle,
hit_bbox.internal(), hit_max)
42295: (4)              return ret, hit_mark
42296: (0)          fz_search_page = fz_search_page_outparams_fn
42297: (0)          def ll_fz_search_page_number(doc, number, needle, hit_bbox, hit_max):
42298: (4)              """
42299: (4)              Wrapper for out-params of fz_search_page_number().
42300: (4)              Returns: int, int hit_mark
42301: (4)              """
42302: (4)              outparams = ll_fz_search_page_number_outparams()
42303: (4)              ret = ll_fz_search_page_number_outparams_fn(doc, number, needle, hit_bbox,
hit_max, outparams)
42304: (4)              return ret, outparams.hit_mark
42305: (0)          def fz_search_page_number_outparams_fn(doc, number, needle, hit_bbox,
hit_max):
42306: (4)              """
42307: (4)              Class-aware helper for out-params of fz_search_page_number()
[fz_search_page_number()].
42308: (4)              """
42309: (4)              ret, hit_mark = ll_fz_search_page_number(doc.m_internal, number, needle,
hit_bbox.internal(), hit_max)
42310: (4)              return ret, hit_mark
42311: (0)          fz_search_page_number = fz_search_page_number_outparams_fn
42312: (0)          def ll_fz_search_stext_page(text, needle, hit_bbox, hit_max):
42313: (4)              """
42314: (4)              Wrapper for out-params of fz_search_stext_page().
42315: (4)              Returns: int, int hit_mark
42316: (4)              """
42317: (4)              outparams = ll_fz_search_stext_page_outparams()
42318: (4)              ret = ll_fz_search_stext_page_outparams_fn(text, needle, hit_bbox,
hit_max, outparams)
```

```
42319: (4)              return ret, outparams.hit_mark
42320: (0)          def fz_search_stext_page_outparams_fn(text, needle, hit_bbox, hit_max):
42321: (4)              """
42322: (4)              Class-aware helper for out-params of fz_search_stext_page()
[fz_search_stext_page()].
42323: (4)              """
42324: (4)              ret, hit_mark = ll_fz_search_stext_page(text.m_internal, needle,
hit_bbox.internal(), hit_max)
42325: (4)              return ret, hit_mark
42326: (0)          fz_search_stext_page = fz_search_stext_page_outparams_fn
42327: (0)          def ll_fz_separation_equivalent(seps, idx, dst_cs, prf, color_params):
42328: (4)              """
42329: (4)              Wrapper for out-params of fz_separation_equivalent().
42330: (4)              Returns: float dst_color
42331: (4)              """
42332: (4)              outparams = ll_fz_separation_equivalent_outparams()
42333: (4)              ret = ll_fz_separation_equivalent_outparams_fn(seps, idx, dst_cs, prf,
color_params, outparams)
42334: (4)              return outparams.dst_color
42335: (0)          def fz_separation_equivalent_outparams_fn(seps, idx, dst_cs, prf,
color_params):
42336: (4)              """
42337: (4)              Class-aware helper for out-params of fz_separation_equivalent()
[fz_separation_equivalent()].
42338: (4)              """
42339: (4)              dst_color = ll_fz_separation_equivalent(seps.m_internal, idx,
dst_cs.m_internal, prf.m_internal, color_params.internal())
42340: (4)              return dst_color
42341: (0)          fz_separation_equivalent = fz_separation_equivalent_outparams_fn
42342: (0)          def ll_fz_store_scavenge(size):
42343: (4)              """
42344: (4)              Wrapper for out-params of fz_store_scavenge().
42345: (4)              Returns: int, int phase
42346: (4)              """
42347: (4)              outparams = ll_fz_store_scavenge_outparams()
42348: (4)              ret = ll_fz_store_scavenge_outparams_fn(size, outparams)
42349: (4)              return ret, outparams.phase
42350: (0)          def fz_store_scavenge_outparams_fn(size):
42351: (4)              """
42352: (4)              Class-aware helper for out-params of fz_store_scavenge()
[fz_store_scavenge()].
42353: (4)              """
42354: (4)              ret, phase = ll_fz_store_scavenge(size)
42355: (4)              return ret, phase
42356: (0)          fz_store_scavenge = fz_store_scavenge_outparams_fn
42357: (0)          def ll_fz_store_scavenge_external(size):
42358: (4)              """
42359: (4)              Wrapper for out-params of fz_store_scavenge_external().
42360: (4)              Returns: int, int phase
42361: (4)              """
42362: (4)              outparams = ll_fz_store_scavenge_external_outparams()
42363: (4)              ret = ll_fz_store_scavenge_external_outparams_fn(size, outparams)
42364: (4)              return ret, outparams.phase
42365: (0)          def fz_store_scavenge_external_outparams_fn(size):
42366: (4)              """
42367: (4)              Class-aware helper for out-params of fz_store_scavenge_external()
[fz_store_scavenge_external()].
42368: (4)              """
42369: (4)              ret, phase = ll_fz_store_scavenge_external(size)
42370: (4)              return ret, phase
42371: (0)          fz_store_scavenge_external = fz_store_scavenge_external_outparams_fn
42372: (0)          def ll_fz_strsep(delim):
42373: (4)              """
42374: (4)              Wrapper for out-params of fz_strsep().
42375: (4)              Returns: char *, char *stringp
42376: (4)              """
42377: (4)              outparams = ll_fz_strsep_outparams()
42378: (4)              ret = ll_fz_strsep_outparams_fn(delim, outparams)
42379: (4)              return ret, outparams.stringp
```

```
42380: (0)          def fz_strsep_outparams_fn(delim):
42381: (4)              """
42382: (4)              Class-aware helper for out-params of fz_strsep() [fz_strsep()].
42383: (4)              """
42384: (4)              ret, stringp = ll_fz_strsep(delim)
42385: (4)              return ret, stringp
42386: (0)          fz_strsep = fz_strsep_outparams_fn
42387: (0)          def ll_fz_strtof(s):
42388: (4)              """
42389: (4)              Wrapper for out-params of fz_strtof().
42390: (4)              Returns: float, char *es
42391: (4)              """
42392: (4)              outparams = ll_fz_strtof_outparams()
42393: (4)              ret = ll_fz_strtof_outparams_fn(s, outparams)
42394: (4)              return ret, outparams.es
42395: (0)          def fz_strtof_outparams_fn(s):
42396: (4)              """
42397: (4)              Class-aware helper for out-params of fz_strtof() [fz_strtof()].
42398: (4)              """
42399: (4)              ret, es = ll_fz_strtof(s)
42400: (4)              return ret, es
42401: (0)          fz_strtof = fz_strtof_outparams_fn
42402: (0)          def ll_fz_subset_cff_for_gids(orig, num_gids, symbolic, cidfont):
42403: (4)              """
42404: (4)              Wrapper for out-params of fz_subset_cff_for_gids().
42405: (4)              Returns: fz_buffer *, int gids
42406: (4)              """
42407: (4)              outparams = ll_fz_subset_cff_for_gids_outparams()
42408: (4)              ret = ll_fz_subset_cff_for_gids_outparams_fn(orig, num_gids, symbolic,
cidfont, outparams)
42409: (4)              return ret, outparams.gids
42410: (0)          def fz_subset_cff_for_gids_outparams_fn(orig, num_gids, symbolic, cidfont):
42411: (4)              """
42412: (4)              Class-aware helper for out-params of fz_subset_cff_for_gids()
[fz_subset_cff_for_gids()].
42413: (4)              """
42414: (4)              ret, gids = ll_fz_subset_cff_for_gids(orig.m_internal, num_gids, symbolic,
cidfont)
42415: (4)              return FzBuffer( ll_fz_keep_buffer( ret)), gids
42416: (0)          fz_subset_cff_for_gids = fz_subset_cff_for_gids_outparams_fn
42417: (0)          def ll_fz_subset_ttf_for_gids(orig, num_gids, symbolic, cidfont):
42418: (4)              """
42419: (4)              Wrapper for out-params of fz_subset_ttf_for_gids().
42420: (4)              Returns: fz_buffer *, int gids
42421: (4)              """
42422: (4)              outparams = ll_fz_subset_ttf_for_gids_outparams()
42423: (4)              ret = ll_fz_subset_ttf_for_gids_outparams_fn(orig, num_gids, symbolic,
cidfont, outparams)
42424: (4)              return ret, outparams.gids
42425: (0)          def fz_subset_ttf_for_gids_outparams_fn(orig, num_gids, symbolic, cidfont):
42426: (4)              """
42427: (4)              Class-aware helper for out-params of fz_subset_ttf_for_gids()
[fz_subset_ttf_for_gids()].
42428: (4)              """
42429: (4)              ret, gids = ll_fz_subset_ttf_for_gids(orig.m_internal, num_gids, symbolic,
cidfont)
42430: (4)              return FzBuffer( ll_fz_keep_buffer( ret)), gids
42431: (0)          fz_subset_ttf_for_gids = fz_subset_ttf_for_gids_outparams_fn
42432: (0)          def ll_fz_warning_callback():
42433: (4)              """
42434: (4)              Wrapper for out-params of fz_warning_callback().
42435: (4)              Returns: fz_warning_cb *, void *user
42436: (4)              """
42437: (4)              outparams = ll_fz_warning_callback_outparams()
42438: (4)              ret = ll_fz_warning_callback_outparams_fn(outparams)
42439: (4)              return ret, outparams.user
42440: (0)          def fz_warning_callback_outparams_fn():
42441: (4)              """
42442: (4)              Class-aware helper for out-params of fz_warning_callback()
```

```
[fz_warning_callback()].
42443: (4)                """
42444: (4)                ret, user = ll_fz_warning_callback()
42445: (4)                return ret, user
42446: (0)            fz_warning_callback = fz_warning_callback_outparams_fn
42447: (0)            def ll_pdf_annot_MK_BC(annot, color):
42448: (4)                """
42449: (4)                Wrapper for out-params of pdf_annot_MK_BC().
42450: (4)                Returns: int n
42451: (4)                """
42452: (4)                outparams = ll_pdf_annot_MK_BC_outparams()
42453: (4)                ret = ll_pdf_annot_MK_BC_outparams_fn(annot, color, outparams)
42454: (4)                return outparams.n
42455: (0)            def pdf_annot_MK_BC_outparams_fn(annot, color):
42456: (4)                """
42457: (4)                Class-aware helper for out-params of pdf_annot_MK_BC()
[pdf_annot_MK_BC()].
42458: (4)                """
42459: (4)                n = ll_pdf_annot_MK_BC(annot.m_internal, color)
42460: (4)                return n
42461: (0)            pdf_annot_MK_BC = pdf_annot_MK_BC_outparams_fn
42462: (0)            def ll_pdf_annot_MK_BG(annot, color):
42463: (4)                """
42464: (4)                Wrapper for out-params of pdf_annot_MK_BG().
42465: (4)                Returns: int n
42466: (4)                """
42467: (4)                outparams = ll_pdf_annot_MK_BG_outparams()
42468: (4)                ret = ll_pdf_annot_MK_BG_outparams_fn(annot, color, outparams)
42469: (4)                return outparams.n
42470: (0)            def pdf_annot_MK_BG_outparams_fn(annot, color):
42471: (4)                """
42472: (4)                Class-aware helper for out-params of pdf_annot_MK_BG()
[pdf_annot_MK_BG()].
42473: (4)                """
42474: (4)                n = ll_pdf_annot_MK_BG(annot.m_internal, color)
42475: (4)                return n
42476: (0)            pdf_annot_MK_BG = pdf_annot_MK_BG_outparams_fn
42477: (0)            def ll_pdf_annot_color(annot, color):
42478: (4)                """
42479: (4)                Wrapper for out-params of pdf_annot_color().
42480: (4)                Returns: int n
42481: (4)                """
42482: (4)                outparams = ll_pdf_annot_color_outparams()
42483: (4)                ret = ll_pdf_annot_color_outparams_fn(annot, color, outparams)
42484: (4)                return outparams.n
42485: (0)            def pdf_annot_color_outparams_fn(annot, color):
42486: (4)                """
42487: (4)                Class-aware helper for out-params of pdf_annot_color()
[pdf_annot_color()].
42488: (4)                """
42489: (4)                n = ll_pdf_annot_color(annot.m_internal, color)
42490: (4)                return n
42491: (0)            pdf_annot_color = pdf_annot_color_outparams_fn
42492: (0)            def ll_pdf_annot_default_appearance(annot, color):
42493: (4)                """
42494: (4)                Wrapper for out-params of pdf_annot_default_appearance().
42495: (4)                Returns: const char *font, float size, int n
42496: (4)                """
42497: (4)                outparams = ll_pdf_annot_default_appearance_outparams()
42498: (4)                ret = ll_pdf_annot_default_appearance_outparams_fn(annot, color,
outparams)
42499: (4)                return outparams.font, outparams.size, outparams.n
42500: (0)            def pdf_annot_default_appearance_outparams_fn(annot, color):
42501: (4)                """
42502: (4)                Class-aware helper for out-params of pdf_annot_default_appearance()
[pdf_annot_default_appearance()].
42503: (4)                """
42504: (4)                font, size, n = ll_pdf_annot_default_appearance(annot.m_internal, color)
42505: (4)                return font, size, n
```

```
(0)          pdf_annot_default_appearance = pdf_annot_default_appearance_outparams_fn
(0)          def ll_pdf_annot_interior_color(annot, color):
(4)              """
(4)              Wrapper for out-params of pdf_annot_interior_color().
(4)              Returns: int n
(4)              """
(4)              outparams = ll_pdf_annot_interior_color_outparams()
(4)              ret = ll_pdf_annot_interior_color_outparams_fn(annot, color, outparams)
(4)              return outparams.n
(0)          def pdf_annot_interior_color_outparams_fn(annot, color):
(4)              """
(4)              Class-aware helper for out-params of pdf_annot_interior_color()
[pdf_annot_interior_color()].
(4)              """
(4)              n = ll_pdf_annot_interior_color(annot.m_internal, color)
(4)              return n
(0)          pdf_annot_interior_color = pdf_annot_interior_color_outparams_fn
(0)          def ll_pdf_annot_line_ending_styles(annot):
(4)              """
(4)              Wrapper for out-params of pdf_annot_line_ending_styles().
(4)              Returns: enum pdf_line_ending start_style, enum pdf_line_ending end_style
(4)              """
(4)              outparams = ll_pdf_annot_line_ending_styles_outparams()
(4)              ret = ll_pdf_annot_line_ending_styles_outparams_fn(annot, outparams)
(4)              return outparams.start_style, outparams.end_style
(0)          def pdf_annot_line_ending_styles_outparams_fn(annot):
(4)              """
(4)              Class-aware helper for out-params of pdf_annot_line_ending_styles()
[pdf_annot_line_ending_styles()].
(4)              """
(4)              start_style, end_style = ll_pdf_annot_line_ending_styles(annot.m_internal)
(4)              return start_style, end_style
(0)          pdf_annot_line_ending_styles = pdf_annot_line_ending_styles_outparams_fn
(0)          def ll_pdf_array_get_string(array, index):
(4)              """
(4)              Wrapper for out-params of pdf_array_get_string().
(4)              Returns: const char *, size_t sizep
(4)              """
(4)              outparams = ll_pdf_array_get_string_outparams()
(4)              ret = ll_pdf_array_get_string_outparams_fn(array, index, outparams)
(4)              return ret, outparams.sizep
(0)          def pdf_array_get_string_outparams_fn(array, index):
(4)              """
(4)              Class-aware helper for out-params of pdf_array_get_string()
[pdf_array_get_string()].
(4)              """
(4)              ret, sizep = ll_pdf_array_get_string(array.m_internal, index)
(4)              return ret, sizep
(0)          pdf_array_get_string = pdf_array_get_string_outparams_fn
(0)          def ll_pdf_count_q_balance(doc, res, stm):
(4)              """
(4)              Wrapper for out-params of pdf_count_q_balance().
(4)              Returns: int prepend, int append
(4)              """
(4)              outparams = ll_pdf_count_q_balance_outparams()
(4)              ret = ll_pdf_count_q_balance_outparams_fn(doc, res, stm, outparams)
(4)              return outparams.prepend, outparams.append
(0)          def pdf_count_q_balance_outparams_fn(doc, res, stm):
(4)              """
(4)              Class-aware helper for out-params of pdf_count_q_balance()
[pdf_count_q_balance()].
(4)              """
(4)              prepend, append = ll_pdf_count_q_balance(doc.m_internal, res.m_internal,
stm.m_internal)
(4)              return prepend, append
(0)          pdf_count_q_balance = pdf_count_q_balance_outparams_fn
(0)          def ll_pdf_decode_cmap(cmap, s, e):
(4)              """
(4)              Wrapper for out-params of pdf_decode_cmap().
```

```
42570: (4)              Returns: int, unsigned int cpt
42571: (4)              """
42572: (4)              outparams = ll_pdf_decode_cmap_outparams()
42573: (4)              ret = ll_pdf_decode_cmap_outparams_fn(cmap, s, e, outparams)
42574: (4)              return ret, outparams.cpt
42575: (0)          def pdf_decode_cmap_outparams_fn(cmap, s, e):
42576: (4)              """
42577: (4)              Class-aware helper for out-params of pdf_decode_cmap()
[pdf_decode_cmap()].
42578: (4)              """
42579: (4)              ret, cpt = ll_pdf_decode_cmap(cmap.m_internal, s, e)
42580: (4)              return ret, cpt
42581: (0)          pdf_decode_cmap = pdf_decode_cmap_outparams_fn
42582: (0)          def ll_pdf_dict_get_inheritable_string(dict, key):
42583: (4)              """
42584: (4)              Wrapper for out-params of pdf_dict_get_inheritable_string().
42585: (4)              Returns: const char *, size_t sizep
42586: (4)              """
42587: (4)              outparams = ll_pdf_dict_get_inheritable_string_outparams()
42588: (4)              ret = ll_pdf_dict_get_inheritable_string_outparams_fn(dict, key,
outparams)
42589: (4)              return ret, outparams.sizep
42590: (0)          def pdf_dict_get_inheritable_string_outparams_fn(dict, key):
42591: (4)              """
42592: (4)              Class-aware helper for out-params of pdf_dict_get_inheritable_string()
[pdf_dict_get_inheritable_string()].
42593: (4)              """
42594: (4)              ret, sizep = ll_pdf_dict_get_inheritable_string(dict.m_internal,
key.m_internal)
42595: (4)              return ret, sizep
42596: (0)          pdf_dict_get_inheritable_string = pdf_dict_get_inheritable_string_outparams_fn
42597: (0)          def ll_pdf_dict_get_put_drop(dict, key, val):
42598: (4)              """
42599: (4)              Wrapper for out-params of pdf_dict_get_put_drop().
42600: (4)              Returns: ::pdf_obj *old_val
42601: (4)              """
42602: (4)              outparams = ll_pdf_dict_get_put_drop_outparams()
42603: (4)              ret = ll_pdf_dict_get_put_drop_outparams_fn(dict, key, val, outparams)
42604: (4)              return outparams.old_val
42605: (0)          def ll_pdf_dict_get_string(dict, key):
42606: (4)              """
42607: (4)              Wrapper for out-params of pdf_dict_get_string().
42608: (4)              Returns: const char *, size_t sizep
42609: (4)              """
42610: (4)              outparams = ll_pdf_dict_get_string_outparams()
42611: (4)              ret = ll_pdf_dict_get_string_outparams_fn(dict, key, outparams)
42612: (4)              return ret, outparams.sizep
42613: (0)          def pdf_dict_get_string_outparams_fn(dict, key):
42614: (4)              """
42615: (4)              Class-aware helper for out-params of pdf_dict_get_string()
[pdf_dict_get_string()].
42616: (4)              """
42617: (4)              ret, sizep = ll_pdf_dict_get_string(dict.m_internal, key.m_internal)
42618: (4)              return ret, sizep
42619: (0)          pdf_dict_get_string = pdf_dict_get_string_outparams_fn
42620: (0)          def ll_pdf_edit_text_field_value(widget, value, change):
42621: (4)              """
42622: (4)              Wrapper for out-params of pdf_edit_text_field_value().
42623: (4)              Returns: int, int selStart, int selEnd, char *newvalue
42624: (4)              """
42625: (4)              outparams = ll_pdf_edit_text_field_value_outparams()
42626: (4)              ret = ll_pdf_edit_text_field_value_outparams_fn(widget, value, change,
outparams)
42627: (4)              return ret, outparams.selStart, outparams.selEnd, outparams.newvalue
42628: (0)          def pdf_edit_text_field_value_outparams_fn(widget, value, change):
42629: (4)              """
42630: (4)              Class-aware helper for out-params of pdf_edit_text_field_value()
[pdf_edit_text_field_value()].
42631: (4)              """
```

```
42632: (4)              ret, selStart, selEnd, newvalue =
ll_pdf_edit_text_field_value(widget.m_internal, value, change)
42633: (4)              return ret, selStart, selEnd, newvalue
42634: (0)          pdf_edit_text_field_value = pdf_edit_text_field_value_outparams_fn
42635: (0)          def ll_pdf_eval_function(func, in_, inlen, outlen):
42636: (4)              """
42637: (4)              Wrapper for out-params of pdf_eval_function().
42638: (4)              Returns: float out
42639: (4)              """
42640: (4)              outparams = ll_pdf_eval_function_outparams()
42641: (4)              ret = ll_pdf_eval_function_outparams_fn(func, in_, inlen, outlen,
outparams)
42642: (4)              return outparams.out
42643: (0)          def pdf_eval_function_outparams_fn(func, in_, inlen, outlen):
42644: (4)              """
42645: (4)              Class-aware helper for out-params of pdf_eval_function()
[pdf_eval_function()].
42646: (4)              """
42647: (4)              out = ll_pdf_eval_function(func.m_internal, in_, inlen, outlen)
42648: (4)              return out
42649: (0)          pdf_eval_function = pdf_eval_function_outparams_fn
42650: (0)          def ll_pdf_field_event_validate(doc, field, value):
42651: (4)              """
42652: (4)              Wrapper for out-params of pdf_field_event_validate().
42653: (4)              Returns: int, char *newvalue
42654: (4)              """
42655: (4)              outparams = ll_pdf_field_event_validate_outparams()
42656: (4)              ret = ll_pdf_field_event_validate_outparams_fn(doc, field, value,
outparams)
42657: (4)              return ret, outparams.newvalue
42658: (0)          def pdf_field_event_validate_outparams_fn(doc, field, value):
42659: (4)              """
42660: (4)              Class-aware helper for out-params of pdf_field_event_validate()
[pdf_field_event_validate()].
42661: (4)              """
42662: (4)              ret, newvalue = ll_pdf_field_event_validate(doc.m_internal,
field.m_internal, value)
42663: (4)              return ret, newvalue
42664: (0)          pdf_field_event_validate = pdf_field_event_validate_outparams_fn
42665: (0)          def ll_pdf_js_event_result_validate(js):
42666: (4)              """
42667: (4)              Wrapper for out-params of pdf_js_event_result_validate().
42668: (4)              Returns: int, char *newvalue
42669: (4)              """
42670: (4)              outparams = ll_pdf_js_event_result_validate_outparams()
42671: (4)              ret = ll_pdf_js_event_result_validate_outparams_fn(js, outparams)
42672: (4)              return ret, outparams.newvalue
42673: (0)          def pdf_js_event_result_validate_outparams_fn(js):
42674: (4)              """
42675: (4)              Class-aware helper for out-params of pdf_js_event_result_validate()
[pdf_js_event_result_validate()].
42676: (4)              """
42677: (4)              ret, newvalue = ll_pdf_js_event_result_validate(js.m_internal)
42678: (4)              return ret, newvalue
42679: (0)          pdf_js_event_result_validate = pdf_js_event_result_validate_outparams_fn
42680: (0)          def ll_pdf_js_execute(js, name, code):
42681: (4)              """
42682: (4)              Wrapper for out-params of pdf_js_execute().
42683: (4)              Returns: char *result
42684: (4)              """
42685: (4)              outparams = ll_pdf_js_execute_outparams()
42686: (4)              ret = ll_pdf_js_execute_outparams_fn(js, name, code, outparams)
42687: (4)              return outparams.result
42688: (0)          def pdf_js_execute_outparams_fn(js, name, code):
42689: (4)              """
42690: (4)              Class-aware helper for out-params of pdf_js_execute() [pdf_js_execute()].
42691: (4)              """
42692: (4)              result = ll_pdf_js_execute(js.m_internal, name, code)
42693: (4)              return result
```

```
42694: (0)          pdf_js_execute = pdf_js_execute_outparams_fn
42695: (0)          def ll_pdf_load_encoding(encoding):
42696: (4)              """
42697: (4)              Wrapper for out-params of pdf_load_encoding().
42698: (4)              Returns: const char *estrings
42699: (4)              """
42700: (4)              outparams = ll_pdf_load_encoding_outparams()
42701: (4)              ret = ll_pdf_load_encoding_outparams_fn(encoding, outparams)
42702: (4)              return outparams.estrings
42703: (0)          def pdf_load_encoding_outparams_fn(encoding):
42704: (4)              """
42705: (4)              Class-aware helper for out-params of pdf_load_encoding()
[pdf_load_encoding()].
42706: (4)              """
42707: (4)              estrings = ll_pdf_load_encoding(encoding)
42708: (4)              return estrings
42709: (0)          pdf_load_encoding = pdf_load_encoding_outparams_fn
42710: (0)          def ll_pdf_load_to_unicode(doc, font, collection, cmapstm):
42711: (4)              """
42712: (4)              Wrapper for out-params of pdf_load_to_unicode().
42713: (4)              Returns: const char *strings
42714: (4)              """
42715: (4)              outparams = ll_pdf_load_to_unicode_outparams()
42716: (4)              ret = ll_pdf_load_to_unicode_outparams_fn(doc, font, collection, cmapstm,
outparams)
42717: (4)              return outparams.strings
42718: (0)          def pdf_load_to_unicode_outparams_fn(doc, font, collection, cmapstm):
42719: (4)              """
42720: (4)              Class-aware helper for out-params of pdf_load_to_unicode()
[pdf_load_to_unicode()].
42721: (4)              """
42722: (4)              strings = ll_pdf_load_to_unicode(doc.m_internal, font.m_internal,
collection, cmapstm.m_internal)
42723: (4)              return strings
42724: (0)          pdf_load_to_unicode = pdf_load_to_unicode_outparams_fn
42725: (0)          def ll_pdf_lookup_cmap_full(cmap, cpt):
42726: (4)              """
42727: (4)              Wrapper for out-params of pdf_lookup_cmap_full().
42728: (4)              Returns: int, int out
42729: (4)              """
42730: (4)              outparams = ll_pdf_lookup_cmap_full_outparams()
42731: (4)              ret = ll_pdf_lookup_cmap_full_outparams_fn(cmap, cpt, outparams)
42732: (4)              return ret, outparams.out
42733: (0)          def pdf_lookup_cmap_full_outparams_fn(cmap, cpt):
42734: (4)              """
42735: (4)              Class-aware helper for out-params of pdf_lookup_cmap_full()
[pdf_lookup_cmap_full()].
42736: (4)              """
42737: (4)              ret, out = ll_pdf_lookup_cmap_full(cmap.m_internal, cpt)
42738: (4)              return ret, out
42739: (0)          pdf_lookup_cmap_full = pdf_lookup_cmap_full_outparams_fn
42740: (0)          def ll_pdf_lookup_page_loc(doc, needle):
42741: (4)              """
42742: (4)              Wrapper for out-params of pdf_lookup_page_loc().
42743: (4)              Returns: pdf_obj *, ::pdf_obj *parentp, int indexp
42744: (4)              """
42745: (4)              outparams = ll_pdf_lookup_page_loc_outparams()
42746: (4)              ret = ll_pdf_lookup_page_loc_outparams_fn(doc, needle, outparams)
42747: (4)              return ret, outparams.parentp, outparams.indexp
42748: (0)          def pdf_lookup_page_loc_outparams_fn(doc, needle):
42749: (4)              """
42750: (4)              Class-aware helper for out-params of pdf_lookup_page_loc()
[pdf_lookup_page_loc()].
42751: (4)              """
42752: (4)              ret, parentp, indexp = ll_pdf_lookup_page_loc(doc.m_internal, needle)
42753: (4)              return PdfObj( ll_pdf_keep_obj( ret)), PdfObj(ll_pdf_keep_obj( parentp)),
indexp
42754: (0)          pdf_lookup_page_loc = pdf_lookup_page_loc_outparams_fn
42755: (0)          def ll_pdf_lookup_substitute_font(mono, serif, bold, italic):
```

```
42756: (4)              """
42757: (4)              Wrapper for out-params of pdf_lookup_substitute_font().
42758: (4)              Returns: const unsigned char *, int len
42759: (4)              """
42760: (4)              outparams = ll_pdf_lookup_substitute_font_outparams()
42761: (4)              ret = ll_pdf_lookup_substitute_font_outparams_fn(mono, serif, bold,
italic, outparams)
42762: (4)              return ret, outparams.len
42763: (0)          def pdf_lookup_substitute_font_outparams_fn(mono, serif, bold, italic):
42764: (4)              """
42765: (4)              Class-aware helper for out-params of pdf_lookup_substitute_font()
[pdf_lookup_substitute_font()].
42766: (4)              """
42767: (4)              ret, len = ll_pdf_lookup_substitute_font(mono, serif, bold, italic)
42768: (4)              return ret, len
42769: (0)          pdf_lookup_substitute_font = pdf_lookup_substitute_font_outparams_fn
42770: (0)          def ll_pdf_map_one_to_many(cmap, one, len):
42771: (4)              """
42772: (4)              Wrapper for out-params of pdf_map_one_to_many().
42773: (4)              Returns: int many
42774: (4)              """
42775: (4)              outparams = ll_pdf_map_one_to_many_outparams()
42776: (4)              ret = ll_pdf_map_one_to_many_outparams_fn(cmap, one, len, outparams)
42777: (4)              return outparams.many
42778: (0)          def pdf_map_one_to_many_outparams_fn(cmap, one, len):
42779: (4)              """
42780: (4)              Class-aware helper for out-params of pdf_map_one_to_many()
[pdf_map_one_to_many()].
42781: (4)              """
42782: (4)              many = ll_pdf_map_one_to_many(cmap.m_internal, one, len)
42783: (4)              return many
42784: (0)          pdf_map_one_to_many = pdf_map_one_to_many_outparams_fn
42785: (0)          def ll_pdf_obj_memo(obj, bit):
42786: (4)              """
42787: (4)              Wrapper for out-params of pdf_obj_memo().
42788: (4)              Returns: int, int memo
42789: (4)              """
42790: (4)              outparams = ll_pdf_obj_memo_outparams()
42791: (4)              ret = ll_pdf_obj_memo_outparams_fn(obj, bit, outparams)
42792: (4)              return ret, outparams.memo
42793: (0)          def pdf_obj_memo_outparams_fn(obj, bit):
42794: (4)              """
42795: (4)              Class-aware helper for out-params of pdf_obj_memo() [pdf_obj_memo()].
42796: (4)              """
42797: (4)              ret, memo = ll_pdf_obj_memo(obj.m_internal, bit)
42798: (4)              return ret, memo
42799: (0)          pdf_obj_memo = pdf_obj_memo_outparams_fn
42800: (0)          def ll_pdf_page_presentation(page, transition):
42801: (4)              """
42802: (4)              Wrapper for out-params of pdf_page_presentation().
42803: (4)              Returns: fz_transition *, float duration
42804: (4)              """
42805: (4)              outparams = ll_pdf_page_presentation_outparams()
42806: (4)              ret = ll_pdf_page_presentation_outparams_fn(page, transition, outparams)
42807: (4)              return ret, outparams.duration
42808: (0)          def pdf_page_presentation_outparams_fn(page, transition):
42809: (4)              """
42810: (4)              Class-aware helper for out-params of pdf_page_presentation()
[pdf_page_presentation()].
42811: (4)              """
42812: (4)              ret, duration = ll_pdf_page_presentation(page.m_internal,
transition.internal())
42813: (4)              return FzTransition(ret), duration
42814: (0)          pdf_page_presentation = pdf_page_presentation_outparams_fn
42815: (0)          def ll_pdf_page_write(doc, mediabox):
42816: (4)              """
42817: (4)              Wrapper for out-params of pdf_page_write().
42818: (4)              Returns: fz_device *, ::pdf_obj *presources, ::fz_buffer *pcontents
42819: (4)              """
```

```
42820: (4)              outparams = ll_pdf_page_write_outparams()
42821: (4)              ret = ll_pdf_page_write_outparams_fn(doc, mediabox, outparams)
42822: (4)              return ret, outparams.presources, outparams.pcontents
42823: (0)          def pdf_page_write_outparams_fn(doc, mediabox):
42824: (4)              """
42825: (4)              Class-aware helper for out-params of pdf_page_write() [pdf_page_write()].
42826: (4)              """
42827: (4)              ret, presources, pcontents = ll_pdf_page_write(doc.m_internal,
mediabox.internal())
42828: (4)              return FzDevice(ret), PdfObj( presources), FzBuffer( pcontents)
42829: (0)          pdf_page_write = pdf_page_write_outparams_fn
42830: (0)          def ll_pdf_parse_default_appearance(da, color):
42831: (4)              """
42832: (4)              Wrapper for out-params of pdf_parse_default_appearance().
42833: (4)              Returns: const char *font, float size, int n
42834: (4)              """
42835: (4)              outparams = ll_pdf_parse_default_appearance_outparams()
42836: (4)              ret = ll_pdf_parse_default_appearance_outparams_fn(da, color, outparams)
42837: (4)              return outparams.font, outparams.size, outparams.n
42838: (0)          def pdf_parse_default_appearance_outparams_fn(da, color):
42839: (4)              """
42840: (4)              Class-aware helper for out-params of pdf_parse_default_appearance()
[pdf_parse_default_appearance()].
42841: (4)              """
42842: (4)              font, size, n = ll_pdf_parse_default_appearance(da, color)
42843: (4)              return font, size, n
42844: (0)          pdf_parse_default_appearance = pdf_parse_default_appearance_outparams_fn
42845: (0)          def ll_pdf_parse_ind_obj(doc, f):
42846: (4)              """
42847: (4)              Wrapper for out-params of pdf_parse_ind_obj().
42848: (4)              Returns: pdf_obj *, int num, int gen, int64_t stm_ofs, int try_repair
42849: (4)              """
42850: (4)              outparams = ll_pdf_parse_ind_obj_outparams()
42851: (4)              ret = ll_pdf_parse_ind_obj_outparams_fn(doc, f, outparams)
42852: (4)              return ret, outparams.num, outparams.gen, outparams.stm_ofs,
outparams.try_repair
42853: (0)          def pdf_parse_ind_obj_outparams_fn(doc, f):
42854: (4)              """
42855: (4)              Class-aware helper for out-params of pdf_parse_ind_obj()
[pdf_parse_ind_obj()].
42856: (4)              """
42857: (4)              ret, num, gen, stm_ofs, try_repair = ll_pdf_parse_ind_obj(doc.m_internal,
f.m_internal)
42858: (4)              return PdfObj(ret), num, gen, stm_ofs, try_repair
42859: (0)          pdf_parse_ind_obj = pdf_parse_ind_obj_outparams_fn
42860: (0)          def ll_pdf_parse_journal_obj(doc, stm):
42861: (4)              """
42862: (4)              Wrapper for out-params of pdf_parse_journal_obj().
42863: (4)              Returns: pdf_obj *, int onum, ::fz_buffer *ostm, int newobj
42864: (4)              """
42865: (4)              outparams = ll_pdf_parse_journal_obj_outparams()
42866: (4)              ret = ll_pdf_parse_journal_obj_outparams_fn(doc, stm, outparams)
42867: (4)              return ret, outparams.onum, outparams.ostm, outparams.newobj
42868: (0)          def pdf_parse_journal_obj_outparams_fn(doc, stm):
42869: (4)              """
42870: (4)              Class-aware helper for out-params of pdf_parse_journal_obj()
[pdf_parse_journal_obj()].
42871: (4)              """
42872: (4)              ret, onum, ostm, newobj = ll_pdf_parse_journal_obj(doc.m_internal,
stm.m_internal)
42873: (4)              return PdfObj(ret), onum, FzBuffer( ostm), newobj
42874: (0)          pdf_parse_journal_obj = pdf_parse_journal_obj_outparams_fn
42875: (0)          def ll_pdf_print_encrypted_obj(out, obj, tight, ascii, crypt, num, gen):
42876: (4)              """
42877: (4)              Wrapper for out-params of pdf_print_encrypted_obj().
42878: (4)              Returns: int sep
42879: (4)              """
42880: (4)              outparams = ll_pdf_print_encrypted_obj_outparams()
42881: (4)              ret = ll_pdf_print_encrypted_obj_outparams_fn(out, obj, tight, ascii,
```

```
crypt, num, gen, outparams)
42882: (4)              return outparams.sep
42883: (0)          def pdf_print_encrypted_obj_outparams_fn(out, obj, tight, ascii, crypt, num,
gen):
42884: (4)              """
42885: (4)              Class-aware helper for out-params of pdf_print_encrypted_obj()
[pdf_print_encrypted_obj()].
42886: (4)              """
42887: (4)              sep = ll_pdf_print_encrypted_obj(out.m_internal, obj.m_internal, tight,
ascii, crypt.m_internal, num, gen)
42888: (4)              return sep
42889: (0)          pdf_print_encrypted_obj = pdf_print_encrypted_obj_outparams_fn
42890: (0)          def ll_pdf_process_contents(proc, doc, res, stm, cookie):
42891: (4)              """
42892: (4)              Wrapper for out-params of pdf_process_contents().
42893: (4)              Returns: ::pdf_obj *out_res
42894: (4)              """
42895: (4)              outparams = ll_pdf_process_contents_outparams()
42896: (4)              ret = ll_pdf_process_contents_outparams_fn(proc, doc, res, stm, cookie,
outparams)
42897: (4)              return outparams.out_res
42898: (0)          def pdf_process_contents_outparams_fn(proc, doc, res, stm, cookie):
42899: (4)              """
42900: (4)              Class-aware helper for out-params of pdf_process_contents()
[pdf_process_contents()].
42901: (4)              """
42902: (4)              out_res = ll_pdf_process_contents(proc.m_internal, doc.m_internal,
res.m_internal, stm.m_internal, cookie.m_internal)
42903: (4)              return PdfObj(ll_pdf_keep_obj( out_res))
42904: (0)          pdf_process_contents = pdf_process_contents_outparams_fn
42905: (0)          def ll_pdf_repair_obj(doc, buf):
42906: (4)              """
42907: (4)              Wrapper for out-params of pdf_repair_obj().
42908: (4)              Returns: int, int64_t stmofsp, int64_t stmlenp, ::pdf_obj *encrypt,
::pdf_obj *id, ::pdf_obj *page, int64_t tmpofs, ::pdf_obj *root
42909: (4)              """
42910: (4)              outparams = ll_pdf_repair_obj_outparams()
42911: (4)              ret = ll_pdf_repair_obj_outparams_fn(doc, buf, outparams)
42912: (4)              return ret, outparams.stmofsp, outparams.stmlenp, outparams.encrypt,
outparams.id, outparams.page, outparams.tmpofs, outparams.root
42913: (0)          def pdf_repair_obj_outparams_fn(doc, buf):
42914: (4)              """
42915: (4)              Class-aware helper for out-params of pdf_repair_obj() [pdf_repair_obj()].
42916: (4)              """
42917: (4)              ret, stmofsp, stmlenp, encrypt, id, page, tmpofs, root =
ll_pdf_repair_obj(doc.m_internal, buf.m_internal)
42918: (4)              return ret, stmofsp, stmlenp, PdfObj(ll_pdf_keep_obj( encrypt)),
PdfObj(ll_pdf_keep_obj( id)), PdfObj(ll_pdf_keep_obj( page)), tmpofs, PdfObj(ll_pdf_keep_obj(
root))
42919: (0)          pdf_repair_obj = pdf_repair_obj_outparams_fn
42920: (0)          def ll_pdf_resolve_link(doc, uri):
42921: (4)              """
42922: (4)              Wrapper for out-params of pdf_resolve_link().
42923: (4)              Returns: int, float xp, float yp
42924: (4)              """
42925: (4)              outparams = ll_pdf_resolve_link_outparams()
42926: (4)              ret = ll_pdf_resolve_link_outparams_fn(doc, uri, outparams)
42927: (4)              return ret, outparams.xp, outparams.yp
42928: (0)          def pdf_resolve_link_outparams_fn(doc, uri):
42929: (4)              """
42930: (4)              Class-aware helper for out-params of pdf_resolve_link()
[pdf_resolve_link()].
42931: (4)              """
42932: (4)              ret, xp, yp = ll_pdf_resolve_link(doc.m_internal, uri)
42933: (4)              return ret, xp, yp
42934: (0)          pdf_resolve_link = pdf_resolve_link_outparams_fn
42935: (0)          def ll_pdf_sample_shade_function(n, funcs, t0, t1):
42936: (4)              """
42937: (4)              Wrapper for out-params of pdf_sample_shade_function().
```

```
42938: (4)              Returns: float samples, ::pdf_function *func
42939: (4)              """
42940: (4)              outparams = ll_pdf_sample_shade_function_outparams()
42941: (4)              ret = ll_pdf_sample_shade_function_outparams_fn(n, funcs, t0, t1,
outparams)
42942: (4)              return outparams.samples, outparams.func
42943: (0)          def pdf_sample_shade_function_outparams_fn(n, funcs, t0, t1):
42944: (4)              """
42945: (4)              Class-aware helper for out-params of pdf_sample_shade_function()
[pdf_sample_shade_function()].
42946: (4)              """
42947: (4)              samples, func = ll_pdf_sample_shade_function(n, funcs, t0, t1)
42948: (4)              return samples, PdfFunction(ll_pdf_keep_function( func))
42949: (0)          pdf_sample_shade_function = pdf_sample_shade_function_outparams_fn
42950: (0)          def ll_pdf_signature_contents(doc, signature):
42951: (4)              """
42952: (4)              Wrapper for out-params of pdf_signature_contents().
42953: (4)              Returns: size_t, char *contents
42954: (4)              """
42955: (4)              outparams = ll_pdf_signature_contents_outparams()
42956: (4)              ret = ll_pdf_signature_contents_outparams_fn(doc, signature, outparams)
42957: (4)              return ret, outparams.contents
42958: (0)          def pdf_signature_contents_outparams_fn(doc, signature):
42959: (4)              """
42960: (4)              Class-aware helper for out-params of pdf_signature_contents()
[pdf_signature_contents()].
42961: (4)              """
42962: (4)              ret, contents = ll_pdf_signature_contents(doc.m_internal,
signature.m_internal)
42963: (4)              return ret, contents
42964: (0)          pdf_signature_contents = pdf_signature_contents_outparams_fn
42965: (0)          def ll_pdf_sprint_obj(buf, cap, obj, tight, ascii):
42966: (4)              """
42967: (4)              Wrapper for out-params of pdf_sprint_obj().
42968: (4)              Returns: char *, size_t len
42969: (4)              """
42970: (4)              outparams = ll_pdf_sprint_obj_outparams()
42971: (4)              ret = ll_pdf_sprint_obj_outparams_fn(buf, cap, obj, tight, ascii,
outparams)
42972: (4)              return ret, outparams.len
42973: (0)          def pdf_sprint_obj_outparams_fn(buf, cap, obj, tight, ascii):
42974: (4)              """
42975: (4)              Class-aware helper for out-params of pdf_sprint_obj() [pdf_sprint_obj()].
42976: (4)              """
42977: (4)              ret, len = ll_pdf_sprint_obj(buf, cap, obj.m_internal, tight, ascii)
42978: (4)              return ret, len
42979: (0)          pdf_sprint_obj = pdf_sprint_obj_outparams_fn
42980: (0)          def ll_pdf_to_string(obj):
42981: (4)              """
42982: (4)              Wrapper for out-params of pdf_to_string().
42983: (4)              Returns: const char *, size_t sizep
42984: (4)              """
42985: (4)              outparams = ll_pdf_to_string_outparams()
42986: (4)              ret = ll_pdf_to_string_outparams_fn(obj, outparams)
42987: (4)              return ret, outparams.sizep
42988: (0)          def pdf_to_string_outparams_fn(obj):
42989: (4)              """
42990: (4)              Class-aware helper for out-params of pdf_to_string() [pdf_to_string()].
42991: (4)              """
42992: (4)              ret, sizep = ll_pdf_to_string(obj.m_internal)
42993: (4)              return ret, sizep
42994: (0)          pdf_to_string = pdf_to_string_outparams_fn
42995: (0)          def ll_pdf_undoredo_state(doc):
42996: (4)              """
42997: (4)              Wrapper for out-params of pdf_undoredo_state().
42998: (4)              Returns: int, int steps
42999: (4)              """
43000: (4)              outparams = ll_pdf_undoredo_state_outparams()
43001: (4)              ret = ll_pdf_undoredo_state_outparams_fn(doc, outparams)
```

```
43002: (4)              return ret, outparams.steps
43003: (0)          def pdf_undoredo_state_outparams_fn(doc):
43004: (4)              """
43005: (4)              Class-aware helper for out-params of pdf_undoredo_state()
[pdf_undoredo_state()].
43006: (4)              """
43007: (4)              ret, steps = ll_pdf_undoredo_state(doc.m_internal)
43008: (4)              return ret, steps
43009: (0)          pdf_undoredo_state = pdf_undoredo_state_outparams_fn
43010: (0)          def ll_pdf_walk_tree(tree, kid_name, arrive, leave, arg):
43011: (4)              """
43012: (4)              Wrapper for out-params of pdf_walk_tree().
43013: (4)              Returns: ::pdf_obj *names, ::pdf_obj *values
43014: (4)              """
43015: (4)              outparams = ll_pdf_walk_tree_outparams()
43016: (4)              ret = ll_pdf_walk_tree_outparams_fn(tree, kid_name, arrive, leave, arg,
outparams)
43017: (4)              return outparams.names, outparams.values
43018: (0)          def pdf_walk_tree_outparams_fn(tree, kid_name, arrive, leave, arg):
43019: (4)              """
43020: (4)              Class-aware helper for out-params of pdf_walk_tree() [pdf_walk_tree()].
43021: (4)              """
43022: (4)              names, values = ll_pdf_walk_tree(tree.m_internal, kid_name.m_internal,
arrive, leave, arg)
43023: (4)              return PdfObj(ll_pdf_keep_obj( names)), PdfObj(ll_pdf_keep_obj( values))
43024: (0)          pdf_walk_tree = pdf_walk_tree_outparams_fn
43025: (0)          def FzBitmap_fz_bitmap_details_outparams_fn( self):
43026: (4)              """
43027: (4)              Helper for out-params of class method fz_bitmap::ll_fz_bitmap_details()
[fz_bitmap_details()].
43028: (4)              """
43029: (4)              w, h, n, stride = ll_fz_bitmap_details( self.m_internal)
43030: (4)              return w, h, n, stride
43031: (0)          FzBitmap.fz_bitmap_details = FzBitmap_fz_bitmap_details_outparams_fn
43032: (0)          def FzBuffer_fz_buffer_extract_outparams_fn( self):
43033: (4)              """
43034: (4)              Helper for out-params of class method fz_buffer::ll_fz_buffer_extract()
[fz_buffer_extract()].
43035: (4)              """
43036: (4)              ret, data = ll_fz_buffer_extract( self.m_internal)
43037: (4)              return ret, data
43038: (0)          FzBuffer.fz_buffer_extract = FzBuffer_fz_buffer_extract_outparams_fn
43039: (0)          def FzBuffer_fz_buffer_storage_outparams_fn( self):
43040: (4)              """
43041: (4)              Helper for out-params of class method fz_buffer::ll_fz_buffer_storage()
[fz_buffer_storage()].
43042: (4)              """
43043: (4)              ret, datap = ll_fz_buffer_storage( self.m_internal)
43044: (4)              return ret, datap
43045: (0)          FzBuffer.fz_buffer_storage = FzBuffer_fz_buffer_storage_outparams_fn
43046: (0)          def FzBuffer_fz_new_display_list_from_svg_outparams_fn( self, base_uri, dir):
43047: (4)              """
43048: (4)              Helper for out-params of class method
fz_buffer::ll_fz_new_display_list_from_svg() [fz_new_display_list_from_svg()].
43049: (4)              """
43050: (4)              ret, w, h = ll_fz_new_display_list_from_svg( self.m_internal, base_uri,
dir.m_internal)
43051: (4)              return FzDisplayList(ret), w, h
43052: (0)          FzBuffer.fz_new_display_list_from_svg =
FzBuffer_fz_new_display_list_from_svg_outparams_fn
43053: (0)          def FzBuffer_fz_subset_cff_for_gids_outparams_fn( self, num_gids, symbolic,
cidfont):
43054: (4)              """
43055: (4)              Helper for out-params of class method
fz_buffer::ll_fz_subset_cff_for_gids() [fz_subset_cff_for_gids()].
43056: (4)              """
43057: (4)              ret, gids = ll_fz_subset_cff_for_gids( self.m_internal, num_gids,
symbolic, cidfont)
43058: (4)              return FzBuffer( ll_fz_keep_buffer( ret)), gids
```

```
43059: (0)          FzBuffer.fz_subset_cff_for_gids = FzBuffer_fz_subset_cff_for_gids_outparams_fn
43060: (0)          def FzBuffer_fz_subset_ttf_for_gids_outparams_fn( self, num_gids, symbolic,
cidfont):
43061: (4)              """
43062: (4)              Helper for out-params of class method
fz_buffer::ll_fz_subset_ttf_for_gids() [fz_subset_ttf_for_gids()].
43063: (4)              """
43064: (4)              ret, gids = ll_fz_subset_ttf_for_gids( self.m_internal, num_gids,
symbolic, cidfont)
43065: (4)              return FzBuffer( ll_fz_keep_buffer( ret)), gids
43066: (0)          FzBuffer.fz_subset_ttf_for_gids = FzBuffer_fz_subset_ttf_for_gids_outparams_fn
43067: (0)          def FzColorspace_fz_clamp_color_outparams_fn( self, in_):
43068: (4)              """
43069: (4)              Helper for out-params of class method fz_colorspace::ll_fz_clamp_color()
[fz_clamp_color()].
43070: (4)              """
43071: (4)              out = ll_fz_clamp_color( self.m_internal, in_)
43072: (4)              return out
43073: (0)          FzColorspace.fz_clamp_color = FzColorspace_fz_clamp_color_outparams_fn
43074: (0)          def FzColorspace_fz_convert_color_outparams_fn( self, sv, params):
43075: (4)              """
43076: (4)              Helper for out-params of class method fz_colorspace::ll_fz_convert_color()
[fz_convert_color()].
43077: (4)              """
43078: (4)              dv = ll_fz_convert_color( self.m_internal, sv, params.internal())
43079: (4)              return dv
43080: (0)          FzColorspace.fz_convert_color = FzColorspace_fz_convert_color_outparams_fn
43081: (0)          def FzColorspace_fz_convert_separation_colors_outparams_fn( self, src_color,
dst_seps, color_params):
43082: (4)              """
43083: (4)              Helper for out-params of class method
fz_colorspace::ll_fz_convert_separation_colors() [fz_convert_separation_colors()].
43084: (4)              """
43085: (4)              dst_color = ll_fz_convert_separation_colors( self.m_internal, src_color,
dst_seps.m_internal, color_params.internal())
43086: (4)              return dst_color
43087: (0)          FzColorspace.fz_convert_separation_colors =
FzColorspace_fz_convert_separation_colors_outparams_fn
43088: (0)          def FzCompressedBuffer_fz_open_image_decomp_stream_from_buffer_outparams_fn(
self):
43089: (4)              """
43090: (4)              Helper for out-params of class method
fz_compressed_buffer::ll_fz_open_image_decomp_stream_from_buffer()
[fz_open_image_decomp_stream_from_buffer()].
43091: (4)              """
43092: (4)              ret, l2factor = ll_fz_open_image_decomp_stream_from_buffer(
self.m_internal)
43093: (4)              return FzStream(ret), l2factor
43094: (0)          FzCompressedBuffer.fz_open_image_decomp_stream_from_buffer =
FzCompressedBuffer_fz_open_image_decomp_stream_from_buffer_outparams_fn
43095: (0)          def FzDisplayList_fz_search_display_list_outparams_fn( self, needle, hit_bbox,
hit_max):
43096: (4)              """
43097: (4)              Helper for out-params of class method
fz_display_list::ll_fz_search_display_list() [fz_search_display_list()].
43098: (4)              """
43099: (4)              ret, hit_mark = ll_fz_search_display_list( self.m_internal, needle,
hit_bbox.internal(), hit_max)
43100: (4)              return ret, hit_mark
43101: (0)          FzDisplayList.fz_search_display_list =
FzDisplayList_fz_search_display_list_outparams_fn
43102: (0)          def FzDocument_fz_resolve_link_outparams_fn( self, uri):
43103: (4)              """
43104: (4)              Helper for out-params of class method fz_document::ll_fz_resolve_link()
[fz_resolve_link()].
43105: (4)              """
43106: (4)              ret, xp, yp = ll_fz_resolve_link( self.m_internal, uri)
43107: (4)              return FzLocation(ret), xp, yp
43108: (0)          FzDocument.fz_resolve_link = FzDocument_fz_resolve_link_outparams_fn
```

```
43109: (0)          def FzDocument_fz_search_chapter_page_number_outparams_fn( self, chapter,
page, needle, hit_bbox, hit_max):
43110: (4)              """
43111: (4)              Helper for out-params of class method
fz_document::ll_fz_search_chapter_page_number() [fz_search_chapter_page_number()].
43112: (4)              """
43113: (4)              ret, hit_mark = ll_fz_search_chapter_page_number( self.m_internal,
chapter, page, needle, hit_bbox.internal(), hit_max)
43114: (4)              return ret, hit_mark
43115: (0)          FzDocument.fz_search_chapter_page_number =
FzDocument_fz_search_chapter_page_number_outparams_fn
43116: (0)          def FzDocument_fz_search_page_number_outparams_fn( self, number, needle,
hit_bbox, hit_max):
43117: (4)              """
43118: (4)              Helper for out-params of class method
fz_document::ll_fz_search_page_number() [fz_search_page_number()].
43119: (4)              """
43120: (4)              ret, hit_mark = ll_fz_search_page_number( self.m_internal, number, needle,
hit_bbox.internal(), hit_max)
43121: (4)              return ret, hit_mark
43122: (0)          FzDocument.fz_search_page_number =
FzDocument_fz_search_page_number_outparams_fn
43123: (0)          def FzFont_fz_encode_character_with_fallback_outparams_fn( self, unicode,
script, language):
43124: (4)              """
43125: (4)              Helper for out-params of class method
fz_font::ll_fz_encode_character_with_fallback() [fz_encode_character_with_fallback()].
43126: (4)              """
43127: (4)              ret, out_font = ll_fz_encode_character_with_fallback( self.m_internal,
unicode, script, language)
43128: (4)              return ret, FzFont(ll_fz_keep_font( out_font))
43129: (0)          FzFont.fz_encode_character_with_fallback =
FzFont_fz_encode_character_with_fallback_outparams_fn
43130: (0)          def FzFunction_fz_eval_function_outparams_fn( self, in_, inlen, outlen):
43131: (4)              """
43132: (4)              Helper for out-params of class method fz_function::ll_fz_eval_function()
[fz_eval_function()].
43133: (4)              """
43134: (4)              out = ll_fz_eval_function( self.m_internal, in_, inlen, outlen)
43135: (4)              return out
43136: (0)          FzFunction.fz_eval_function = FzFunction_fz_eval_function_outparams_fn
43137: (0)          def FzImage_fz_get_pixmap_from_image_outparams_fn( self, subarea, ctm):
43138: (4)              """
43139: (4)              Helper for out-params of class method
fz_image::ll_fz_get_pixmap_from_image() [fz_get_pixmap_from_image()].
43140: (4)              """
43141: (4)              ret, w, h = ll_fz_get_pixmap_from_image( self.m_internal,
subarea.internal(), ctm.internal())
43142: (4)              return FzPixmap(ret), w, h
43143: (0)          FzImage.fz_get_pixmap_from_image =
FzImage_fz_get_pixmap_from_image_outparams_fn
43144: (0)          def FzImage_fz_image_resolution_outparams_fn( self):
43145: (4)              """
43146: (4)              Helper for out-params of class method fz_image::ll_fz_image_resolution()
[fz_image_resolution()].
43147: (4)              """
43148: (4)              xres, yres = ll_fz_image_resolution( self.m_internal)
43149: (4)              return xres, yres
43150: (0)          FzImage.fz_image_resolution = FzImage_fz_image_resolution_outparams_fn
43151: (0)          def FzOutput_fz_new_svg_device_with_id_outparams_fn( self, page_width,
page_height, text_format, reuse_images):
43152: (4)              """
43153: (4)              Helper for out-params of class method
fz_output::ll_fz_new_svg_device_with_id() [fz_new_svg_device_with_id()].
43154: (4)              """
43155: (4)              ret, id = ll_fz_new_svg_device_with_id( self.m_internal, page_width,
page_height, text_format, reuse_images)
43156: (4)              return FzDevice(ret), id
43157: (0)          FzOutput.fz_new_svg_device_with_id =
```

```
FzOutput_fz_new_svg_device_with_id_outparams_fn
43158: (0)          def FzOutput_pdf_print_encrypted_obj_outparams_fn( self, obj, tight, ascii,
crypt, num, gen):
43159: (4)              """
43160: (4)              Helper for out-params of class method
fz_output::ll_pdf_print_encrypted_obj() [pdf_print_encrypted_obj()].
43161: (4)              """
43162: (4)              sep = ll_pdf_print_encrypted_obj( self.m_internal, obj.m_internal, tight,
ascii, crypt.m_internal, num, gen)
43163: (4)              return sep
43164: (0)          FzOutput.pdf_print_encrypted_obj =
FzOutput_pdf_print_encrypted_obj_outparams_fn
43165: (0)          def FzPage_fz_page_presentation_outparams_fn( self, transition):
43166: (4)              """
43167: (4)              Helper for out-params of class method fz_page::ll_fz_page_presentation()
[fz_page_presentation()].
43168: (4)              """
43169: (4)              ret, duration = ll_fz_page_presentation( self.m_internal,
transition.internal())
43170: (4)              return FzTransition(ret), duration
43171: (0)          FzPage.fz_page_presentation = FzPage_fz_page_presentation_outparams_fn
43172: (0)          def FzPage_fz_search_page_outparams_fn( self, needle, hit_bbox, hit_max):
43173: (4)              """
43174: (4)              Helper for out-params of class method fz_page::ll_fz_search_page()
[fz_search_page()].
43175: (4)              """
43176: (4)              ret, hit_mark = ll_fz_search_page( self.m_internal, needle,
hit_bbox.internal(), hit_max)
43177: (4)              return ret, hit_mark
43178: (0)          FzPage.fz_search_page = FzPage_fz_search_page_outparams_fn
43179: (0)          def FzPixmap_fz_fill_pixmap_with_color_outparams_fn( self, colorspace,
color_params):
43180: (4)              """
43181: (4)              Helper for out-params of class method
fz_pixmap::ll_fz_fill_pixmap_with_color() [fz_fill_pixmap_with_color()].
43182: (4)              """
43183: (4)              color = ll_fz_fill_pixmap_with_color( self.m_internal,
colorspace.m_internal, color_params.internal())
43184: (4)              return color
43185: (0)          FzPixmap.fz_fill_pixmap_with_color =
FzPixmap_fz_fill_pixmap_with_color_outparams_fn
43186: (0)          def FzSeparations_fz_separation_equivalent_outparams_fn( self, idx, dst_cs,
prf, color_params):
43187: (4)              """
43188: (4)              Helper for out-params of class method
fz_separations::ll_fz_separation_equivalent() [fz_separation_equivalent()].
43189: (4)              """
43190: (4)              dst_color = ll_fz_separation_equivalent( self.m_internal, idx,
dst_cs.m_internal, prf.m_internal, color_params.internal())
43191: (4)              return dst_color
43192: (0)          FzSeparations.fz_separation_equivalent =
FzSeparations_fz_separation_equivalent_outparams_fn
43193: (0)          def FzShade_fz_paint_shade_outparams_fn( self, override_cs, ctm, dest,
color_params, bbox, eop):
43194: (4)              """
43195: (4)              Helper for out-params of class method fz_shade::ll_fz_paint_shade()
[fz_paint_shade()].
43196: (4)              """
43197: (4)              cache = ll_fz_paint_shade( self.m_internal, override_cs.m_internal,
ctm.internal(), dest.m_internal, color_params.internal(), bbox.internal(), eop.m_internal)
43198: (4)              return FzShadeColorCache(ll_fz_keep_shade_color_cache( cache))
43199: (0)          FzShade.fz_paint_shade = FzShade_fz_paint_shade_outparams_fn
43200: (0)          def FzStextPage_fz_search_stext_page_outparams_fn( self, needle, hit_bbox,
hit_max):
43201: (4)              """
43202: (4)              Helper for out-params of class method
fz_stext_page::ll_fz_search_stext_page() [fz_search_stext_page()].
43203: (4)              """
43204: (4)              ret, hit_mark = ll_fz_search_stext_page( self.m_internal, needle,
```

```
hit_bbox.internal(), hit_max)
43205: (4)              return ret, hit_mark
43206: (0)          FzStextPage.fz_search_stext_page =
FzStextPage_fz_search_stext_page_outparams_fn
43207: (0)          def FzStream_fz_decomp_image_from_stream_outparams_fn( self, image, subarea,
indexed, l2factor):
43208: (4)              """
43209: (4)              Helper for out-params of class method
fz_stream::ll_fz_decomp_image_from_stream() [fz_decomp_image_from_stream()].
43210: (4)              """
43211: (4)              ret, l2extra = ll_fz_decomp_image_from_stream( self.m_internal,
image.m_internal, subarea.internal(), indexed, l2factor)
43212: (4)              return FzPixmap(ret), l2extra
43213: (0)          FzStream.fz_decomp_image_from_stream =
FzStream_fz_decomp_image_from_stream_outparams_fn
43214: (0)          def FzStream_fz_open_image_decomp_stream_outparams_fn( self, arg_1):
43215: (4)              """
43216: (4)              Helper for out-params of class method
fz_stream::ll_fz_open_image_decomp_stream() [fz_open_image_decomp_stream()].
43217: (4)              """
43218: (4)              ret, l2factor = ll_fz_open_image_decomp_stream( self.m_internal,
arg_1.m_internal)
43219: (4)              return FzStream(ret), l2factor
43220: (0)          FzStream.fz_open_image_decomp_stream =
FzStream_fz_open_image_decomp_stream_outparams_fn
43221: (0)          def FzStream_fz_read_best_outparams_fn( self, initial, worst_case):
43222: (4)              """
43223: (4)              Helper for out-params of class method fz_stream::ll_fz_read_best()
[fz_read_best()].
43224: (4)              """
43225: (4)              ret, truncated = ll_fz_read_best( self.m_internal, initial, worst_case)
43226: (4)              return FzBuffer(ret), truncated
43227: (0)          FzStream.fz_read_best = FzStream_fz_read_best_outparams_fn
43228: (0)          def FzXml_fz_dom_get_attribute_outparams_fn( self, i):
43229: (4)              """
43230: (4)              Helper for out-params of class method fz_xml::ll_fz_dom_get_attribute()
[fz_dom_get_attribute()].
43231: (4)              """
43232: (4)              ret, att = ll_fz_dom_get_attribute( self.m_internal, i)
43233: (4)              return ret, att
43234: (0)          FzXml.fz_dom_get_attribute = FzXml_fz_dom_get_attribute_outparams_fn
43235: (0)          def FzXml_fz_new_display_list_from_svg_xml_outparams_fn( self, xmldoc,
base_uri, dir):
43236: (4)              """
43237: (4)              Helper for out-params of class method
fz_xml::ll_fz_new_display_list_from_svg_xml() [fz_new_display_list_from_svg_xml()].
43238: (4)              """
43239: (4)              ret, w, h = ll_fz_new_display_list_from_svg_xml( self.m_internal,
xmldoc.m_internal, base_uri, dir.m_internal)
43240: (4)              return FzDisplayList(ret), w, h
43241: (0)          FzXml.fz_new_display_list_from_svg_xml =
FzXml_fz_new_display_list_from_svg_xml_outparams_fn
43242: (0)          def PdfAnnot_pdf_annot_MK_BC_outparams_fn( self, color):
43243: (4)              """
43244: (4)              Helper for out-params of class method pdf_annot::ll_pdf_annot_MK_BC()
[pdf_annot_MK_BC()].
43245: (4)              """
43246: (4)              n = ll_pdf_annot_MK_BC( self.m_internal, color)
43247: (4)              return n
43248: (0)          PdfAnnot.pdf_annot_MK_BC = PdfAnnot_pdf_annot_MK_BC_outparams_fn
43249: (0)          def PdfAnnot_pdf_annot_MK_BG_outparams_fn( self, color):
43250: (4)              """
43251: (4)              Helper for out-params of class method pdf_annot::ll_pdf_annot_MK_BG()
[pdf_annot_MK_BG()].
43252: (4)              """
43253: (4)              n = ll_pdf_annot_MK_BG( self.m_internal, color)
43254: (4)              return n
43255: (0)          PdfAnnot.pdf_annot_MK_BG = PdfAnnot_pdf_annot_MK_BG_outparams_fn
43256: (0)          def PdfAnnot_pdf_annot_color_outparams_fn( self, color):
```

```
43257: (4)                """
43258: (4)                Helper for out-params of class method pdf_annot::ll_pdf_annot_color()
[pdf_annot_color()].
43259: (4)                """
43260: (4)                n = ll_pdf_annot_color( self.m_internal, color)
43261: (4)                return n
43262: (0)            PdfAnnot.pdf_annot_color = PdfAnnot_pdf_annot_color_outparams_fn
43263: (0)            def PdfAnnot_pdf_annot_default_appearance_outparams_fn( self, color):
43264: (4)                """
43265: (4)                Helper for out-params of class method
pdf_annot::ll_pdf_annot_default_appearance() [pdf_annot_default_appearance()].
43266: (4)                """
43267: (4)                font, size, n = ll_pdf_annot_default_appearance( self.m_internal, color)
43268: (4)                return font, size, n
43269: (0)            PdfAnnot.pdf_annot_default_appearance =
PdfAnnot_pdf_annot_default_appearance_outparams_fn
43270: (0)            def PdfAnnot_pdf_annot_interior_color_outparams_fn( self, color):
43271: (4)                """
43272: (4)                Helper for out-params of class method
pdf_annot::ll_pdf_annot_interior_color() [pdf_annot_interior_color()].
43273: (4)                """
43274: (4)                n = ll_pdf_annot_interior_color( self.m_internal, color)
43275: (4)                return n
43276: (0)            PdfAnnot.pdf_annot_interior_color =
PdfAnnot_pdf_annot_interior_color_outparams_fn
43277: (0)            def PdfAnnot_pdf_annot_line_ending_styles_outparams_fn( self):
43278: (4)                """
43279: (4)                Helper for out-params of class method
pdf_annot::ll_pdf_annot_line_ending_styles() [pdf_annot_line_ending_styles()].
43280: (4)                """
43281: (4)                start_style, end_style = ll_pdf_annot_line_ending_styles( self.m_internal)
43282: (4)                return start_style, end_style
43283: (0)            PdfAnnot.pdf_annot_line_ending_styles =
PdfAnnot_pdf_annot_line_ending_styles_outparams_fn
43284: (0)            def PdfAnnot_pdf_edit_text_field_value_outparams_fn( self, value, change):
43285: (4)                """
43286: (4)                Helper for out-params of class method
pdf_annot::ll_pdf_edit_text_field_value() [pdf_edit_text_field_value()].
43287: (4)                """
43288: (4)                ret, selStart, selEnd, newvalue = ll_pdf_edit_text_field_value(
self.m_internal, value, change)
43289: (4)                return ret, selStart, selEnd, newvalue
43290: (0)            PdfAnnot.pdf_edit_text_field_value =
PdfAnnot_pdf_edit_text_field_value_outparams_fn
43291: (0)            def PdfCmap_pdf_decode_cmap_outparams_fn( self, s, e):
43292: (4)                """
43293: (4)                Helper for out-params of class method pdf_cmap::ll_pdf_decode_cmap()
[pdf_decode_cmap()].
43294: (4)                """
43295: (4)                ret, cpt = ll_pdf_decode_cmap( self.m_internal, s, e)
43296: (4)                return ret, cpt
43297: (0)            PdfCmap.pdf_decode_cmap = PdfCmap_pdf_decode_cmap_outparams_fn
43298: (0)            def PdfCmap_pdf_lookup_cmap_full_outparams_fn( self, cpt):
43299: (4)                """
43300: (4)                Helper for out-params of class method pdf_cmap::ll_pdf_lookup_cmap_full()
[pdf_lookup_cmap_full()].
43301: (4)                """
43302: (4)                ret, out = ll_pdf_lookup_cmap_full( self.m_internal, cpt)
43303: (4)                return ret, out
43304: (0)            PdfCmap.pdf_lookup_cmap_full = PdfCmap_pdf_lookup_cmap_full_outparams_fn
43305: (0)            def PdfCmap_pdf_map_one_to_many_outparams_fn( self, one, len):
43306: (4)                """
43307: (4)                Helper for out-params of class method pdf_cmap::ll_pdf_map_one_to_many()
[pdf_map_one_to_many()].
43308: (4)                """
43309: (4)                many = ll_pdf_map_one_to_many( self.m_internal, one, len)
43310: (4)                return many
43311: (0)            PdfCmap.pdf_map_one_to_many = PdfCmap_pdf_map_one_to_many_outparams_fn
43312: (0)            def PdfDocument_pdf_count_q_balance_outparams_fn( self, res, stm):
```

```
43313: (4)              """
43314: (4)              Helper for out-params of class method
pdf_document::ll_pdf_count_q_balance() [pdf_count_q_balance()].
43315: (4)              """
43316: (4)              prepend, append = ll_pdf_count_q_balance( self.m_internal, res.m_internal,
stm.m_internal)
43317: (4)              return prepend, append
43318: (0)          PdfDocument.pdf_count_q_balance = PdfDocument_pdf_count_q_balance_outparams_fn
43319: (0)          def PdfDocument_pdf_field_event_validate_outparams_fn( self, field, value):
43320: (4)              """
43321: (4)              Helper for out-params of class method
pdf_document::ll_pdf_field_event_validate() [pdf_field_event_validate()].
43322: (4)              """
43323: (4)              ret, newvalue = ll_pdf_field_event_validate( self.m_internal,
field.m_internal, value)
43324: (4)              return ret, newvalue
43325: (0)          PdfDocument.pdf_field_event_validate =
PdfDocument_pdf_field_event_validate_outparams_fn
43326: (0)          def PdfDocument_pdf_load_to_unicode_outparams_fn( self, font, collection,
cmapstm):
43327: (4)              """
43328: (4)              Helper for out-params of class method
pdf_document::ll_pdf_load_to_unicode() [pdf_load_to_unicode()].
43329: (4)              """
43330: (4)              strings = ll_pdf_load_to_unicode( self.m_internal, font.m_internal,
collection, cmapstm.m_internal)
43331: (4)              return strings
43332: (0)          PdfDocument.pdf_load_to_unicode = PdfDocument_pdf_load_to_unicode_outparams_fn
43333: (0)          def PdfDocument_pdf_lookup_page_loc_outparams_fn( self, needle):
43334: (4)              """
43335: (4)              Helper for out-params of class method
pdf_document::ll_pdf_lookup_page_loc() [pdf_lookup_page_loc()].
43336: (4)              """
43337: (4)              ret, parentp, indexp = ll_pdf_lookup_page_loc( self.m_internal, needle)
43338: (4)              return PdfObj( ll_pdf_keep_obj( ret)), PdfObj(ll_pdf_keep_obj( parentp)),
indexp
43339: (0)          PdfDocument.pdf_lookup_page_loc = PdfDocument_pdf_lookup_page_loc_outparams_fn
43340: (0)          def PdfDocument_pdf_page_write_outparams_fn( self, mediabox):
43341: (4)              """
43342: (4)              Helper for out-params of class method pdf_document::ll_pdf_page_write()
[pdf_page_write()].
43343: (4)              """
43344: (4)              ret, presources, pcontents = ll_pdf_page_write( self.m_internal,
mediabox.internal())
43345: (4)              return FzDevice(ret), PdfObj( presources), FzBuffer( pcontents)
43346: (0)          PdfDocument.pdf_page_write = PdfDocument_pdf_page_write_outparams_fn
43347: (0)          def PdfDocument_pdf_parse_ind_obj_outparams_fn( self, f):
43348: (4)              """
43349: (4)              Helper for out-params of class method pdf_document::ll_pdf_parse_ind_obj()
[pdf_parse_ind_obj()].
43350: (4)              """
43351: (4)              ret, num, gen, stm_ofs, try_repair = ll_pdf_parse_ind_obj(
self.m_internal, f.m_internal)
43352: (4)              return PdfObj(ret), num, gen, stm_ofs, try_repair
43353: (0)          PdfDocument.pdf_parse_ind_obj = PdfDocument_pdf_parse_ind_obj_outparams_fn
43354: (0)          def PdfDocument_pdf_parse_journal_obj_outparams_fn( self, stm):
43355: (4)              """
43356: (4)              Helper for out-params of class method
pdf_document::ll_pdf_parse_journal_obj() [pdf_parse_journal_obj()].
43357: (4)              """
43358: (4)              ret, onum, ostm, newobj = ll_pdf_parse_journal_obj( self.m_internal,
stm.m_internal)
43359: (4)              return PdfObj(ret), onum, FzBuffer( ostm), newobj
43360: (0)          PdfDocument.pdf_parse_journal_obj =
PdfDocument_pdf_parse_journal_obj_outparams_fn
43361: (0)          def PdfDocument_pdf_repair_obj_outparams_fn( self, buf):
43362: (4)              """
43363: (4)              Helper for out-params of class method pdf_document::ll_pdf_repair_obj()
[pdf_repair_obj()].
```

```
43364: (4)                """
43365: (4)                ret, stmofsp, stmlenp, encrypt, id, page, tmpofs, root =
ll_pdf_repair_obj( self.m_internal, buf.m_internal)
43366: (4)                return ret, stmofsp, stmlenp, PdfObj(ll_pdf_keep_obj( encrypt)),
PdfObj(ll_pdf_keep_obj( id)), PdfObj(ll_pdf_keep_obj( page)), tmpofs, PdfObj(ll_pdf_keep_obj(
root))
43367: (0)          PdfDocument.pdf_repair_obj = PdfDocument_pdf_repair_obj_outparams_fn
43368: (0)          def PdfDocument_pdf_resolve_link_outparams_fn( self, uri):
43369: (4)              """
43370: (4)              Helper for out-params of class method pdf_document::ll_pdf_resolve_link()
[pdf_resolve_link()].
43371: (4)              """
43372: (4)              ret, xp, yp = ll_pdf_resolve_link( self.m_internal, uri)
43373: (4)              return ret, xp, yp
43374: (0)          PdfDocument.pdf_resolve_link = PdfDocument_pdf_resolve_link_outparams_fn
43375: (0)          def PdfDocument_pdf_signature_contents_outparams_fn( self, signature):
43376: (4)              """
43377: (4)              Helper for out-params of class method
pdf_document::ll_pdf_signature_contents() [pdf_signature_contents()].
43378: (4)              """
43379: (4)              ret, contents = ll_pdf_signature_contents( self.m_internal,
signature.m_internal)
43380: (4)              return ret, contents
43381: (0)          PdfDocument.pdf_signature_contents =
PdfDocument_pdf_signature_contents_outparams_fn
43382: (0)          def PdfDocument_pdf_undoredo_state_outparams_fn( self):
43383: (4)              """
43384: (4)              Helper for out-params of class method
pdf_document::ll_pdf_undoredo_state() [pdf_undoredo_state()].
43385: (4)              """
43386: (4)              ret, steps = ll_pdf_undoredo_state( self.m_internal)
43387: (4)              return ret, steps
43388: (0)          PdfDocument.pdf_undoredo_state = PdfDocument_pdf_undoredo_state_outparams_fn
43389: (0)          def PdfFunction_pdf_eval_function_outparams_fn( self, in_, inlen, outlen):
43390: (4)              """
43391: (4)              Helper for out-params of class method pdf_function::ll_pdf_eval_function()
[pdf_eval_function()].
43392: (4)              """
43393: (4)              out = ll_pdf_eval_function( self.m_internal, in_, inlen, outlen)
43394: (4)              return out
43395: (0)          PdfFunction.pdf_eval_function = PdfFunction_pdf_eval_function_outparams_fn
43396: (0)          def PdfJs_pdf_js_event_result_validate_outparams_fn( self):
43397: (4)              """
43398: (4)              Helper for out-params of class method
pdf_js::ll_pdf_js_event_result_validate() [pdf_js_event_result_validate()].
43399: (4)              """
43400: (4)              ret, newvalue = ll_pdf_js_event_result_validate( self.m_internal)
43401: (4)              return ret, newvalue
43402: (0)          PdfJs.pdf_js_event_result_validate =
PdfJs_pdf_js_event_result_validate_outparams_fn
43403: (0)          def PdfJs_pdf_js_execute_outparams_fn( self, name, code):
43404: (4)              """
43405: (4)              Helper for out-params of class method pdf_js::ll_pdf_js_execute()
[pdf_js_execute()].
43406: (4)              """
43407: (4)              result = ll_pdf_js_execute( self.m_internal, name, code)
43408: (4)              return result
43409: (0)          PdfJs.pdf_js_execute = PdfJs_pdf_js_execute_outparams_fn
43410: (0)          def PdfObj_pdf_array_get_string_outparams_fn( self, index):
43411: (4)              """
43412: (4)              Helper for out-params of class method pdf_obj::ll_pdf_array_get_string()
[pdf_array_get_string()].
43413: (4)              """
43414: (4)              ret, sizep = ll_pdf_array_get_string( self.m_internal, index)
43415: (4)              return ret, sizep
43416: (0)          PdfObj.pdf_array_get_string = PdfObj_pdf_array_get_string_outparams_fn
43417: (0)          def PdfObj_pdf_dict_get_inheritable_string_outparams_fn( self):
43418: (4)              """
43419: (4)              Helper for out-params of class method
```

```
pdf_obj::ll_pdf_dict_get_inheritable_string() [pdf_dict_get_inheritable_string()].
43420: (4)                """
43421: (4)                ret, sizep = ll_pdf_dict_get_inheritable_string( self.m_internal)
43422: (4)                return ret, sizep
43423: (0)            PdfObj.pdf_dict_get_inheritable_string =
PdfObj_pdf_dict_get_inheritable_string_outparams_fn
43424: (0)            def PdfObj_pdf_dict_get_string_outparams_fn( self):
43425: (4)                """
43426: (4)                Helper for out-params of class method pdf_obj::ll_pdf_dict_get_string()
[pdf_dict_get_string()].
43427: (4)                """
43428: (4)                ret, sizep = ll_pdf_dict_get_string( self.m_internal)
43429: (4)                return ret, sizep
43430: (0)            PdfObj.pdf_dict_get_string = PdfObj_pdf_dict_get_string_outparams_fn
43431: (0)            def PdfObj_pdf_obj_memo_outparams_fn( self, bit):
43432: (4)                """
43433: (4)                Helper for out-params of class method pdf_obj::ll_pdf_obj_memo()
[pdf_obj_memo()].
43434: (4)                """
43435: (4)                ret, memo = ll_pdf_obj_memo( self.m_internal, bit)
43436: (4)                return ret, memo
43437: (0)            PdfObj.pdf_obj_memo = PdfObj_pdf_obj_memo_outparams_fn
43438: (0)            def PdfObj_pdf_to_string_outparams_fn( self):
43439: (4)                """
43440: (4)                Helper for out-params of class method pdf_obj::ll_pdf_to_string()
[pdf_to_string()].
43441: (4)                """
43442: (4)                ret, sizep = ll_pdf_to_string( self.m_internal)
43443: (4)                return ret, sizep
43444: (0)            PdfObj.pdf_to_string = PdfObj_pdf_to_string_outparams_fn
43445: (0)            def PdfObj_pdf_walk_tree_outparams_fn( self, arrive, leave, arg):
43446: (4)                """
43447: (4)                Helper for out-params of class method pdf_obj::ll_pdf_walk_tree()
[pdf_walk_tree()].
43448: (4)                """
43449: (4)                names, values = ll_pdf_walk_tree( self.m_internal, arrive, leave, arg)
43450: (4)                return PdfObj(ll_pdf_keep_obj( names)), PdfObj(ll_pdf_keep_obj( values))
43451: (0)            PdfObj.pdf_walk_tree = PdfObj_pdf_walk_tree_outparams_fn
43452: (0)            def PdfPage_pdf_page_presentation_outparams_fn( self, transition):
43453: (4)                """
43454: (4)                Helper for out-params of class method pdf_page::ll_pdf_page_presentation()
[pdf_page_presentation()].
43455: (4)                """
43456: (4)                ret, duration = ll_pdf_page_presentation( self.m_internal,
transition.internal())
43457: (4)                return FzTransition(ret), duration
43458: (0)            PdfPage.pdf_page_presentation = PdfPage_pdf_page_presentation_outparams_fn
43459: (0)            def PdfProcessor_pdf_process_contents_outparams_fn( self, doc, res, stm,
cookie):
43460: (4)                """
43461: (4)                Helper for out-params of class method
pdf_processor::ll_pdf_process_contents() [pdf_process_contents()].
43462: (4)                """
43463: (4)                out_res = ll_pdf_process_contents( self.m_internal, doc.m_internal,
res.m_internal, stm.m_internal, cookie.m_internal)
43464: (4)                return PdfObj(ll_pdf_keep_obj( out_res))
43465: (0)            PdfProcessor.pdf_process_contents =
PdfProcessor_pdf_process_contents_outparams_fn
43466: (0)            FzErrorBase.__str__ = lambda self: self.what()
43467: (0)            FzErrorNone.__str__ = lambda self: self.what()
43468: (0)            FzErrorGeneric.__str__ = lambda self: self.what()
43469: (0)            FzErrorSystem.__str__ = lambda self: self.what()
43470: (0)            FzErrorLibrary.__str__ = lambda self: self.what()
43471: (0)            FzErrorArgument.__str__ = lambda self: self.what()
43472: (0)            FzErrorLimit.__str__ = lambda self: self.what()
43473: (0)            FzErrorUnsupported.__str__ = lambda self: self.what()
43474: (0)            FzErrorFormat.__str__ = lambda self: self.what()
43475: (0)            FzErrorSyntax.__str__ = lambda self: self.what()
43476: (0)            FzErrorTrylater.__str__ = lambda self: self.what()
```

```
43477: (0)          FzErrorAbort.__str__ = lambda self: self.what()
43478: (0)          FzErrorRepaired.__str__ = lambda self: self.what()
43479: (0)          internal_set_error_classes([
43480: (8)                  FzErrorNone,
43481: (8)                  FzErrorGeneric,
43482: (8)                  FzErrorSystem,
43483: (8)                  FzErrorLibrary,
43484: (8)                  FzErrorArgument,
43485: (8)                  FzErrorLimit,
43486: (8)                  FzErrorUnsupported,
43487: (8)                  FzErrorFormat,
43488: (8)                  FzErrorSyntax,
43489: (8)                  FzErrorTrylater,
43490: (8)                  FzErrorAbort,
43491: (8)                  FzErrorRepaired,
43492: (0)          FzErrorBase,
43493: (0)          ])
43494: (0)          ll_fz_parse_page_range_orig = ll_fz_parse_page_range
43495: (0)          def ll_fz_parse_page_range(s, n):
43496: (4)              ret = ll_fz_parse_page_range_orig(s, n)
43497: (4)              if len(ret) == 2:
43498: (8)                  return None, 0, 0
43499: (4)              else:
43500: (8)                  return ret[0], ret[1], ret[2]
43501: (0)          fz_parse_page_range = ll_fz_parse_page_range
43502: (0)          def ll_fz_format_output_path( format, page):
43503: (4)              m = re.search( '(%[0-9]*d)', format)
43504: (4)              if m:
43505: (8)                  ret = format[ :m.start(1)] + str(page) + format[ m.end(1):]
43506: (4)              else:
43507: (8)                  dot = format.rfind( '.')
43508: (8)                  if dot < 0:
43509: (12)                     dot = len( format)
43510: (8)                  ret = format[:dot] + str(page) + format[dot:]
43511: (4)              return ret
43512: (0)          fz_format_output_path = ll_fz_format_output_path
43513: (0)          class IteratorWrap:
43514: (4)              """
43515: (4)              This is a Python iterator for containers that have C++-style
43516: (4)              begin() and end() methods that return iterators.
43517: (4)              Iterators must have the following methods:
43518: (8)                  __increment__(): move to next item in the container.
43519: (8)                  __ref__(): return reference to item in the container.
43520: (4)              Must also be able to compare two iterators for equality.
43521: (4)              """
43522: (4)              def __init__( self, container):
43523: (8)                  self.container = container
43524: (8)                  self.pos = None
43525: (8)                  self.end = container.end()
43526: (4)              def __iter__( self):
43527: (8)                  return self
43528: (4)              def __next__( self):    # for python2.
43529: (8)                  if self.pos is None:
43530: (12)                     self.pos = self.container.begin()
43531: (8)                  else:
43532: (12)                     self.pos.__increment__()
43533: (8)                  if self.pos == self.end:
43534: (12)                     raise StopIteration()
43535: (8)                  return self.pos.__ref__()
43536: (4)              def next( self):    # for python3.
43537: (8)                  return self.__next__()
43538: (0)          def ll_fz_buffer_extract(buffer):
43539: (4)              """
43540: (4)              Returns buffer data as a Python bytes instance, leaving the
43541: (4)              buffer empty.
43542: (4)              """
43543: (4)              assert isinstance( buffer, fz_buffer)
43544: (4)              return ll_fz_buffer_to_bytes_internal(buffer, clear=1)
43545: (0)          def fz_buffer_extract(buffer):
```

```
43546: (4)              """
43547: (4)              Returns buffer data as a Python bytes instance, leaving the
43548: (4)              buffer empty.
43549: (4)              """
43550: (4)              assert isinstance( buffer, FzBuffer)
43551: (4)              return ll_fz_buffer_extract(buffer.m_internal)
43552: (0)          FzBuffer.fz_buffer_extract = fz_buffer_extract
43553: (0)          def ll_fz_buffer_extract_copy( buffer):
43554: (4)              """
43555: (4)              Returns buffer data as a Python bytes instance, leaving the
43556: (4)              buffer unchanged.
43557: (4)              """
43558: (4)              assert isinstance( buffer, fz_buffer)
43559: (4)              return ll_fz_buffer_to_bytes_internal(buffer, clear=0)
43560: (0)          def fz_buffer_extract_copy( buffer):
43561: (4)              """
43562: (4)              Returns buffer data as a Python bytes instance, leaving the
43563: (4)              buffer unchanged.
43564: (4)              """
43565: (4)              assert isinstance( buffer, FzBuffer)
43566: (4)              return ll_fz_buffer_extract_copy(buffer.m_internal)
43567: (0)          FzBuffer.fz_buffer_extract_copy = fz_buffer_extract_copy
43568: (0)          def fz_buffer_storage_memoryview( buffer, writable=False):
43569: (4)              """
43570: (4)              Returns a read-only or writable Python `memoryview` onto
43571: (4)              `fz_buffer` data. This relies on `buffer` existing and
43572: (4)              not changing size while the `memoryview` is used.
43573: (4)              """
43574: (4)              assert isinstance( buffer, FzBuffer)
43575: (4)              return ll_fz_buffer_storage_memoryview( buffer.m_internal, writable)
43576: (0)          FzBuffer.fz_buffer_storage_memoryview = fz_buffer_storage_memoryview
43577: (0)          ll_fz_new_buffer_from_copied_data_orig = ll_fz_new_buffer_from_copied_data
43578: (0)          def ll_fz_new_buffer_from_copied_data(data):
43579: (4)              """
43580: (4)              Returns fz_buffer containing copy of `data`, which should
43581: (4)              be a `bytes` or similar Python buffer instance.
43582: (4)              """
43583: (4)              buffer_ = ll_fz_new_buffer_from_copied_data_orig(python_buffer_data(data),
len(data))
43584: (4)              return buffer_
43585: (0)          def fz_new_buffer_from_copied_data(data):
43586: (4)              """
43587: (4)              Returns FzBuffer containing copy of `data`, which should be
43588: (4)              a `bytes` or similar Python buffer instance.
43589: (4)              """
43590: (4)              return FzBuffer( ll_fz_new_buffer_from_copied_data( data))
43591: (0)          FzBuffer.fz_new_buffer_from_copied_data = fz_new_buffer_from_copied_data
43592: (0)          def ll_pdf_dict_getl(obj, *tail):
43593: (4)              """
43594: (4)              Python implementation of ll_pdf_dict_getl(), because SWIG
43595: (4)              doesn't handle variadic args. Each item in `tail` should be
43596: (4)              `mupdf.pdf_obj`.
43597: (4)              """
43598: (4)              for key in tail:
43599: (8)                  if not obj:
43600: (12)                     break
43601: (8)                  obj = ll_pdf_dict_get(obj, key)
43602: (4)              assert isinstance(obj, pdf_obj)
43603: (4)              return obj
43604: (0)          def pdf_dict_getl(obj, *tail):
43605: (4)              """
43606: (4)              Python implementation of pdf_dict_getl(), because SWIG
43607: (4)              doesn't handle variadic args. Each item in `tail` should be
43608: (4)              a `mupdf.PdfObj`.
43609: (4)              """
43610: (4)              for key in tail:
43611: (8)                  if not obj.m_internal:
43612: (12)                     break
43613: (8)                  obj = pdf_dict_get(obj, key)
```

```
43614: (4)              assert isinstance(obj, PdfObj)
43615: (4)              return obj
43616: (0)          PdfObj.pdf_dict_getl = pdf_dict_getl
43617: (0)          def ll_pdf_dict_putl(obj, val, *tail):
43618: (4)              """
43619: (4)              Python implementation of ll_pdf_dict_putl() because SWIG
43620: (4)              doesn't handle variadic args. Each item in `tail` should
43621: (4)              be a SWIG wrapper for a `pdf_obj`.
43622: (4)              """
43623: (4)              if ll_pdf_is_indirect( obj):
43624: (8)                  obj = ll_pdf_resolve_indirect_chain( obj)
43625: (4)              if not pdf_is_dict( obj):
43626: (8)                  raise Exception(f'not a dict: {obj}')
43627: (4)              if not tail:
43628: (8)                  return
43629: (4)              doc = ll_pdf_get_bound_document( obj)
43630: (4)              for i, key in enumerate( tail[:-1]):
43631: (8)                  assert isinstance( key, PdfObj), f'Item {i} in `tail` should be a
pdf_obj but is a {type(key)}.'
43632: (8)                  next_obj = ll_pdf_dict_get( obj, key)
43633: (8)                  if not next_obj:
43634: (12)                     next_obj = ll_pdf_new_dict( doc, 1)
43635: (12)                     ll_pdf_dict_put( obj, key, next_obj)
43636: (8)                  obj = next_obj
43637: (4)              key = tail[-1]
43638: (4)              ll_pdf_dict_put( obj, key, val)
43639: (0)          def pdf_dict_putl(obj, val, *tail):
43640: (4)              """
43641: (4)              Python implementation of pdf_dict_putl(fz_context *ctx,
43642: (4)              pdf_obj *obj, pdf_obj *val, ...) because SWIG doesn't
43643: (4)              handle variadic args. Each item in `tail` should
43644: (4)              be a SWIG wrapper for a `PdfObj`.
43645: (4)              """
43646: (4)              if pdf_is_indirect( obj):
43647: (8)                  obj = pdf_resolve_indirect_chain( obj)
43648: (4)              if not pdf_is_dict( obj):
43649: (8)                  raise Exception(f'not a dict: {obj}')
43650: (4)              if not tail:
43651: (8)                  return
43652: (4)              doc = pdf_get_bound_document( obj)
43653: (4)              for i, key in enumerate( tail[:-1]):
43654: (8)                  assert isinstance( key, PdfObj), f'item {i} in `tail` should be a
PdfObj but is a {type(key)}.'
43655: (8)                  next_obj = pdf_dict_get( obj, key)
43656: (8)                  if not next_obj.m_internal:
43657: (12)                     next_obj = pdf_new_dict( doc, 1)
43658: (12)                     pdf_dict_put( obj, key, next_obj)
43659: (8)                  obj = next_obj
43660: (4)              key = tail[-1]
43661: (4)              pdf_dict_put( obj, key, val)
43662: (0)          PdfObj.pdf_dict_putl = pdf_dict_putl
43663: (0)          def pdf_dict_putl_drop(obj, *tail):
43664: (4)              raise Exception('mupdf.pdf_dict_putl_drop() is unsupported and unnecessary
in Python because reference counting is automatic. Instead use mupdf.pdf_dict_putl().')
43665: (0)          PdfObj.pdf_dict_putl_drop = pdf_dict_putl_drop
43666: (0)          def ll_pdf_set_annot_color(annot, color):
43667: (4)              """
43668: (4)              Low-level Python implementation of pdf_set_annot_color()
43669: (4)              using ll_pdf_set_annot_color2().
43670: (4)              """
43671: (4)              if isinstance(color, float):
43672: (8)                  ll_pdf_set_annot_color2(annot, 1, color, 0, 0, 0)
43673: (4)              elif len(color) == 1:
43674: (8)                  ll_pdf_set_annot_color2(annot, 1, color[0], 0, 0, 0)
43675: (4)              elif len(color) == 2:
43676: (8)                  ll_pdf_set_annot_color2(annot, 2, color[0], color[1], 0, 0)
43677: (4)              elif len(color) == 3:
43678: (8)                  ll_pdf_set_annot_color2(annot, 3, color[0], color[1], color[2], 0)
43679: (4)              elif len(color) == 4:
```

```
43680: (8)                  ll_pdf_set_annot_color2(annot, 4, color[0], color[1], color[2],
color[3])
43681: (4)              else:
43682: (8)                  raise Exception( f'Unexpected color should be float or list of 1-4
floats: {color}')
43683: (0)          def pdf_set_annot_color(self, color):
43684: (4)              return ll_pdf_set_annot_color(self.m_internal, color)
43685: (0)          PdfAnnot.pdf_set_annot_color = pdf_set_annot_color
43686: (0)          def ll_pdf_set_annot_interior_color(annot, color):
43687: (4)              """
43688: (4)              Low-level Python version of pdf_set_annot_color() using
43689: (4)              pdf_set_annot_color2().
43690: (4)              """
43691: (4)              if isinstance(color, float):
43692: (8)                  ll_pdf_set_annot_interior_color2(annot, 1, color, 0, 0, 0)
43693: (4)              elif len(color) == 1:
43694: (8)                  ll_pdf_set_annot_interior_color2(annot, 1, color[0], 0, 0, 0)
43695: (4)              elif len(color) == 2:
43696: (8)                  ll_pdf_set_annot_interior_color2(annot, 2, color[0], color[1], 0, 0)
43697: (4)              elif len(color) == 3:
43698: (8)                  ll_pdf_set_annot_interior_color2(annot, 3, color[0], color[1],
color[2], 0)
43699: (4)              elif len(color) == 4:
43700: (8)                  ll_pdf_set_annot_interior_color2(annot, 4, color[0], color[1],
color[2], color[3])
43701: (4)              else:
43702: (8)                  raise Exception( f'Unexpected color should be float or list of 1-4
floats: {color}')
43703: (0)          def pdf_set_annot_interior_color(self, color):
43704: (4)              """
43705: (4)              Python version of pdf_set_annot_color() using
43706: (4)              pdf_set_annot_color2().
43707: (4)              """
43708: (4)              return ll_pdf_set_annot_interior_color(self.m_internal, color)
43709: (0)          PdfAnnot.pdf_set_annot_interior_color = pdf_set_annot_interior_color
43710: (0)          def ll_fz_fill_text( dev, text, ctm, colorspace, color, alpha, color_params):
43711: (4)              """
43712: (4)              Low-level Python version of fz_fill_text() taking list/tuple for `color`.
43713: (4)              """
43714: (4)              color = tuple(color) + (0,) * (4-len(color))
43715: (4)              assert len(color) == 4, f'color not len 4: len={len(color)}: {color}'
43716: (4)              return ll_fz_fill_text2(dev, text, ctm, colorspace, *color, alpha,
color_params)
43717: (0)          def fz_fill_text(dev, text, ctm, colorspace, color, alpha, color_params):
43718: (4)              """
43719: (4)              Python version of fz_fill_text() taking list/tuple for `color`.
43720: (4)              """
43721: (4)              return ll_fz_fill_text(
43722: (12)                     dev.m_internal,
43723: (12)                     text.m_internal,
43724: (12)                     ctm.internal(),
43725: (12)                     colorspace.m_internal,
43726: (12)                     color,
43727: (12)                     alpha,
43728: (12)                     color_params.internal(),
43729: (12)                     )
43730: (0)          FzDevice.fz_fill_text = fz_fill_text
43731: (0)          def ll_fz_convert_color( ss, sv, ds, is_, params):
43732: (4)              """
43733: (4)              Low-level Python version of fz_convert_color().
43734: (4)              `sv` should be a float or list of 1-4 floats or a SWIG
43735: (4)              representation of a float*.
43736: (4)              Returns (dv0, dv1, dv2, dv3).
43737: (4)              """
43738: (4)              dv = fz_convert_color2_v()
43739: (4)              if isinstance( sv, float):
43740: (7)                 ll_fz_convert_color2( ss, sv, 0.0, 0.0, 0.0, ds, dv, is_, params)
43741: (4)              elif isinstance( sv, (tuple, list)):
43742: (8)                  sv2 = tuple(sv) + (0,) * (4-len(sv))
```

```
43743: (8)                  ll_fz_convert_color2( ss, *sv2, ds, dv, is_, params)
43744: (4)              else:
43745: (8)                  ll_fz_convert_color2( ss, sv, ds, dv, is_, params)
43746: (4)              return dv.v0, dv.v1, dv.v2, dv.v3
43747: (0)          def fz_convert_color( ss, sv, ds, is_, params):
43748: (4)              """
43749: (4)              Python version of fz_convert_color().
43750: (4)              `sv` should be a float or list of 1-4 floats or a SWIG
43751: (4)              representation of a float*.
43752: (4)              Returns (dv0, dv1, dv2, dv3).
43753: (4)              """
43754: (4)              return ll_fz_convert_color( ss.m_internal, sv, ds.m_internal,
is_.m_internal, params.internal())
43755: (0)          FzColorspace.fz_convert_color = fz_convert_color
43756: (0)          set_warning_callback_s = None
43757: (0)          set_error_callback_s = None
43758: (0)          class DiagnosticCallbackPython( DiagnosticCallback):
43759: (4)              """
43760: (4)              Overrides Director class DiagnosticCallback's virtual
43761: (4)              `_print()` method in Python.
43762: (4)              """
43763: (4)              def __init__( self, description, printfn):
43764: (8)                  super().__init__( description)
43765: (8)                  self.printfn = printfn
43766: (8)                  if g_mupdf_trace_director:
43767: (12)                     log( f'DiagnosticCallbackPython[{self.m_description}].__init__()
self={self!r} printfn={printfn!r}')
43768: (4)              def __del__( self):
43769: (8)                  if g_mupdf_trace_director:
43770: (12)                     log( f'DiagnosticCallbackPython[{self.m_description}].__del__()
destructor called.')
43771: (4)              def _print( self, message):
43772: (8)                  if g_mupdf_trace_director:
43773: (12)                     log( f'DiagnosticCallbackPython[{self.m_description}]._print():
Calling self.printfn={self.printfn!r} with message={message!r}')
43774: (8)                  try:
43775: (12)                     self.printfn( message)
43776: (8)                  except Exception as e:
43777: (12)                     log( f'DiagnosticCallbackPython[{self.m_description}]._print():
Warning: exception from self.printfn={self.printfn!r}: e={e!r}')
43778: (12)                     raise
43779: (0)          def set_diagnostic_callback( description, printfn):
43780: (4)              if g_mupdf_trace_director:
43781: (8)                  log( f'set_diagnostic_callback() description={description!r} printfn=
{printfn!r}')
43782: (4)              if printfn:
43783: (8)                  ret = DiagnosticCallbackPython( description, printfn)
43784: (8)                  return ret
43785: (4)              else:
43786: (8)                  if g_mupdf_trace_director:
43787: (12)                     log( f'Calling ll_fz_set_{description}_callback() with (None,
None)')
43788: (8)                  if description == 'error':
43789: (12)                     ll_fz_set_error_callback( None, None)
43790: (8)                  elif description == 'warning':
43791: (12)                     ll_fz_set_warning_callback( None, None)
43792: (8)                  else:
43793: (12)                     assert 0, f'Unrecognised description={description!r}'
43794: (8)                  return None
43795: (0)          def fz_set_error_callback( printfn):
43796: (4)              global set_error_callback_s
43797: (4)              set_error_callback_s = set_diagnostic_callback( 'error', printfn)
43798: (0)          def fz_set_warning_callback( printfn):
43799: (4)              global set_warning_callback_s
43800: (4)              set_warning_callback_s = set_diagnostic_callback( 'warning', printfn)
43801: (0)          def ll_fz_pixmap_samples_memoryview( pixmap):
43802: (4)              """
43803: (4)              Returns a writable Python `memoryview` for a `fz_pixmap`.
43804: (4)              """
```

```
43805: (4)              assert isinstance( pixmap, fz_pixmap)
43806: (4)              ret = python_memoryview_from_memory(
43807: (12)                     ll_fz_pixmap_samples( pixmap),
43808: (12)                     ll_fz_pixmap_stride( pixmap) * ll_fz_pixmap_height( pixmap),
43809: (12)                     1, # writable
43810: (12)                     )
43811: (4)              return ret
43812: (0)          def fz_pixmap_samples_memoryview( pixmap):
43813: (4)              """
43814: (4)              Returns a writable Python `memoryview` for a `FzPixmap`.
43815: (4)              """
43816: (4)              return ll_fz_pixmap_samples_memoryview( pixmap.m_internal)
43817: (0)          FzPixmap.fz_pixmap_samples_memoryview = fz_pixmap_samples_memoryview
43818: (0)          ll_fz_warn_original = ll_fz_warn
43819: (0)          def ll_fz_warn( text):
43820: (4)              assert isinstance( text, str), f'text={text!r} str={str!r}'
43821: (4)              text = text.replace( '%', '%%')
43822: (4)              return ll_fz_warn_original( text)
43823: (0)          fz_warn = ll_fz_warn
43824: (0)          ll_pdf_load_field_name = ll_pdf_load_field_name2
43825: (0)          pdf_load_field_name = pdf_load_field_name2
43826: (0)          PdfObj.pdf_load_field_name = pdf_load_field_name
43827: (0)          class StoryPositionsCallback_python( StoryPositionsCallback):
43828: (4)              def __init__( self, python_callback):
43829: (8)                  super().__init__()
43830: (8)                  self.python_callback = python_callback
43831: (4)              def call( self, position):
43832: (8)                  self.python_callback( position)
43833: (0)          ll_fz_story_positions_orig = ll_fz_story_positions
43834: (0)          def ll_fz_story_positions( story, python_callback):
43835: (4)              """
43836: (4)              Custom replacement for `ll_fz_story_positions()` that takes
43837: (4)              a Python callable `python_callback`.
43838: (4)              """
43839: (4)              python_callback_instance = StoryPositionsCallback_python( python_callback)
43840: (4)              ll_fz_story_positions_director( story, python_callback_instance)
43841: (0)          def fz_story_positions( story, python_callback):
43842: (4)              assert isinstance( story, FzStory)
43843: (4)              assert callable( python_callback)
43844: (4)              def python_callback2( position):
43845: (8)                  position2 = FzStoryElementPosition( position)
43846: (8)                  python_callback( position2)
43847: (4)              ll_fz_story_positions( story.m_internal, python_callback2)
43848: (0)          FzStory.fz_story_positions = fz_story_positions
43849: (0)          FzDocumentWriter__init__0 = FzDocumentWriter.__init__
43850: (0)          def FzDocumentWriter__init__1(self, *args):
43851: (4)              out = None
43852: (4)              for arg in args:
43853: (8)                  if isinstance( arg, FzOutput2):
43854: (12)                     assert not out, "More than one FzOutput2 passed to
FzDocumentWriter.__init__()"
43855: (12)                     out = arg
43856: (4)              if out:
43857: (8)                  self._out = out
43858: (4)              return FzDocumentWriter__init__0(self, *args)
43859: (0)          FzDocumentWriter.__init__ = FzDocumentWriter__init__1
43860: (0)          class
fz_install_load_system_font_funcs_args3(FzInstallLoadSystemFontFuncsArgs2):
43861: (4)              """
43862: (4)              Class derived from Swig Director class
43863: (4)              fz_install_load_system_font_funcs_args2, to allow
43864: (4)              implementation of fz_install_load_system_font_funcs with
43865: (4)              Python callbacks.
43866: (4)              """
43867: (4)              def __init__(self, f=None, f_cjk=None, f_fallback=None):
43868: (8)                  super().__init__()
43869: (8)                  self.f3 = f
43870: (8)                  self.f_cjk3 = f_cjk
43871: (8)                  self.f_fallback3 = f_fallback
```

```
43872: (8)                  self.use_virtual_f(True if f else False)
43873: (8)                  self.use_virtual_f_cjk(True if f_cjk else False)
43874: (8)                  self.use_virtual_f_fallback(True if f_fallback else False)
43875: (4)              def ret_font(self, font):
43876: (8)                  if font is None:
43877: (12)                     return None
43878: (8)                  elif isinstance(font, FzFont):
43879: (12)                     return ll_fz_keep_font(font.m_internal)
43880: (8)                  elif isinstance(font, fz_font):
43881: (12)                     return font
43882: (8)                  else:
43883: (12)                     assert 0, f'Expected FzFont or fz_font, but
fz_install_load_system_font_funcs() callback returned {type(font)=}'
43884: (4)              def f(self, ctx, name, bold, italic, needs_exact_metrics):
43885: (8)                  font = self.f3(name, bold, italic, needs_exact_metrics)
43886: (8)                  return self.ret_font(font)
43887: (4)              def f_cjk(self, ctx, name, ordering, serif):
43888: (8)                  font = self.f_cjk3(name, ordering, serif)
43889: (8)                  return self.ret_font(font)
43890: (4)              def f_fallback(self, ctx, script, language, serif, bold, italic):
43891: (8)                  font = self.f_fallback3(script, language, serif, bold, italic)
43892: (8)                  return self.ret_font(font)
43893: (0)          g_fz_install_load_system_font_funcs_args = None
43894: (0)          def fz_install_load_system_font_funcs(f=None, f_cjk=None, f_fallback=None):
43895: (4)              """
43896: (4)              Python override for MuPDF
43897: (4)              fz_install_load_system_font_funcs() using Swig Director
43898: (4)              support. Python callbacks are not passed a `ctx` arg, and
43899: (4)              can return None, a mupdf.fz_font or a mupdf.FzFont.
43900: (4)              """
43901: (4)              global g_fz_install_load_system_font_funcs_args
43902: (4)              g_fz_install_load_system_font_funcs_args =
fz_install_load_system_font_funcs_args3(
43903: (12)                     f,
43904: (12)                     f_cjk,
43905: (12)                     f_fallback,
43906: (12)                     )
43907: (4)
fz_install_load_system_font_funcs2(g_fz_install_load_system_font_funcs_args)
43908: (0)          FzLink.__iter__ = lambda self: IteratorWrap( self)
43909: (0)          FzStextBlock.__iter__ = lambda self: IteratorWrap( self)
43910: (0)          FzStextLine.__iter__ = lambda self: IteratorWrap( self)
43911: (0)          FzStextPage.__iter__ = lambda self: IteratorWrap( self)
43912: (0)          fz_aa_context.__str__ = lambda s: to_string_fz_aa_context(s)
43913: (0)          fz_aa_context.__repr__ = lambda s: to_string_fz_aa_context(s)
43914: (0)          fz_color_params.__str__ = lambda s: to_string_fz_color_params(s)
43915: (0)          fz_color_params.__repr__ = lambda s: to_string_fz_color_params(s)
43916: (0)          fz_cookie.__str__ = lambda s: to_string_fz_cookie(s)
43917: (0)          fz_cookie.__repr__ = lambda s: to_string_fz_cookie(s)
43918: (0)          fz_draw_options.__str__ = lambda s: to_string_fz_draw_options(s)
43919: (0)          fz_draw_options.__repr__ = lambda s: to_string_fz_draw_options(s)
43920: (0)          fz_install_load_system_font_funcs_args.__str__ = lambda s:
to_string_fz_install_load_system_font_funcs_args(s)
43921: (0)          fz_install_load_system_font_funcs_args.__repr__ = lambda s:
to_string_fz_install_load_system_font_funcs_args(s)
43922: (0)          fz_irect.__str__ = lambda s: to_string_fz_irect(s)
43923: (0)          fz_irect.__repr__ = lambda s: to_string_fz_irect(s)
43924: (0)          fz_location.__str__ = lambda s: to_string_fz_location(s)
43925: (0)          fz_location.__repr__ = lambda s: to_string_fz_location(s)
43926: (0)          fz_matrix.__str__ = lambda s: to_string_fz_matrix(s)
43927: (0)          fz_matrix.__repr__ = lambda s: to_string_fz_matrix(s)
43928: (0)          fz_md5.__str__ = lambda s: to_string_fz_md5(s)
43929: (0)          fz_md5.__repr__ = lambda s: to_string_fz_md5(s)
43930: (0)          fz_pdfocr_options.__str__ = lambda s: to_string_fz_pdfocr_options(s)
43931: (0)          fz_pdfocr_options.__repr__ = lambda s: to_string_fz_pdfocr_options(s)
43932: (0)          fz_point.__str__ = lambda s: to_string_fz_point(s)
43933: (0)          fz_point.__repr__ = lambda s: to_string_fz_point(s)
43934: (0)          fz_pwg_options.__str__ = lambda s: to_string_fz_pwg_options(s)
43935: (0)          fz_pwg_options.__repr__ = lambda s: to_string_fz_pwg_options(s)
```

```
43936: (0)          fz_quad.__str__ = lambda s: to_string_fz_quad(s)
43937: (0)          fz_quad.__repr__ = lambda s: to_string_fz_quad(s)
43938: (0)          fz_rect.__str__ = lambda s: to_string_fz_rect(s)
43939: (0)          fz_rect.__repr__ = lambda s: to_string_fz_rect(s)
43940: (0)          fz_stext_options.__str__ = lambda s: to_string_fz_stext_options(s)
43941: (0)          fz_stext_options.__repr__ = lambda s: to_string_fz_stext_options(s)
43942: (0)          fz_story_element_position.__str__ = lambda s:
to_string_fz_story_element_position(s)
43943: (0)          fz_story_element_position.__repr__ = lambda s:
to_string_fz_story_element_position(s)
43944: (0)          fz_transition.__str__ = lambda s: to_string_fz_transition(s)
43945: (0)          fz_transition.__repr__ = lambda s: to_string_fz_transition(s)
43946: (0)          pdf_clean_options.__str__ = lambda s: to_string_pdf_clean_options(s)
43947: (0)          pdf_clean_options.__repr__ = lambda s: to_string_pdf_clean_options(s)
43948: (0)          pdf_filter_factory.__str__ = lambda s: to_string_pdf_filter_factory(s)
43949: (0)          pdf_filter_factory.__repr__ = lambda s: to_string_pdf_filter_factory(s)
43950: (0)          pdf_filter_options.__str__ = lambda s: to_string_pdf_filter_options(s)
43951: (0)          pdf_filter_options.__repr__ = lambda s: to_string_pdf_filter_options(s)
43952: (0)          pdf_image_rewriter_options.__str__ = lambda s:
to_string_pdf_image_rewriter_options(s)
43953: (0)          pdf_image_rewriter_options.__repr__ = lambda s:
to_string_pdf_image_rewriter_options(s)
43954: (0)          pdf_layer_config.__str__ = lambda s: to_string_pdf_layer_config(s)
43955: (0)          pdf_layer_config.__repr__ = lambda s: to_string_pdf_layer_config(s)
43956: (0)          pdf_layer_config_ui.__str__ = lambda s: to_string_pdf_layer_config_ui(s)
43957: (0)          pdf_layer_config_ui.__repr__ = lambda s: to_string_pdf_layer_config_ui(s)
43958: (0)          pdf_redact_options.__str__ = lambda s: to_string_pdf_redact_options(s)
43959: (0)          pdf_redact_options.__repr__ = lambda s: to_string_pdf_redact_options(s)
43960: (0)          pdf_sanitize_filter_options.__str__ = lambda s:
to_string_pdf_sanitize_filter_options(s)
43961: (0)          pdf_sanitize_filter_options.__repr__ = lambda s:
to_string_pdf_sanitize_filter_options(s)
43962: (0)          pdf_write_options.__str__ = lambda s: to_string_pdf_write_options(s)
43963: (0)          pdf_write_options.__repr__ = lambda s: to_string_pdf_write_options(s)
43964: (0)          FzAaContext.__str__ = lambda self: self.to_string()
43965: (0)          FzAaContext.__repr__ = lambda self: self.to_string()
43966: (0)          FzColorParams.__str__ = lambda self: self.to_string()
43967: (0)          FzColorParams.__repr__ = lambda self: self.to_string()
43968: (0)          FzCookie.__str__ = lambda self: self.to_string()
43969: (0)          FzCookie.__repr__ = lambda self: self.to_string()
43970: (0)          FzDrawOptions.__str__ = lambda self: self.to_string()
43971: (0)          FzDrawOptions.__repr__ = lambda self: self.to_string()
43972: (0)          FzInstallLoadSystemFontFuncsArgs.__str__ = lambda self: self.to_string()
43973: (0)          FzInstallLoadSystemFontFuncsArgs.__repr__ = lambda self: self.to_string()
43974: (0)          FzIrect.__str__ = lambda self: self.to_string()
43975: (0)          FzIrect.__repr__ = lambda self: self.to_string()
43976: (0)          FzLocation.__str__ = lambda self: self.to_string()
43977: (0)          FzLocation.__repr__ = lambda self: self.to_string()
43978: (0)          FzMatrix.__str__ = lambda self: self.to_string()
43979: (0)          FzMatrix.__repr__ = lambda self: self.to_string()
43980: (0)          FzMd5.__str__ = lambda self: self.to_string()
43981: (0)          FzMd5.__repr__ = lambda self: self.to_string()
43982: (0)          FzPdfocrOptions.__str__ = lambda self: self.to_string()
43983: (0)          FzPdfocrOptions.__repr__ = lambda self: self.to_string()
43984: (0)          FzPoint.__str__ = lambda self: self.to_string()
43985: (0)          FzPoint.__repr__ = lambda self: self.to_string()
43986: (0)          FzPwgOptions.__str__ = lambda self: self.to_string()
43987: (0)          FzPwgOptions.__repr__ = lambda self: self.to_string()
43988: (0)          FzQuad.__str__ = lambda self: self.to_string()
43989: (0)          FzQuad.__repr__ = lambda self: self.to_string()
43990: (0)          FzRect.__str__ = lambda self: self.to_string()
43991: (0)          FzRect.__repr__ = lambda self: self.to_string()
43992: (0)          FzStextOptions.__str__ = lambda self: self.to_string()
43993: (0)          FzStextOptions.__repr__ = lambda self: self.to_string()
43994: (0)          FzStoryElementPosition.__str__ = lambda self: self.to_string()
43995: (0)          FzStoryElementPosition.__repr__ = lambda self: self.to_string()
43996: (0)          FzTransition.__str__ = lambda self: self.to_string()
43997: (0)          FzTransition.__repr__ = lambda self: self.to_string()
43998: (0)          PdfCleanOptions.__str__ = lambda self: self.to_string()
```

```
43999: (0)          PdfCleanOptions.__repr__ = lambda self: self.to_string()
44000: (0)          PdfFilterFactory.__str__ = lambda self: self.to_string()
44001: (0)          PdfFilterFactory.__repr__ = lambda self: self.to_string()
44002: (0)          PdfFilterOptions.__str__ = lambda self: self.to_string()
44003: (0)          PdfFilterOptions.__repr__ = lambda self: self.to_string()
44004: (0)          PdfImageRewriterOptions.__str__ = lambda self: self.to_string()
44005: (0)          PdfImageRewriterOptions.__repr__ = lambda self: self.to_string()
44006: (0)          PdfLayerConfig.__str__ = lambda self: self.to_string()
44007: (0)          PdfLayerConfig.__repr__ = lambda self: self.to_string()
44008: (0)          PdfLayerConfigUi.__str__ = lambda self: self.to_string()
44009: (0)          PdfLayerConfigUi.__repr__ = lambda self: self.to_string()
44010: (0)          PdfRedactOptions.__str__ = lambda self: self.to_string()
44011: (0)          PdfRedactOptions.__repr__ = lambda self: self.to_string()
44012: (0)          PdfSanitizeFilterOptions.__str__ = lambda self: self.to_string()
44013: (0)          PdfSanitizeFilterOptions.__repr__ = lambda self: self.to_string()
44014: (0)          PdfWriteOptions.__str__ = lambda self: self.to_string()
44015: (0)          PdfWriteOptions.__repr__ = lambda self: self.to_string()
44016: (0)          PDF_ENUM_NAME_1_2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_1_2))
44017: (0)          PDF_ENUM_NAME_1_5 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_1_5))
44018: (0)          PDF_ENUM_NAME_3D = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_3D))
44019: (0)          PDF_ENUM_NAME_A = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_A))
44020: (0)          PDF_ENUM_NAME_A85 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_A85))
44021: (0)          PDF_ENUM_NAME_AA = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AA))
44022: (0)          PDF_ENUM_NAME_AC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AC))
44023: (0)          PDF_ENUM_NAME_AESV2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AESV2))
44024: (0)          PDF_ENUM_NAME_AESV3 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AESV3))
44025: (0)          PDF_ENUM_NAME_AHx = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AHx))
44026: (0)          PDF_ENUM_NAME_AP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AP))
44027: (0)          PDF_ENUM_NAME_AS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AS))
44028: (0)          PDF_ENUM_NAME_ASCII85Decode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ASCII85Decode))
44029: (0)          PDF_ENUM_NAME_ASCIIHexDecode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ASCIIHexDecode))
44030: (0)          PDF_ENUM_NAME_AcroForm = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AcroForm))
44031: (0)          PDF_ENUM_NAME_Action = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Action))
44032: (0)          PDF_ENUM_NAME_ActualText = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ActualText))
44033: (0)          PDF_ENUM_NAME_Adobe_PPKLite = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Adobe_PPKLite))
44034: (0)          PDF_ENUM_NAME_All = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_All))
44035: (0)          PDF_ENUM_NAME_AllOff = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AllOff))
44036: (0)          PDF_ENUM_NAME_AllOn = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AllOn))
44037: (0)          PDF_ENUM_NAME_Alpha = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Alpha))
44038: (0)          PDF_ENUM_NAME_Alt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Alt))
44039: (0)          PDF_ENUM_NAME_Alternate = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Alternate))
44040: (0)          PDF_ENUM_NAME_Annot = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Annot))
44041: (0)          PDF_ENUM_NAME_Annots = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Annots))
44042: (0)          PDF_ENUM_NAME_AnyOff = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AnyOff))
44043: (0)          PDF_ENUM_NAME_App = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_App))
44044: (0)          PDF_ENUM_NAME_Approved = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Approved))
44045: (0)          PDF_ENUM_NAME_Art = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Art))
44046: (0)          PDF_ENUM_NAME_ArtBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ArtBox))
44047: (0)          PDF_ENUM_NAME_Artifact = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Artifact))
44048: (0)          PDF_ENUM_NAME_AsIs = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AsIs))
44049: (0)          PDF_ENUM_NAME_Ascent = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ascent))
44050: (0)          PDF_ENUM_NAME_Aside = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Aside))
44051: (0)          PDF_ENUM_NAME_AuthEvent = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_AuthEvent))
44052: (0)          PDF_ENUM_NAME_Author = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Author))
44053: (0)          PDF_ENUM_NAME_B = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_B))
44054: (0)          PDF_ENUM_NAME_BBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BBox))
44055: (0)          PDF_ENUM_NAME_BC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BC))
44056: (0)          PDF_ENUM_NAME_BE = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BE))
44057: (0)          PDF_ENUM_NAME_BG = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BG))
44058: (0)          PDF_ENUM_NAME_BM = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BM))
44059: (0)          PDF_ENUM_NAME_BPC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BPC))
44060: (0)          PDF_ENUM_NAME_BS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BS))
44061: (0)          PDF_ENUM_NAME_Background = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Background))
44062: (0)          PDF_ENUM_NAME_BaseEncoding = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_BaseEncoding))
44063: (0)          PDF_ENUM_NAME_BaseFont = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BaseFont))
```

```
44064: (0)          PDF_ENUM_NAME_BaseState = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BaseState))
44065: (0)          PDF_ENUM_NAME_BibEntry = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BibEntry))
44066: (0)          PDF_ENUM_NAME_BitsPerComponent = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_BitsPerComponent))
44067: (0)          PDF_ENUM_NAME_BitsPerCoordinate = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_BitsPerCoordinate))
44068: (0)          PDF_ENUM_NAME_BitsPerFlag = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_BitsPerFlag))
44069: (0)          PDF_ENUM_NAME_BitsPerSample = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_BitsPerSample))
44070: (0)          PDF_ENUM_NAME_BlackIs1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BlackIs1))
44071: (0)          PDF_ENUM_NAME_BlackPoint = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BlackPoint))
44072: (0)          PDF_ENUM_NAME_BleedBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BleedBox))
44073: (0)          PDF_ENUM_NAME_Blinds = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Blinds))
44074: (0)          PDF_ENUM_NAME_BlockQuote = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_BlockQuote))
44075: (0)          PDF_ENUM_NAME_Border = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Border))
44076: (0)          PDF_ENUM_NAME_Bounds = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Bounds))
44077: (0)          PDF_ENUM_NAME_Box = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Box))
44078: (0)          PDF_ENUM_NAME_Bt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Bt))
44079: (0)          PDF_ENUM_NAME_Btn = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Btn))
44080: (0)          PDF_ENUM_NAME_Butt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Butt))
44081: (0)          PDF_ENUM_NAME_ByteRange = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ByteRange))
44082: (0)          PDF_ENUM_NAME_C = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_C))
44083: (0)          PDF_ENUM_NAME_C0 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_C0))
44084: (0)          PDF_ENUM_NAME_C1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_C1))
44085: (0)          PDF_ENUM_NAME_CA = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CA))
44086: (0)          PDF_ENUM_NAME_CCF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CCF))
44087: (0)          PDF_ENUM_NAME_CCITTFaxDecode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CCITTFaxDecode))
44088: (0)          PDF_ENUM_NAME_CF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CF))
44089: (0)          PDF_ENUM_NAME_CFM = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CFM))
44090: (0)          PDF_ENUM_NAME_CI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CI))
44091: (0)          PDF_ENUM_NAME_CIDFontType0 = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CIDFontType0))
44092: (0)          PDF_ENUM_NAME_CIDFontType0C = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CIDFontType0C))
44093: (0)          PDF_ENUM_NAME_CIDFontType2 = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CIDFontType2))
44094: (0)          PDF_ENUM_NAME_CIDSystemInfo = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CIDSystemInfo))
44095: (0)          PDF_ENUM_NAME_CIDToGIDMap = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CIDToGIDMap))
44096: (0)          PDF_ENUM_NAME_CMYK = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CMYK))
44097: (0)          PDF_ENUM_NAME_CS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CS))
44098: (0)          PDF_ENUM_NAME_CalCMYK = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CalCMYK))
44099: (0)          PDF_ENUM_NAME_CalGray = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CalGray))
44100: (0)          PDF_ENUM_NAME_CalRGB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CalRGB))
44101: (0)          PDF_ENUM_NAME_CapHeight = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CapHeight))
44102: (0)          PDF_ENUM_NAME_Caption = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Caption))
44103: (0)          PDF_ENUM_NAME_Caret = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Caret))
44104: (0)          PDF_ENUM_NAME_Catalog = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Catalog))
44105: (0)          PDF_ENUM_NAME_Cert = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Cert))
44106: (0)          PDF_ENUM_NAME_Ch = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ch))
44107: (0)          PDF_ENUM_NAME_Changes = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Changes))
44108: (0)          PDF_ENUM_NAME_CharProcs = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CharProcs))
44109: (0)          PDF_ENUM_NAME_CheckSum = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CheckSum))
44110: (0)          PDF_ENUM_NAME_Circle = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Circle))
44111: (0)          PDF_ENUM_NAME_ClosedArrow = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ClosedArrow))
44112: (0)          PDF_ENUM_NAME_Code = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Code))
44113: (0)          PDF_ENUM_NAME_Collection = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Collection))
44114: (0)          PDF_ENUM_NAME_ColorSpace = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ColorSpace))
44115: (0)          PDF_ENUM_NAME_ColorTransform = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ColorTransform))
44116: (0)          PDF_ENUM_NAME_Colorants = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Colorants))
44117: (0)          PDF_ENUM_NAME_Colors = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Colors))
44118: (0)          PDF_ENUM_NAME_Columns = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Columns))
44119: (0)          PDF_ENUM_NAME_Confidential = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Confidential))
```

```
44120: (0)          PDF_ENUM_NAME_Configs = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Configs))
44121: (0)          PDF_ENUM_NAME_ContactInfo = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ContactInfo))
44122: (0)          PDF_ENUM_NAME_Contents = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Contents))
44123: (0)          PDF_ENUM_NAME_Coords = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Coords))
44124: (0)          PDF_ENUM_NAME_Count = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Count))
44125: (0)          PDF_ENUM_NAME_Cover = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Cover))
44126: (0)          PDF_ENUM_NAME_CreationDate = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_CreationDate))
44127: (0)          PDF_ENUM_NAME_Creator = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Creator))
44128: (0)          PDF_ENUM_NAME_CropBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_CropBox))
44129: (0)          PDF_ENUM_NAME_Crypt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Crypt))
44130: (0)          PDF_ENUM_NAME_D = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_D))
44131: (0)          PDF_ENUM_NAME_DA = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DA))
44132: (0)          PDF_ENUM_NAME_DC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DC))
44133: (0)          PDF_ENUM_NAME_DCT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DCT))
44134: (0)          PDF_ENUM_NAME_DCTDecode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DCTDecode))
44135: (0)          PDF_ENUM_NAME_DL = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DL))
44136: (0)          PDF_ENUM_NAME_DOS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DOS))
44137: (0)          PDF_ENUM_NAME_DP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DP))
44138: (0)          PDF_ENUM_NAME_DR = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DR))
44139: (0)          PDF_ENUM_NAME_DS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DS))
44140: (0)          PDF_ENUM_NAME_DV = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DV))
44141: (0)          PDF_ENUM_NAME_DW = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DW))
44142: (0)          PDF_ENUM_NAME_DW2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DW2))
44143: (0)          PDF_ENUM_NAME_DamagedRowsBeforeError = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DamagedRowsBeforeError))
44144: (0)          PDF_ENUM_NAME_Data = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Data))
44145: (0)          PDF_ENUM_NAME_Date = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Date))
44146: (0)          PDF_ENUM_NAME_Decode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Decode))
44147: (0)          PDF_ENUM_NAME_DecodeParms = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DecodeParms))
44148: (0)          PDF_ENUM_NAME_Default = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Default))
44149: (0)          PDF_ENUM_NAME_DefaultCMYK = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DefaultCMYK))
44150: (0)          PDF_ENUM_NAME_DefaultGray = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DefaultGray))
44151: (0)          PDF_ENUM_NAME_DefaultRGB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DefaultRGB))
44152: (0)          PDF_ENUM_NAME_Departmental = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Departmental))
44153: (0)          PDF_ENUM_NAME_Desc = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Desc))
44154: (0)          PDF_ENUM_NAME_DescendantFonts = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DescendantFonts))
44155: (0)          PDF_ENUM_NAME_Descent = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Descent))
44156: (0)          PDF_ENUM_NAME_Design = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Design))
44157: (0)          PDF_ENUM_NAME_Dest = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Dest))
44158: (0)          PDF_ENUM_NAME_DestOutputProfile = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DestOutputProfile))
44159: (0)          PDF_ENUM_NAME_Dests = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Dests))
44160: (0)          PDF_ENUM_NAME_DeviceCMYK = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DeviceCMYK))
44161: (0)          PDF_ENUM_NAME_DeviceGray = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DeviceGray))
44162: (0)          PDF_ENUM_NAME_DeviceN = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DeviceN))
44163: (0)          PDF_ENUM_NAME_DeviceRGB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DeviceRGB))
44164: (0)          PDF_ENUM_NAME_Di = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Di))
44165: (0)          PDF_ENUM_NAME_Diamond = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Diamond))
44166: (0)          PDF_ENUM_NAME_Differences = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Differences))
44167: (0)          PDF_ENUM_NAME_DigestLocation = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DigestLocation))
44168: (0)          PDF_ENUM_NAME_DigestMethod = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DigestMethod))
44169: (0)          PDF_ENUM_NAME_DigestValue = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_DigestValue))
44170: (0)          PDF_ENUM_NAME_Dissolve = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Dissolve))
44171: (0)          PDF_ENUM_NAME_Div = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Div))
44172: (0)          PDF_ENUM_NAME_Dm = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Dm))
44173: (0)          PDF_ENUM_NAME_DocMDP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_DocMDP))
44174: (0)          PDF_ENUM_NAME_Document = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Document))
44175: (0)          PDF_ENUM_NAME_DocumentFragment = PdfObj( obj_enum_to_obj(
```

```
PDF_ENUM_NAME_DocumentFragment))
44176: (0)          PDF_ENUM_NAME_Domain = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Domain))
44177: (0)          PDF_ENUM_NAME_Draft = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Draft))
44178: (0)          PDF_ENUM_NAME_Dur = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Dur))
44179: (0)          PDF_ENUM_NAME_E = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_E))
44180: (0)          PDF_ENUM_NAME_EF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_EF))
44181: (0)          PDF_ENUM_NAME_EarlyChange = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_EarlyChange))
44182: (0)          PDF_ENUM_NAME_Em = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Em))
44183: (0)          PDF_ENUM_NAME_EmbeddedFile = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_EmbeddedFile))
44184: (0)          PDF_ENUM_NAME_EmbeddedFiles = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_EmbeddedFiles))
44185: (0)          PDF_ENUM_NAME_Encode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Encode))
44186: (0)          PDF_ENUM_NAME_EncodedByteAlign = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_EncodedByteAlign))
44187: (0)          PDF_ENUM_NAME_Encoding = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Encoding))
44188: (0)          PDF_ENUM_NAME_Encrypt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Encrypt))
44189: (0)          PDF_ENUM_NAME_EncryptMetadata = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_EncryptMetadata))
44190: (0)          PDF_ENUM_NAME_EndOfBlock = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_EndOfBlock))
44191: (0)          PDF_ENUM_NAME_EndOfLine = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_EndOfLine))
44192: (0)          PDF_ENUM_NAME_Exclude = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Exclude))
44193: (0)          PDF_ENUM_NAME_Experimental = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Experimental))
44194: (0)          PDF_ENUM_NAME_Expired = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Expired))
44195: (0)          PDF_ENUM_NAME_ExtGState = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ExtGState))
44196: (0)          PDF_ENUM_NAME_Extend = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Extend))
44197: (0)          PDF_ENUM_NAME_F = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_F))
44198: (0)          PDF_ENUM_NAME_FENote = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FENote))
44199: (0)          PDF_ENUM_NAME_FL = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FL))
44200: (0)          PDF_ENUM_NAME_FRM = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FRM))
44201: (0)          PDF_ENUM_NAME_FS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FS))
44202: (0)          PDF_ENUM_NAME_FT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FT))
44203: (0)          PDF_ENUM_NAME_Fade = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Fade))
44204: (0)          PDF_ENUM_NAME_Ff = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ff))
44205: (0)          PDF_ENUM_NAME_FieldMDP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FieldMDP))
44206: (0)          PDF_ENUM_NAME_Fields = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Fields))
44207: (0)          PDF_ENUM_NAME_Figure = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Figure))
44208: (0)          PDF_ENUM_NAME_FileAttachment = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FileAttachment))
44209: (0)          PDF_ENUM_NAME_FileSize = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FileSize))
44210: (0)          PDF_ENUM_NAME_Filespec = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Filespec))
44211: (0)          PDF_ENUM_NAME_Filter = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Filter))
44212: (0)          PDF_ENUM_NAME_Final = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Final))
44213: (0)          PDF_ENUM_NAME_Fingerprint = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Fingerprint))
44214: (0)          PDF_ENUM_NAME_First = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_First))
44215: (0)          PDF_ENUM_NAME_FirstChar = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FirstChar))
44216: (0)          PDF_ENUM_NAME_FirstPage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FirstPage))
44217: (0)          PDF_ENUM_NAME_Fit = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Fit))
44218: (0)          PDF_ENUM_NAME_FitB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitB))
44219: (0)          PDF_ENUM_NAME_FitBH = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitBH))
44220: (0)          PDF_ENUM_NAME_FitBV = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitBV))
44221: (0)          PDF_ENUM_NAME_FitH = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitH))
44222: (0)          PDF_ENUM_NAME_FitR = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitR))
44223: (0)          PDF_ENUM_NAME_FitV = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FitV))
44224: (0)          PDF_ENUM_NAME_Fl = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Fl))
44225: (0)          PDF_ENUM_NAME_Flags = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Flags))
44226: (0)          PDF_ENUM_NAME_FlateDecode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FlateDecode))
44227: (0)          PDF_ENUM_NAME_Fly = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Fly))
44228: (0)          PDF_ENUM_NAME_Font = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Font))
44229: (0)          PDF_ENUM_NAME_FontBBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontBBox))
44230: (0)          PDF_ENUM_NAME_FontDescriptor = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FontDescriptor))
44231: (0)          PDF_ENUM_NAME_FontFile = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontFile))
44232: (0)          PDF_ENUM_NAME_FontFile2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontFile2))
44233: (0)          PDF_ENUM_NAME_FontFile3 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontFile3))
```

```
44234: (0)          PDF_ENUM_NAME_FontMatrix = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontMatrix))
44235: (0)          PDF_ENUM_NAME_FontName = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FontName))
44236: (0)          PDF_ENUM_NAME_ForComment = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ForComment))
44237: (0)          PDF_ENUM_NAME_ForPublicRelease = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ForPublicRelease))
44238: (0)          PDF_ENUM_NAME_Form = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Form))
44239: (0)          PDF_ENUM_NAME_FormEx = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FormEx))
44240: (0)          PDF_ENUM_NAME_FormType = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FormType))
44241: (0)          PDF_ENUM_NAME_Formula = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Formula))
44242: (0)          PDF_ENUM_NAME_FreeText = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_FreeText))
44243: (0)          PDF_ENUM_NAME_FreeTextCallout = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FreeTextCallout))
44244: (0)          PDF_ENUM_NAME_FreeTextTypeWriter = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FreeTextTypeWriter))
44245: (0)          PDF_ENUM_NAME_Function = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Function))
44246: (0)          PDF_ENUM_NAME_FunctionType = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_FunctionType))
44247: (0)          PDF_ENUM_NAME_Functions = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Functions))
44248: (0)          PDF_ENUM_NAME_G = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_G))
44249: (0)          PDF_ENUM_NAME_GTS_PDFX = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_GTS_PDFX))
44250: (0)          PDF_ENUM_NAME_Gamma = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Gamma))
44251: (0)          PDF_ENUM_NAME_Glitter = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Glitter))
44252: (0)          PDF_ENUM_NAME_GoTo = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_GoTo))
44253: (0)          PDF_ENUM_NAME_GoToR = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_GoToR))
44254: (0)          PDF_ENUM_NAME_Group = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Group))
44255: (0)          PDF_ENUM_NAME_H = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H))
44256: (0)          PDF_ENUM_NAME_H1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H1))
44257: (0)          PDF_ENUM_NAME_H2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H2))
44258: (0)          PDF_ENUM_NAME_H3 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H3))
44259: (0)          PDF_ENUM_NAME_H4 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H4))
44260: (0)          PDF_ENUM_NAME_H5 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H5))
44261: (0)          PDF_ENUM_NAME_H6 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_H6))
44262: (0)          PDF_ENUM_NAME_Height = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Height))
44263: (0)          PDF_ENUM_NAME_Helv = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Helv))
44264: (0)          PDF_ENUM_NAME_Highlight = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Highlight))
44265: (0)          PDF_ENUM_NAME_HistoryPos = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_HistoryPos))
44266: (0)          PDF_ENUM_NAME_I = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_I))
44267: (0)          PDF_ENUM_NAME_IC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_IC))
44268: (0)          PDF_ENUM_NAME_ICCBased = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ICCBased))
44269: (0)          PDF_ENUM_NAME_ID = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ID))
44270: (0)          PDF_ENUM_NAME_IM = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_IM))
44271: (0)          PDF_ENUM_NAME_IRT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_IRT))
44272: (0)          PDF_ENUM_NAME_IT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_IT))
44273: (0)          PDF_ENUM_NAME_Identity = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Identity))
44274: (0)          PDF_ENUM_NAME_Identity_H = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Identity_H))
44275: (0)          PDF_ENUM_NAME_Identity_V = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Identity_V))
44276: (0)          PDF_ENUM_NAME_Image = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Image))
44277: (0)          PDF_ENUM_NAME_ImageB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ImageB))
44278: (0)          PDF_ENUM_NAME_ImageC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ImageC))
44279: (0)          PDF_ENUM_NAME_ImageI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ImageI))
44280: (0)          PDF_ENUM_NAME_ImageMask = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ImageMask))
44281: (0)          PDF_ENUM_NAME_Include = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Include))
44282: (0)          PDF_ENUM_NAME_Index = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Index))
44283: (0)          PDF_ENUM_NAME_Indexed = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Indexed))
44284: (0)          PDF_ENUM_NAME_Info = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Info))
44285: (0)          PDF_ENUM_NAME_Ink = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ink))
44286: (0)          PDF_ENUM_NAME_InkList = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_InkList))
44287: (0)          PDF_ENUM_NAME_Intent = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Intent))
44288: (0)          PDF_ENUM_NAME_Interpolate = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Interpolate))
44289: (0)          PDF_ENUM_NAME_IsMap = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_IsMap))
44290: (0)          PDF_ENUM_NAME_ItalicAngle = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ItalicAngle))
44291: (0)          PDF_ENUM_NAME_JBIG2Decode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_JBIG2Decode))
44292: (0)          PDF_ENUM_NAME_JBIG2Globals = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_JBIG2Globals))
44293: (0)          PDF_ENUM_NAME_JPXDecode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_JPXDecode))
44294: (0)          PDF_ENUM_NAME_JS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_JS))
```

```
44295: (0)          PDF_ENUM_NAME_JavaScript = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_JavaScript))
44296: (0)          PDF_ENUM_NAME_K = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_K))
44297: (0)          PDF_ENUM_NAME_Keywords = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Keywords))
44298: (0)          PDF_ENUM_NAME_Kids = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Kids))
44299: (0)          PDF_ENUM_NAME_L = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_L))
44300: (0)          PDF_ENUM_NAME_LBody = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LBody))
44301: (0)          PDF_ENUM_NAME_LC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LC))
44302: (0)          PDF_ENUM_NAME_LE = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LE))
44303: (0)          PDF_ENUM_NAME_LI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LI))
44304: (0)          PDF_ENUM_NAME_LJ = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LJ))
44305: (0)          PDF_ENUM_NAME_LW = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LW))
44306: (0)          PDF_ENUM_NAME_LZ = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LZ))
44307: (0)          PDF_ENUM_NAME_LZW = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LZW))
44308: (0)          PDF_ENUM_NAME_LZWDecode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LZWDecode))
44309: (0)          PDF_ENUM_NAME_Lab = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Lab))
44310: (0)          PDF_ENUM_NAME_Label = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Label))
44311: (0)          PDF_ENUM_NAME_Lang = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Lang))
44312: (0)          PDF_ENUM_NAME_Last = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Last))
44313: (0)          PDF_ENUM_NAME_LastChar = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LastChar))
44314: (0)          PDF_ENUM_NAME_LastPage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LastPage))
44315: (0)          PDF_ENUM_NAME_Launch = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Launch))
44316: (0)          PDF_ENUM_NAME_Layer = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Layer))
44317: (0)          PDF_ENUM_NAME_Lbl = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Lbl))
44318: (0)          PDF_ENUM_NAME_Length = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Length))
44319: (0)          PDF_ENUM_NAME_Length1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Length1))
44320: (0)          PDF_ENUM_NAME_Length2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Length2))
44321: (0)          PDF_ENUM_NAME_Length3 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Length3))
44322: (0)          PDF_ENUM_NAME_Limits = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Limits))
44323: (0)          PDF_ENUM_NAME_Line = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Line))
44324: (0)          PDF_ENUM_NAME_LineArrow = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_LineArrow))
44325: (0)          PDF_ENUM_NAME_LineDimension = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_LineDimension))
44326: (0)          PDF_ENUM_NAME_Linearized = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Linearized))
44327: (0)          PDF_ENUM_NAME_Link = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Link))
44328: (0)          PDF_ENUM_NAME_List = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_List))
44329: (0)          PDF_ENUM_NAME_Location = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Location))
44330: (0)          PDF_ENUM_NAME_Lock = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Lock))
44331: (0)          PDF_ENUM_NAME_Locked = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Locked))
44332: (0)          PDF_ENUM_NAME_Luminosity = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Luminosity))
44333: (0)          PDF_ENUM_NAME_M = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_M))
44334: (0)          PDF_ENUM_NAME_MCID = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_MCID))
44335: (0)          PDF_ENUM_NAME_MK = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_MK))
44336: (0)          PDF_ENUM_NAME_ML = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ML))
44337: (0)          PDF_ENUM_NAME_MMType1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_MMType1))
44338: (0)          PDF_ENUM_NAME_Mac = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Mac))
44339: (0)          PDF_ENUM_NAME_Mask = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Mask))
44340: (0)          PDF_ENUM_NAME_Matrix = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Matrix))
44341: (0)          PDF_ENUM_NAME_Matte = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Matte))
44342: (0)          PDF_ENUM_NAME_MaxLen = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_MaxLen))
44343: (0)          PDF_ENUM_NAME_MediaBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_MediaBox))
44344: (0)          PDF_ENUM_NAME_Metadata = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Metadata))
44345: (0)          PDF_ENUM_NAME_MissingWidth = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_MissingWidth))
44346: (0)          PDF_ENUM_NAME_ModDate = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ModDate))
44347: (0)          PDF_ENUM_NAME_Movie = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Movie))
44348: (0)          PDF_ENUM_NAME_Msg = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Msg))
44349: (0)          PDF_ENUM_NAME_Multiply = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Multiply))
44350: (0)          PDF_ENUM_NAME_N = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_N))
44351: (0)          PDF_ENUM_NAME_Name = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Name))
44352: (0)          PDF_ENUM_NAME_Named = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Named))
44353: (0)          PDF_ENUM_NAME_Names = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Names))
44354: (0)          PDF_ENUM_NAME_NewWindow = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_NewWindow))
44355: (0)          PDF_ENUM_NAME_Next = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Next))
44356: (0)          PDF_ENUM_NAME_NextPage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_NextPage))
44357: (0)          PDF_ENUM_NAME_NonEFontNoWarn = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_NonEFontNoWarn))
44358: (0)          PDF_ENUM_NAME_NonStruct = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_NonStruct))
44359: (0)          PDF_ENUM_NAME_None = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_None))
44360: (0)          PDF_ENUM_NAME_Normal = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Normal))
```

```
44361: (0)          PDF_ENUM_NAME_NotApproved = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_NotApproved))
44362: (0)          PDF_ENUM_NAME_NotForPublicRelease = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_NotForPublicRelease))
44363: (0)          PDF_ENUM_NAME_Note = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Note))
44364: (0)          PDF_ENUM_NAME_NumSections = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_NumSections))
44365: (0)          PDF_ENUM_NAME_Nums = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Nums))
44366: (0)          PDF_ENUM_NAME_O = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_O))
44367: (0)          PDF_ENUM_NAME_OC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OC))
44368: (0)          PDF_ENUM_NAME_OCG = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OCG))
44369: (0)          PDF_ENUM_NAME_OCGs = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OCGs))
44370: (0)          PDF_ENUM_NAME_OCMD = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OCMD))
44371: (0)          PDF_ENUM_NAME_OCProperties = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_OCProperties))
44372: (0)          PDF_ENUM_NAME_OE = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OE))
44373: (0)          PDF_ENUM_NAME_OFF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OFF))
44374: (0)          PDF_ENUM_NAME_ON = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ON))
44375: (0)          PDF_ENUM_NAME_OP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OP))
44376: (0)          PDF_ENUM_NAME_OPM = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OPM))
44377: (0)          PDF_ENUM_NAME_OS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OS))
44378: (0)          PDF_ENUM_NAME_ObjStm = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ObjStm))
44379: (0)          PDF_ENUM_NAME_Of = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Of))
44380: (0)          PDF_ENUM_NAME_Off = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Off))
44381: (0)          PDF_ENUM_NAME_Open = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Open))
44382: (0)          PDF_ENUM_NAME_OpenArrow = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OpenArrow))
44383: (0)          PDF_ENUM_NAME_OpenType = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_OpenType))
44384: (0)          PDF_ENUM_NAME_Opt = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Opt))
44385: (0)          PDF_ENUM_NAME_Order = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Order))
44386: (0)          PDF_ENUM_NAME_Ordering = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ordering))
44387: (0)          PDF_ENUM_NAME_Outlines = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Outlines))
44388: (0)          PDF_ENUM_NAME_OutputCondition = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_OutputCondition))
44389: (0)          PDF_ENUM_NAME_OutputConditionIdentifier = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_OutputConditionIdentifier))
44390: (0)          PDF_ENUM_NAME_OutputIntent = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_OutputIntent))
44391: (0)          PDF_ENUM_NAME_OutputIntents = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_OutputIntents))
44392: (0)          PDF_ENUM_NAME_P = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_P))
44393: (0)          PDF_ENUM_NAME_PDF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PDF))
44394: (0)          PDF_ENUM_NAME_PS = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PS))
44395: (0)          PDF_ENUM_NAME_Page = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Page))
44396: (0)          PDF_ENUM_NAME_PageLabels = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PageLabels))
44397: (0)          PDF_ENUM_NAME_PageMode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PageMode))
44398: (0)          PDF_ENUM_NAME_Pages = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Pages))
44399: (0)          PDF_ENUM_NAME_PaintType = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PaintType))
44400: (0)          PDF_ENUM_NAME_Params = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Params))
44401: (0)          PDF_ENUM_NAME_Parent = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Parent))
44402: (0)          PDF_ENUM_NAME_ParentTree = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ParentTree))
44403: (0)          PDF_ENUM_NAME_Part = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Part))
44404: (0)          PDF_ENUM_NAME_Pattern = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Pattern))
44405: (0)          PDF_ENUM_NAME_PatternType = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_PatternType))
44406: (0)          PDF_ENUM_NAME_Perms = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Perms))
44407: (0)          PDF_ENUM_NAME_PolyLine = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PolyLine))
44408: (0)          PDF_ENUM_NAME_PolyLineDimension = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_PolyLineDimension))
44409: (0)          PDF_ENUM_NAME_Polygon = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Polygon))
44410: (0)          PDF_ENUM_NAME_PolygonCloud = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_PolygonCloud))
44411: (0)          PDF_ENUM_NAME_PolygonDimension = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_PolygonDimension))
44412: (0)          PDF_ENUM_NAME_Popup = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Popup))
44413: (0)          PDF_ENUM_NAME_PreRelease = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PreRelease))
44414: (0)          PDF_ENUM_NAME_Predictor = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Predictor))
44415: (0)          PDF_ENUM_NAME_Prev = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Prev))
44416: (0)          PDF_ENUM_NAME_PrevPage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PrevPage))
44417: (0)          PDF_ENUM_NAME_Preview = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Preview))
```

```
44418: (0)          PDF_ENUM_NAME_Print = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Print))
44419: (0)          PDF_ENUM_NAME_PrinterMark = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_PrinterMark))
44420: (0)          PDF_ENUM_NAME_Private = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Private))
44421: (0)          PDF_ENUM_NAME_ProcSet = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ProcSet))
44422: (0)          PDF_ENUM_NAME_Producer = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Producer))
44423: (0)          PDF_ENUM_NAME_Prop_AuthTime = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Prop_AuthTime))
44424: (0)          PDF_ENUM_NAME_Prop_AuthType = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Prop_AuthType))
44425: (0)          PDF_ENUM_NAME_Prop_Build = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Prop_Build))
44426: (0)          PDF_ENUM_NAME_Properties = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Properties))
44427: (0)          PDF_ENUM_NAME_PubSec = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_PubSec))
44428: (0)          PDF_ENUM_NAME_Push = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Push))
44429: (0)          PDF_ENUM_NAME_Q = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Q))
44430: (0)          PDF_ENUM_NAME_QuadPoints = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_QuadPoints))
44431: (0)          PDF_ENUM_NAME_Quote = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Quote))
44432: (0)          PDF_ENUM_NAME_R = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_R))
44433: (0)          PDF_ENUM_NAME_RB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RB))
44434: (0)          PDF_ENUM_NAME_RBGroups = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RBGroups))
44435: (0)          PDF_ENUM_NAME_RC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RC))
44436: (0)          PDF_ENUM_NAME_RClosedArrow = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_RClosedArrow))
44437: (0)          PDF_ENUM_NAME_RD = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RD))
44438: (0)          PDF_ENUM_NAME_REx = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_REx))
44439: (0)          PDF_ENUM_NAME_RGB = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RGB))
44440: (0)          PDF_ENUM_NAME_RI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RI))
44441: (0)          PDF_ENUM_NAME_RL = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RL))
44442: (0)          PDF_ENUM_NAME_ROpenArrow = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ROpenArrow))
44443: (0)          PDF_ENUM_NAME_RP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RP))
44444: (0)          PDF_ENUM_NAME_RT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RT))
44445: (0)          PDF_ENUM_NAME_Range = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Range))
44446: (0)          PDF_ENUM_NAME_Reason = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Reason))
44447: (0)          PDF_ENUM_NAME_Rect = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Rect))
44448: (0)          PDF_ENUM_NAME_Redact = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Redact))
44449: (0)          PDF_ENUM_NAME_Ref = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ref))
44450: (0)          PDF_ENUM_NAME_Reference = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Reference))
44451: (0)          PDF_ENUM_NAME_Registry = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Registry))
44452: (0)          PDF_ENUM_NAME_ResetForm = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ResetForm))
44453: (0)          PDF_ENUM_NAME_Resources = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Resources))
44454: (0)          PDF_ENUM_NAME_RoleMap = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_RoleMap))
44455: (0)          PDF_ENUM_NAME_Root = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Root))
44456: (0)          PDF_ENUM_NAME_Rotate = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Rotate))
44457: (0)          PDF_ENUM_NAME_Rows = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Rows))
44458: (0)          PDF_ENUM_NAME_Ruby = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Ruby))
44459: (0)          PDF_ENUM_NAME_RunLengthDecode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_RunLengthDecode))
44460: (0)          PDF_ENUM_NAME_S = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_S))
44461: (0)          PDF_ENUM_NAME_SMask = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_SMask))
44462: (0)          PDF_ENUM_NAME_SMaskInData = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_SMaskInData))
44463: (0)          PDF_ENUM_NAME_Schema = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Schema))
44464: (0)          PDF_ENUM_NAME_Screen = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Screen))
44465: (0)          PDF_ENUM_NAME_Sect = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Sect))
44466: (0)          PDF_ENUM_NAME_Separation = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Separation))
44467: (0)          PDF_ENUM_NAME_Shading = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Shading))
44468: (0)          PDF_ENUM_NAME_ShadingType = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_ShadingType))
44469: (0)          PDF_ENUM_NAME_Si = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Si))
44470: (0)          PDF_ENUM_NAME_Sig = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Sig))
44471: (0)          PDF_ENUM_NAME_SigFlags = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_SigFlags))
44472: (0)          PDF_ENUM_NAME_SigQ = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_SigQ))
44473: (0)          PDF_ENUM_NAME_SigRef = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_SigRef))
44474: (0)          PDF_ENUM_NAME_Size = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Size))
44475: (0)          PDF_ENUM_NAME_Slash = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Slash))
44476: (0)          PDF_ENUM_NAME_Sold = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Sold))
44477: (0)          PDF_ENUM_NAME_Sound = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Sound))
44478: (0)          PDF_ENUM_NAME_Span = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Span))
44479: (0)          PDF_ENUM_NAME_Split = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Split))
```

```
44480: (0)          PDF_ENUM_NAME_Square = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Square))
44481: (0)          PDF_ENUM_NAME_Squiggly = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Squiggly))
44482: (0)          PDF_ENUM_NAME_St = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_St))
44483: (0)          PDF_ENUM_NAME_Stamp = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Stamp))
44484: (0)          PDF_ENUM_NAME_StampImage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StampImage))
44485: (0)          PDF_ENUM_NAME_StampSnapshot = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_StampSnapshot))
44486: (0)          PDF_ENUM_NAME_Standard = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Standard))
44487: (0)          PDF_ENUM_NAME_StdCF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StdCF))
44488: (0)          PDF_ENUM_NAME_StemV = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StemV))
44489: (0)          PDF_ENUM_NAME_StmF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StmF))
44490: (0)          PDF_ENUM_NAME_StrF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StrF))
44491: (0)          PDF_ENUM_NAME_StrikeOut = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_StrikeOut))
44492: (0)          PDF_ENUM_NAME_Strong = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Strong))
44493: (0)          PDF_ENUM_NAME_StructParent = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_StructParent))
44494: (0)          PDF_ENUM_NAME_StructParents = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_StructParents))
44495: (0)          PDF_ENUM_NAME_StructTreeRoot = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_StructTreeRoot))
44496: (0)          PDF_ENUM_NAME_Sub = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Sub))
44497: (0)          PDF_ENUM_NAME_SubFilter = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_SubFilter))
44498: (0)          PDF_ENUM_NAME_Subject = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Subject))
44499: (0)          PDF_ENUM_NAME_Subtype = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Subtype))
44500: (0)          PDF_ENUM_NAME_Subtype2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Subtype2))
44501: (0)          PDF_ENUM_NAME_Supplement = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Supplement))
44502: (0)          PDF_ENUM_NAME_Symb = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Symb))
44503: (0)          PDF_ENUM_NAME_T = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_T))
44504: (0)          PDF_ENUM_NAME_TBody = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TBody))
44505: (0)          PDF_ENUM_NAME_TD = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TD))
44506: (0)          PDF_ENUM_NAME_TFoot = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TFoot))
44507: (0)          PDF_ENUM_NAME_TH = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TH))
44508: (0)          PDF_ENUM_NAME_THead = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_THead))
44509: (0)          PDF_ENUM_NAME_TI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TI))
44510: (0)          PDF_ENUM_NAME_TOC = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TOC))
44511: (0)          PDF_ENUM_NAME_TOCI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TOCI))
44512: (0)          PDF_ENUM_NAME_TR = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TR))
44513: (0)          PDF_ENUM_NAME_TR2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TR2))
44514: (0)          PDF_ENUM_NAME_TU = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TU))
44515: (0)          PDF_ENUM_NAME_Table = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Table))
44516: (0)          PDF_ENUM_NAME_Text = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Text))
44517: (0)          PDF_ENUM_NAME_TilingType = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TilingType))
44518: (0)          PDF_ENUM_NAME_Times = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Times))
44519: (0)          PDF_ENUM_NAME_Title = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Title))
44520: (0)          PDF_ENUM_NAME_ToUnicode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ToUnicode))
44521: (0)          PDF_ENUM_NAME_TopSecret = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TopSecret))
44522: (0)          PDF_ENUM_NAME_Trans = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Trans))
44523: (0)          PDF_ENUM_NAME_TransformMethod = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_TransformMethod))
44524: (0)          PDF_ENUM_NAME_TransformParams = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_TransformParams))
44525: (0)          PDF_ENUM_NAME_Transparency = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_Transparency))
44526: (0)          PDF_ENUM_NAME_TrapNet = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TrapNet))
44527: (0)          PDF_ENUM_NAME_TrimBox = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TrimBox))
44528: (0)          PDF_ENUM_NAME_TrueType = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_TrueType))
44529: (0)          PDF_ENUM_NAME_TrustedMode = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_TrustedMode))
44530: (0)          PDF_ENUM_NAME_Tx = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Tx))
44531: (0)          PDF_ENUM_NAME_Type = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Type))
44532: (0)          PDF_ENUM_NAME_Type0 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Type0))
44533: (0)          PDF_ENUM_NAME_Type1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Type1))
44534: (0)          PDF_ENUM_NAME_Type1C = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Type1C))
44535: (0)          PDF_ENUM_NAME_Type3 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Type3))
44536: (0)          PDF_ENUM_NAME_U = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_U))
44537: (0)          PDF_ENUM_NAME_UE = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_UE))
44538: (0)          PDF_ENUM_NAME_UF = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_UF))
44539: (0)          PDF_ENUM_NAME_URI = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_URI))
44540: (0)          PDF_ENUM_NAME_URL = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_URL))
```

```
44541: (0)          PDF_ENUM_NAME_Unchanged = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Unchanged))
44542: (0)          PDF_ENUM_NAME_Uncover = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Uncover))
44543: (0)          PDF_ENUM_NAME_Underline = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Underline))
44544: (0)          PDF_ENUM_NAME_Unix = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Unix))
44545: (0)          PDF_ENUM_NAME_Usage = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Usage))
44546: (0)          PDF_ENUM_NAME_UseBlackPtComp = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_UseBlackPtComp))
44547: (0)          PDF_ENUM_NAME_UseCMap = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_UseCMap))
44548: (0)          PDF_ENUM_NAME_UseOutlines = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_UseOutlines))
44549: (0)          PDF_ENUM_NAME_UserUnit = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_UserUnit))
44550: (0)          PDF_ENUM_NAME_V = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_V))
44551: (0)          PDF_ENUM_NAME_V2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_V2))
44552: (0)          PDF_ENUM_NAME_VE = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_VE))
44553: (0)          PDF_ENUM_NAME_Version = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Version))
44554: (0)          PDF_ENUM_NAME_Vertices = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Vertices))
44555: (0)          PDF_ENUM_NAME_VerticesPerRow = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_VerticesPerRow))
44556: (0)          PDF_ENUM_NAME_View = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_View))
44557: (0)          PDF_ENUM_NAME_W = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_W))
44558: (0)          PDF_ENUM_NAME_W2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_W2))
44559: (0)          PDF_ENUM_NAME_WMode = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_WMode))
44560: (0)          PDF_ENUM_NAME_WP = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_WP))
44561: (0)          PDF_ENUM_NAME_WT = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_WT))
44562: (0)          PDF_ENUM_NAME_Warichu = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Warichu))
44563: (0)          PDF_ENUM_NAME_Watermark = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Watermark))
44564: (0)          PDF_ENUM_NAME_WhitePoint = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_WhitePoint))
44565: (0)          PDF_ENUM_NAME_Widget = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Widget))
44566: (0)          PDF_ENUM_NAME_Width = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Width))
44567: (0)          PDF_ENUM_NAME_Widths = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Widths))
44568: (0)          PDF_ENUM_NAME_WinAnsiEncoding = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_WinAnsiEncoding))
44569: (0)          PDF_ENUM_NAME_Wipe = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Wipe))
44570: (0)          PDF_ENUM_NAME_XFA = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XFA))
44571: (0)          PDF_ENUM_NAME_XHeight = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XHeight))
44572: (0)          PDF_ENUM_NAME_XML = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XML))
44573: (0)          PDF_ENUM_NAME_XObject = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XObject))
44574: (0)          PDF_ENUM_NAME_XRef = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XRef))
44575: (0)          PDF_ENUM_NAME_XRefStm = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XRefStm))
44576: (0)          PDF_ENUM_NAME_XStep = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XStep))
44577: (0)          PDF_ENUM_NAME_XYZ = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_XYZ))
44578: (0)          PDF_ENUM_NAME_YStep = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_YStep))
44579: (0)          PDF_ENUM_NAME_Yes = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_Yes))
44580: (0)          PDF_ENUM_NAME_ZaDb = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ZaDb))
44581: (0)          PDF_ENUM_NAME_a = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_a))
44582: (0)          PDF_ENUM_NAME_adbe_pkcs7_detached = PdfObj( obj_enum_to_obj(
PDF_ENUM_NAME_adbe_pkcs7_detached))
44583: (0)          PDF_ENUM_NAME_ca = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_ca))
44584: (0)          PDF_ENUM_NAME_n0 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_n0))
44585: (0)          PDF_ENUM_NAME_n1 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_n1))
44586: (0)          PDF_ENUM_NAME_n2 = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_n2))
44587: (0)          PDF_ENUM_NAME_op = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_op))
44588: (0)          PDF_ENUM_NAME_r = PdfObj( obj_enum_to_obj( PDF_ENUM_NAME_r))

----------------------------------------

File 3 - pymupdf.py:

1: (0)              from . import *

----------------------------------------

File 4 - table.py:

1: (0)              """
2: (0)              Copyright (C) 2023 Artifex Software, Inc.
3: (0)              This file is part of PyMuPDF.
4: (0)              PyMuPDF is free software: you can redistribute it and/or modify it under the
5: (0)              terms of the GNU Affero General Public License as published by the Free
```

```
6: (0)              Software Foundation, either version 3 of the License, or (at your option)
7: (0)              any later version.
8: (0)              PyMuPDF is distributed in the hope that it will be useful, but WITHOUT ANY
9: (0)              WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10: (0)             FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more
11: (0)             details.
12: (0)             You should have received a copy of the GNU Affero General Public License
13: (0)             along with MuPDF. If not, see <https://www.gnu.org/licenses/agpl-3.0.en.html>
14: (0)             Alternative licensing terms are available from the licensor.
15: (0)             For commercial licensing, see <https://www.artifex.com/> or contact
16: (0)             Artifex Software, Inc., 39 Mesa Street, Suite 108A, San Francisco,
17: (0)             CA 94129, USA, for further information.
18: (0)             ---------------------------------------------------------------------
19: (0)             Portions of this code have been ported from pdfplumber, see
20: (0)             https://pypi.org/project/pdfplumber/.
21: (0)             The ported code is under the following MIT license:
22: (0)             ---------------------------------------------------------------------
23: (0)             The MIT License (MIT)
24: (0)             Copyright (c) 2015, Jeremy Singer-Vine
25: (0)             Permission is hereby granted, free of charge, to any person obtaining a copy
26: (0)             of this software and associated documentation files (the "Software"), to deal
27: (0)             in the Software without restriction, including without limitation the rights
28: (0)             to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
29: (0)             copies of the Software, and to permit persons to whom the Software is
30: (0)             furnished to do so, subject to the following conditions:
31: (0)             The above copyright notice and this permission notice shall be included in all
32: (0)             copies or substantial portions of the Software.
33: (0)             THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
34: (0)             IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
35: (0)             FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
36: (0)             AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
37: (0)             LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
38: (0)             OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
39: (0)             SOFTWARE.
40: (0)             ---------------------------------------------------------------------
41: (0)             Also see here: https://github.com/jsvine/pdfplumber/blob/stable/LICENSE.txt
42: (0)             ---------------------------------------------------------------------
43: (0)             The porting mainly pertains to files "table.py" and relevant parts of
44: (0)             "utils/text.py" within pdfplumber's repository on Github.
45: (0)             With respect to "text.py", we have removed functions or features that are not
46: (0)             used by table processing. Examples are:
47: (0)             * the text search function
48: (0)             * simple text extraction
49: (0)             * text extraction by lines
50: (0)             Original pdfplumber code does neither detect, nor identify table headers.
51: (0)             This PyMuPDF port adds respective code to the 'Table' class as method
'_get_header'.
52: (0)             This is implemented as new class TableHeader with the properties:
53: (0)             * bbox: A tuple for the header's bbox
54: (0)             * cells: A tuple for each bbox of a column header
55: (0)             * names: A list of strings with column header text
56: (0)             * external: A bool indicating whether the header is outside the table cells.
57: (0)             """
58: (0)             import inspect
59: (0)             import itertools
60: (0)             import string
61: (0)             import html
62: (0)             from collections.abc import Sequence
63: (0)             from dataclasses import dataclass
64: (0)             from operator import itemgetter
65: (0)             from . import (
66: (4)                 Rect,
67: (4)                 Matrix,
68: (4)                 TEXTFLAGS_TEXT,
69: (4)                 TOOLS,
70: (4)                 EMPTY_RECT,
71: (4)                 sRGB_to_pdf,
72: (4)                 Point,
73: (4)                 message,
```

```
74: (0)             )
75: (0)             EDGES = []  # vector graphics from PyMuPDF
76: (0)             CHARS = []  # text characters from PyMuPDF
77: (0)             TEXTPAGE = None
78: (0)             white_spaces = set(string.whitespace)  # for checking white space only cells
79: (0)             class UnsetFloat(float):
80: (4)                 pass
81: (0)             NON_NEGATIVE_SETTINGS = [
82: (4)                 "snap_tolerance",
83: (4)                 "snap_x_tolerance",
84: (4)                 "snap_y_tolerance",
85: (4)                 "join_tolerance",
86: (4)                 "join_x_tolerance",
87: (4)                 "join_y_tolerance",
88: (4)                 "edge_min_length",
89: (4)                 "min_words_vertical",
90: (4)                 "min_words_horizontal",
91: (4)                 "intersection_tolerance",
92: (4)                 "intersection_x_tolerance",
93: (4)                 "intersection_y_tolerance",
94: (0)             ]
95: (0)             TABLE_STRATEGIES = ["lines", "lines_strict", "text", "explicit"]
96: (0)             UNSET = UnsetFloat(0)
97: (0)             DEFAULT_SNAP_TOLERANCE = 3
98: (0)             DEFAULT_JOIN_TOLERANCE = 3
99: (0)             DEFAULT_MIN_WORDS_VERTICAL = 3
100: (0)            DEFAULT_MIN_WORDS_HORIZONTAL = 1
101: (0)            DEFAULT_X_TOLERANCE = 3
102: (0)            DEFAULT_Y_TOLERANCE = 3
103: (0)            DEFAULT_X_DENSITY = 7.25
104: (0)            DEFAULT_Y_DENSITY = 13
105: (0)            bbox_getter = itemgetter("x0", "top", "x1", "bottom")
106: (0)            LIGATURES = {
107: (4)                "ï¬€": "ff",
108: (4)                "ï¬ƒ": "ffi",
109: (4)                "ï¬„": "ffl",
110: (4)                "ï¬": "fi",
111: (4)                "ï¬‚": "fl",
112: (4)                "ï¬†": "st",
113: (4)                "ï¬…": "st",
114: (0)            }
115: (0)            def to_list(collection) -> list:
116: (4)                if isinstance(collection, list):
117: (8)                    return collection
118: (4)                elif isinstance(collection, Sequence):
119: (8)                    return list(collection)
120: (4)                elif hasattr(collection, "to_dict"):
121: (8)                    res = collection.to_dict("records")  # pragma: nocover
122: (8)                    return res
123: (4)                else:
124: (8)                    return list(collection)
125: (0)            class TextMap:
126: (4)                """
127: (4)                A TextMap maps each unicode character in the text to an individual `char`
128: (4)                object (or, in the case of layout-implied whitespace, `None`).
129: (4)                """
130: (4)                def __init__(self, tuples=None) -> None:
131: (8)                    self.tuples = tuples
132: (8)                    self.as_string = "".join(map(itemgetter(0), tuples))
133: (4)                def match_to_dict(
134: (8)                    self,
135: (8)                    m,
136: (8)                    main_group: int = 0,
137: (8)                    return_groups: bool = True,
138: (8)                    return_chars: bool = True,
139: (4)                ) -> dict:
140: (8)                    subset = self.tuples[m.start(main_group) : m.end(main_group)]
141: (8)                    chars = [c for (text, c) in subset if c is not None]
142: (8)                    x0, top, x1, bottom = objects_to_bbox(chars)
```

```
143: (8)                    result = {
144: (12)                       "text": m.group(main_group),
145: (12)                       "x0": x0,
146: (12)                       "top": top,
147: (12)                       "x1": x1,
148: (12)                       "bottom": bottom,
149: (8)                    }
150: (8)                    if return_groups:
151: (12)                       result["groups"] = m.groups()
152: (8)                    if return_chars:
153: (12)                       result["chars"] = chars
154: (8)                    return result
155: (0)            class WordMap:
156: (4)                """
157: (4)                A WordMap maps words->chars.
158: (4)                """
159: (4)                def __init__(self, tuples) -> None:
160: (8)                    self.tuples = tuples
161: (4)                def to_textmap(
162: (8)                    self,
163: (8)                    layout: bool = False,
164: (8)                    layout_width=0,
165: (8)                    layout_height=0,
166: (8)                    layout_width_chars: int = 0,
167: (8)                    layout_height_chars: int = 0,
168: (8)                    x_density=DEFAULT_X_DENSITY,
169: (8)                    y_density=DEFAULT_Y_DENSITY,
170: (8)                    x_shift=0,
171: (8)                    y_shift=0,
172: (8)                    y_tolerance=DEFAULT_Y_TOLERANCE,
173: (8)                    use_text_flow: bool = False,
174: (8)                    presorted: bool = False,
175: (8)                    expand_ligatures: bool = True,
176: (4)                ) -> TextMap:
177: (8)                    """
178: (8)                    Given a list of (word, chars) tuples (i.e., a WordMap), return a list
of
179: (8)                    (char-text, char) tuples (i.e., a TextMap) that can be used to mimic
the
180: (8)                    structural layout of the text on the page(s), using the following
approach:
181: (8)                    - Sort the words by (doctop, x0) if not already sorted.
182: (8)                    - Calculate the initial doctop for the starting page.
183: (8)                    - Cluster the words by doctop (taking `y_tolerance` into account), and
184: (10)                     iterate through them.
185: (8)                    - For each cluster, calculate the distance between that doctop and the
186: (10)                     initial doctop, in points, minus `y_shift`. Divide that distance by
187: (10)                     `y_density` to calculate the minimum number of newlines that should
come
188: (10)                     before this cluster. Append that number of newlines *minus* the
number of
189: (10)                     newlines already appended, with a minimum of one.
190: (8)                    - Then for each cluster, iterate through each word in it. Divide each
191: (10)                     word's x0, minus `x_shift`, by `x_density` to calculate the minimum
192: (10)                     number of characters that should come before this cluster.  Append
that
193: (10)                     number of spaces *minus* the number of characters and spaces already
194: (10)                     appended, with a minimum of one. Then append the word's text.
195: (8)                    - At the termination of each line, add more spaces if necessary to
196: (10)                     mimic `layout_width`.
197: (8)                    - Finally, add newlines to the end if necessary to mimic to
198: (10)                     `layout_height`.
199: (8)                    Note: This approach currently works best for horizontal, left-to-right
200: (8)                    text, but will display all words regardless of orientation. There is
room
201: (8)                    for improvement in better supporting right-to-left text, as well as
202: (8)                    vertical text.
203: (8)                    """
204: (8)                    _textmap = []
```

```
205: (8)                    if not len(self.tuples):
206: (12)                       return TextMap(_textmap)
207: (8)                    expansions = LIGATURES if expand_ligatures else {}
208: (8)                    if layout:
209: (12)                       if layout_width_chars:
210: (16)                           if layout_width:
211: (20)                               raise ValueError(
212: (24)                                   "`layout_width` and `layout_width_chars` cannot both
be set."
213: (20)                               )
214: (12)                       else:
215: (16)                           layout_width_chars = int(round(layout_width / x_density))
216: (12)                       if layout_height_chars:
217: (16)                           if layout_height:
218: (20)                               raise ValueError(
219: (24)                                   "`layout_height` and `layout_height_chars` cannot both
be set."
220: (20)                               )
221: (12)                       else:
222: (16)                           layout_height_chars = int(round(layout_height / y_density))
223: (12)                       blank_line = [(" ", None)] * layout_width_chars
224: (8)                    else:
225: (12)                       blank_line = []
226: (8)                    num_newlines = 0
227: (8)                    words_sorted_doctop = (
228: (12)                       self.tuples
229: (12)                       if presorted or use_text_flow
230: (12)                       else sorted(self.tuples, key=lambda x: float(x[0]["doctop"]))
231: (8)                    )
232: (8)                    first_word = words_sorted_doctop[0][0]
233: (8)                    doctop_start = first_word["doctop"] - first_word["top"]
234: (8)                    for i, ws in enumerate(
235: (12)                       cluster_objects(
236: (16)                           words_sorted_doctop, lambda x: float(x[0]["doctop"]),
y_tolerance
237: (12)                       )
238: (8)                    ):
239: (12)                       y_dist = (
240: (16)                           (ws[0][0]["doctop"] - (doctop_start + y_shift)) / y_density
241: (16)                           if layout
242: (16)                           else 0
243: (12)                       )
244: (12)                       num_newlines_prepend = max(
245: (16)                           int(i > 0),
246: (16)                           round(y_dist) - num_newlines,
247: (12)                       )
248: (12)                       for i in range(num_newlines_prepend):
249: (16)                           if not len(_textmap) or _textmap[-1][0] == "\n":
250: (20)                               _textmap += blank_line
251: (16)                           _textmap.append(("\n", None))
252: (12)                       num_newlines += num_newlines_prepend
253: (12)                       line_len = 0
254: (12)                       line_words_sorted_x0 = (
255: (16)                           ws
256: (16)                           if presorted or use_text_flow
257: (16)                           else sorted(ws, key=lambda x: float(x[0]["x0"]))
258: (12)                       )
259: (12)                       for word, chars in line_words_sorted_x0:
260: (16)                           x_dist = (word["x0"] - x_shift) / x_density if layout else 0
261: (16)                           num_spaces_prepend = max(min(1, line_len), round(x_dist) -
line_len)
262: (16)                           _textmap += [(" ", None)] * num_spaces_prepend
263: (16)                           line_len += num_spaces_prepend
264: (16)                           for c in chars:
265: (20)                               letters = expansions.get(c["text"], c["text"])
266: (20)                               for letter in letters:
267: (24)                                   _textmap.append((letter, c))
268: (24)                                   line_len += 1
269: (12)                       if layout:
```

```
270: (16)                            _textmap += [(" ", None)] * (layout_width_chars - line_len)
271: (8)                    if layout:
272: (12)                       num_newlines_append = layout_height_chars - (num_newlines + 1)
273: (12)                       for i in range(num_newlines_append):
274: (16)                           if i > 0:
275: (20)                               _textmap += blank_line
276: (16)                           _textmap.append(("\n", None))
277: (12)                       if _textmap[-1] == ("\n", None):
278: (16)                           _textmap = _textmap[:-1]
279: (8)                    return TextMap(_textmap)
280: (0)            class WordExtractor:
281: (4)                def __init__(
282: (8)                    self,
283: (8)                    x_tolerance=DEFAULT_X_TOLERANCE,
284: (8)                    y_tolerance=DEFAULT_Y_TOLERANCE,
285: (8)                    keep_blank_chars: bool = False,
286: (8)                    use_text_flow=False,
287: (8)                    horizontal_ltr=True,  # Should words be read left-to-right?
288: (8)                    vertical_ttb=False,  # Should vertical words be read top-to-bottom?
289: (8)                    extra_attrs=None,
290: (8)                    split_at_punctuation=False,
291: (8)                    expand_ligatures=True,
292: (4)                ):
293: (8)                    self.x_tolerance = x_tolerance
294: (8)                    self.y_tolerance = y_tolerance
295: (8)                    self.keep_blank_chars = keep_blank_chars
296: (8)                    self.use_text_flow = use_text_flow
297: (8)                    self.horizontal_ltr = horizontal_ltr
298: (8)                    self.vertical_ttb = vertical_ttb
299: (8)                    self.extra_attrs = [] if extra_attrs is None else extra_attrs
300: (8)                    self.split_at_punctuation = (
301: (12)                       string.punctuation
302: (12)                       if split_at_punctuation is True
303: (12)                       else (split_at_punctuation or "")
304: (8)                    )
305: (8)                    self.expansions = LIGATURES if expand_ligatures else {}
306: (4)                def merge_chars(self, ordered_chars: list):
307: (8)                    x0, top, x1, bottom = objects_to_bbox(ordered_chars)
308: (8)                    doctop_adj = ordered_chars[0]["doctop"] - ordered_chars[0]["top"]
309: (8)                    upright = ordered_chars[0]["upright"]
310: (8)                    direction = 1 if (self.horizontal_ltr if upright else
self.vertical_ttb) else -1
311: (8)                    matrix = ordered_chars[0]["matrix"]
312: (8)                    rotation = 0
313: (8)                    if not upright and matrix[1] < 0:
314: (12)                       ordered_chars = reversed(ordered_chars)
315: (12)                       rotation = 270
316: (8)                    if matrix[0] < 0 and matrix[3] < 0:
317: (12)                       rotation = 180
318: (8)                    elif matrix[1] > 0:
319: (12)                       rotation = 90
320: (8)                    word = {
321: (12)                       "text": "".join(
322: (16)                           self.expansions.get(c["text"], c["text"]) for c in
ordered_chars
323: (12)                       ),
324: (12)                       "x0": x0,
325: (12)                       "x1": x1,
326: (12)                       "top": top,
327: (12)                       "doctop": top + doctop_adj,
328: (12)                       "bottom": bottom,
329: (12)                       "upright": upright,
330: (12)                       "direction": direction,
331: (12)                       "rotation": rotation,
332: (8)                    }
333: (8)                    for key in self.extra_attrs:
334: (12)                       word[key] = ordered_chars[0][key]
335: (8)                    return word
336: (4)                def char_begins_new_word(
```

```
(8)                 self,
(8)                 prev_char,
(8)                 curr_char,
(4)             ) -> bool:
(8)                 """This method takes several factors into account to determine if
(8)                 `curr_char` represents the beginning of a new word:
(8)                 - Whether the text is "upright" (i.e., non-rotated)
(8)                 - Whether the user has specified that horizontal text runs
(10)                  left-to-right (default) or right-to-left, as represented by
(10)                  self.horizontal_ltr
(8)                 - Whether the user has specified that vertical text the text runs
(10)                  top-to-bottom (default) or bottom-to-top, as represented by
(10)                  self.vertical_ttb
(8)                 - The x0, top, x1, and bottom attributes of prev_char and
(10)                  curr_char
(8)                 - The self.x_tolerance and self.y_tolerance settings. Note: In
(10)                  this case, x/y refer to those directions for non-rotated text.
(10)                  For vertical text, they are flipped. A more accurate terminology
(10)                  might be "*intra*line character distance tolerance" and
(10)                  "*inter*line character distance tolerance"
(8)                 An important note: The *intra*line distance is measured from the
(8)                 *end* of the previous character to the *beginning* of the current
(8)                 character, while the *inter*line distance is measured from the
(8)                 *top* of the previous character to the *top* of the next
(8)                 character. The reasons for this are partly repository-historical,
(8)                 and partly logical, as successive text lines' bounding boxes often
(8)                 overlap slightly (and we don't want that overlap to be interpreted
(8)                 as the two lines being the same line).
(8)                 The upright-ness of the character determines the attributes to
(8)                 compare, while horizontal_ltr/vertical_ttb determine the direction
(8)                 of the comparison.
(8)                 """
(8)                 if curr_char["upright"]:
(12)                    x = self.x_tolerance
(12)                    y = self.y_tolerance
(12)                    ay = prev_char["top"]
(12)                    cy = curr_char["top"]
(12)                    if self.horizontal_ltr:
(16)                        ax = prev_char["x0"]
(16)                        bx = prev_char["x1"]
(16)                        cx = curr_char["x0"]
(12)                    else:
(16)                        ax = -prev_char["x1"]
(16)                        bx = -prev_char["x0"]
(16)                        cx = -curr_char["x1"]
(8)                 else:
(12)                    x = self.y_tolerance
(12)                    y = self.x_tolerance
(12)                    ay = prev_char["x0"]
(12)                    cy = curr_char["x0"]
(12)                    if self.vertical_ttb:
(16)                        ax = prev_char["top"]
(16)                        bx = prev_char["bottom"]
(16)                        cx = curr_char["top"]
(12)                    else:
(16)                        ax = -prev_char["bottom"]
(16)                        bx = -prev_char["top"]
(16)                        cx = -curr_char["bottom"]
(8)                 return bool(
(12)                    (cx < ax)
(12)                    or (cx > bx + x)
(12)                    or (cy > ay + y)
(8)                 )
(4)             def iter_chars_to_words(self, ordered_chars):
(8)                 current_word: list = []
(8)                 def start_next_word(new_char=None):
(12)                    nonlocal current_word
(12)                    if current_word:
(16)                        yield current_word
```

```
406: (12)                        current_word = [] if new_char is None else [new_char]
407: (8)                    for char in ordered_chars:
408: (12)                        text = char["text"]
409: (12)                        if not self.keep_blank_chars and text.isspace():
410: (16)                            yield from start_next_word(None)
411: (12)                        elif text in self.split_at_punctuation:
412: (16)                            yield from start_next_word(char)
413: (16)                            yield from start_next_word(None)
414: (12)                        elif current_word and self.char_begins_new_word(current_word[-1],
char):
415: (16)                            yield from start_next_word(char)
416: (12)                        else:
417: (16)                            current_word.append(char)
418: (8)                    if current_word:
419: (12)                        yield current_word
420: (4)                def iter_sort_chars(self, chars):
421: (8)                    def upright_key(x) -> int:
422: (12)                        return -int(x["upright"])
423: (8)                    for upright_cluster in cluster_objects(list(chars), upright_key, 0):
424: (12)                        upright = upright_cluster[0]["upright"]
425: (12)                        cluster_key = "doctop" if upright else "x0"
426: (12)                        subclusters = cluster_objects(
427: (16)                            upright_cluster, itemgetter(cluster_key), self.y_tolerance
428: (12)                        )
429: (12)                        for sc in subclusters:
430: (16)                            sort_key = "x0" if upright else "doctop"
431: (16)                            to_yield = sorted(sc, key=itemgetter(sort_key))
432: (16)                            if not (self.horizontal_ltr if upright else
self.vertical_ttb):
433: (20)                                yield from reversed(to_yield)
434: (16)                            else:
435: (20)                                yield from to_yield
436: (4)                def iter_extract_tuples(self, chars):
437: (8)                    ordered_chars = chars if self.use_text_flow else
self.iter_sort_chars(chars)
438: (8)                    grouping_key = itemgetter("upright", *self.extra_attrs)
439: (8)                    grouped_chars = itertools.groupby(ordered_chars, grouping_key)
440: (8)                    for keyvals, char_group in grouped_chars:
441: (12)                        for word_chars in self.iter_chars_to_words(char_group):
442: (16)                            yield (self.merge_chars(word_chars), word_chars)
443: (4)                def extract_wordmap(self, chars) -> WordMap:
444: (8)                    return WordMap(list(self.iter_extract_tuples(chars)))
445: (4)                def extract_words(self, chars: list) -> list:
446: (8)                    words = list(word for word, word_chars in
self.iter_extract_tuples(chars))
447: (8)                    return words
448: (0)            def extract_words(chars: list, **kwargs) -> list:
449: (4)                return WordExtractor(**kwargs).extract_words(chars)
450: (0)            TEXTMAP_KWARGS = inspect.signature(WordMap.to_textmap).parameters.keys()
451: (0)            WORD_EXTRACTOR_KWARGS = inspect.signature(WordExtractor).parameters.keys()
452: (0)            def chars_to_textmap(chars: list, **kwargs) -> TextMap:
453: (4)                kwargs.update({"presorted": True})
454: (4)                extractor = WordExtractor(
455: (8)                    **{k: kwargs[k] for k in WORD_EXTRACTOR_KWARGS if k in kwargs}
456: (4)                )
457: (4)                wordmap = extractor.extract_wordmap(chars)
458: (4)                textmap = wordmap.to_textmap(
459: (8)                    **{k: kwargs[k] for k in TEXTMAP_KWARGS if k in kwargs}
460: (4)                )
461: (4)                return textmap
462: (0)            def extract_text(chars: list, **kwargs) -> str:
463: (4)                chars = to_list(chars)
464: (4)                if len(chars) == 0:
465: (8)                    return ""
466: (4)                if kwargs.get("layout"):
467: (8)                    return chars_to_textmap(chars, **kwargs).as_string
468: (4)                else:
469: (8)                    y_tolerance = kwargs.get("y_tolerance", DEFAULT_Y_TOLERANCE)
470: (8)                    extractor = WordExtractor(
```

```
471: (12)                        **{k: kwargs[k] for k in WORD_EXTRACTOR_KWARGS if k in kwargs}
472: (8)                    )
473: (8)                    words = extractor.extract_words(chars)
474: (8)                    if words:
475: (12)                        rotation = words[0]["rotation"]  # rotation cannot change within a
cell
476: (8)                    else:
477: (12)                        rotation = 0
478: (8)                    if rotation == 90:
479: (12)                        words.sort(key=lambda w: (w["x1"], -w["top"]))
480: (12)                        lines = " ".join([w["text"] for w in words])
481: (8)                    elif rotation == 270:
482: (12)                        words.sort(key=lambda w: (-w["x1"], w["top"]))
483: (12)                        lines = " ".join([w["text"] for w in words])
484: (8)                    else:
485: (12)                        lines = cluster_objects(words, itemgetter("doctop"), y_tolerance)
486: (12)                        lines = "\n".join(" ".join(word["text"] for word in line) for line
in lines)
487: (12)                        if rotation == 180:  # needs extra treatment
488: (16)                            lines = "".join([(c if c != "\n" else " ") for c in
reversed(lines)])
489: (8)                    return lines
490: (0)            def collate_line(
491: (4)                line_chars: list,
492: (4)                tolerance=DEFAULT_X_TOLERANCE,
493: (0)            ) -> str:
494: (4)                coll = ""
495: (4)                last_x1 = None
496: (4)                for char in sorted(line_chars, key=itemgetter("x0")):
497: (8)                    if (last_x1 is not None) and (char["x0"] > (last_x1 + tolerance)):
498: (12)                        coll += " "
499: (8)                    last_x1 = char["x1"]
500: (8)                    coll += char["text"]
501: (4)                return coll
502: (0)            def dedupe_chars(chars: list, tolerance=1) -> list:
503: (4)                """
504: (4)                Removes duplicate chars â€”Â those sharing the same text, fontname, size,
505: (4)                and positioning (within `tolerance`) as other characters in the set.
506: (4)                """
507: (4)                key = itemgetter("fontname", "size", "upright", "text")
508: (4)                pos_key = itemgetter("doctop", "x0")
509: (4)                def yield_unique_chars(chars: list):
510: (8)                    sorted_chars = sorted(chars, key=key)
511: (8)                    for grp, grp_chars in itertools.groupby(sorted_chars, key=key):
512: (12)                        for y_cluster in cluster_objects(
513: (16)                            list(grp_chars), itemgetter("doctop"), tolerance
514: (12)                        ):
515: (16)                            for x_cluster in cluster_objects(
516: (20)                                y_cluster, itemgetter("x0"), tolerance
517: (16)                            ):
518: (20)                                yield sorted(x_cluster, key=pos_key)[0]
519: (4)                deduped = yield_unique_chars(chars)
520: (4)                return sorted(deduped, key=chars.index)
521: (0)            def line_to_edge(line):
522: (4)                edge = dict(line)
523: (4)                edge["orientation"] = "h" if (line["top"] == line["bottom"]) else "v"
524: (4)                return edge
525: (0)            def rect_to_edges(rect) -> list:
526: (4)                top, bottom, left, right = [dict(rect) for x in range(4)]
527: (4)                top.update(
528: (8)                    {
529: (12)                        "object_type": "rect_edge",
530: (12)                        "height": 0,
531: (12)                        "y0": rect["y1"],
532: (12)                        "bottom": rect["top"],
533: (12)                        "orientation": "h",
534: (8)                    }
535: (4)                )
536: (4)                bottom.update(
```

```
537: (8)                    {
538: (12)                       "object_type": "rect_edge",
539: (12)                       "height": 0,
540: (12)                       "y1": rect["y0"],
541: (12)                       "top": rect["top"] + rect["height"],
542: (12)                       "doctop": rect["doctop"] + rect["height"],
543: (12)                       "orientation": "h",
544: (8)                    }
545: (4)                )
546: (4)                left.update(
547: (8)                    {
548: (12)                       "object_type": "rect_edge",
549: (12)                       "width": 0,
550: (12)                       "x1": rect["x0"],
551: (12)                       "orientation": "v",
552: (8)                    }
553: (4)                )
554: (4)                right.update(
555: (8)                    {
556: (12)                       "object_type": "rect_edge",
557: (12)                       "width": 0,
558: (12)                       "x0": rect["x1"],
559: (12)                       "orientation": "v",
560: (8)                    }
561: (4)                )
562: (4)                return [top, bottom, left, right]
563: (0)            def curve_to_edges(curve) -> list:
564: (4)                point_pairs = zip(curve["pts"], curve["pts"][1:])
565: (4)                return [
566: (8)                    {
567: (12)                       "object_type": "curve_edge",
568: (12)                       "x0": min(p0[0], p1[0]),
569: (12)                       "x1": max(p0[0], p1[0]),
570: (12)                       "top": min(p0[1], p1[1]),
571: (12)                       "doctop": min(p0[1], p1[1]) + (curve["doctop"] - curve["top"]),
572: (12)                       "bottom": max(p0[1], p1[1]),
573: (12)                       "width": abs(p0[0] - p1[0]),
574: (12)                       "height": abs(p0[1] - p1[1]),
575: (12)                       "orientation": "v" if p0[0] == p1[0] else ("h" if p0[1] == p1[1]
else None),
576: (8)                    }
577: (8)                    for p0, p1 in point_pairs
578: (4)                ]
579: (0)            def obj_to_edges(obj) -> list:
580: (4)                t = obj["object_type"]
581: (4)                if "_edge" in t:
582: (8)                    return [obj]
583: (4)                elif t == "line":
584: (8)                    return [line_to_edge(obj)]
585: (4)                else:
586: (8)                    return {"rect": rect_to_edges, "curve": curve_to_edges}[t](obj)
587: (0)            def filter_edges(
588: (4)                edges,
589: (4)                orientation=None,
590: (4)                edge_type=None,
591: (4)                min_length=1,
592: (0)            ) -> list:
593: (4)                if orientation not in ("v", "h", None):
594: (8)                    raise ValueError("Orientation must be 'v' or 'h'")
595: (4)                def test(e) -> bool:
596: (8)                    dim = "height" if e["orientation"] == "v" else "width"
597: (8)                    et_correct = e["object_type"] == edge_type if edge_type is not None
else True
598: (8)                    orient_correct = orientation is None or e["orientation"] ==
orientation
599: (8)                    return bool(et_correct and orient_correct and (e[dim] >= min_length))
600: (4)                return list(filter(test, edges))
601: (0)            def cluster_list(xs, tolerance=0) -> list:
602: (4)                if tolerance == 0:
```

```
603: (8)                    return [[x] for x in sorted(xs)]
604: (4)                if len(xs) < 2:
605: (8)                    return [[x] for x in sorted(xs)]
606: (4)                groups = []
607: (4)                xs = list(sorted(xs))
608: (4)                current_group = [xs[0]]
609: (4)                last = xs[0]
610: (4)                for x in xs[1:]:
611: (8)                    if x <= (last + tolerance):
612: (12)                       current_group.append(x)
613: (8)                    else:
614: (12)                       groups.append(current_group)
615: (12)                       current_group = [x]
616: (8)                    last = x
617: (4)                groups.append(current_group)
618: (4)                return groups
619: (0)            def make_cluster_dict(values, tolerance) -> dict:
620: (4)                clusters = cluster_list(list(set(values)), tolerance)
621: (4)                nested_tuples = [
622: (8)                    [(val, i) for val in value_cluster] for i, value_cluster in
enumerate(clusters)
623: (4)                ]
624: (4)                return dict(itertools.chain(*nested_tuples))
625: (0)            def cluster_objects(xs, key_fn, tolerance) -> list:
626: (4)                if not callable(key_fn):
627: (8)                    key_fn = itemgetter(key_fn)
628: (4)                values = map(key_fn, xs)
629: (4)                cluster_dict = make_cluster_dict(values, tolerance)
630: (4)                get_0, get_1 = itemgetter(0), itemgetter(1)
631: (4)                cluster_tuples = sorted(((x, cluster_dict.get(key_fn(x))) for x in xs),
key=get_1)
632: (4)                grouped = itertools.groupby(cluster_tuples, key=get_1)
633: (4)                return [list(map(get_0, v)) for k, v in grouped]
634: (0)            def move_object(obj, axis: str, value):
635: (4)                assert axis in ("h", "v")
636: (4)                if axis == "h":
637: (8)                    new_items = [
638: (12)                       ("x0", obj["x0"] + value),
639: (12)                       ("x1", obj["x1"] + value),
640: (8)                    ]
641: (4)                if axis == "v":
642: (8)                    new_items = [
643: (12)                       ("top", obj["top"] + value),
644: (12)                       ("bottom", obj["bottom"] + value),
645: (8)                    ]
646: (8)                    if "doctop" in obj:
647: (12)                       new_items += [("doctop", obj["doctop"] + value)]
648: (8)                    if "y0" in obj:
649: (12)                       new_items += [
650: (16)                           ("y0", obj["y0"] - value),
651: (16)                           ("y1", obj["y1"] - value),
652: (12)                       ]
653: (4)                return obj.__class__(tuple(obj.items()) + tuple(new_items))
654: (0)            def snap_objects(objs, attr: str, tolerance) -> list:
655: (4)                axis = {"x0": "h", "x1": "h", "top": "v", "bottom": "v"}[attr]
656: (4)                list_objs = list(objs)
657: (4)                clusters = cluster_objects(list_objs, itemgetter(attr), tolerance)
658: (4)                avgs = [sum(map(itemgetter(attr), cluster)) / len(cluster) for cluster in
clusters]
659: (4)                snapped_clusters = [
660: (8)                    [move_object(obj, axis, avg - obj[attr]) for obj in cluster]
661: (8)                    for cluster, avg in zip(clusters, avgs)
662: (4)                ]
663: (4)                return list(itertools.chain(*snapped_clusters))
664: (0)            def snap_edges(
665: (4)                edges,
666: (4)                x_tolerance=DEFAULT_SNAP_TOLERANCE,
667: (4)                y_tolerance=DEFAULT_SNAP_TOLERANCE,
668: (0)            ):
```

```
669: (4)                """
670: (4)                Given a list of edges, snap any within `tolerance` pixels of one another
671: (4)                to their positional average.
672: (4)                """
673: (4)                by_orientation = {"v": [], "h": []}
674: (4)                for e in edges:
675: (8)                    by_orientation[e["orientation"]].append(e)
676: (4)                snapped_v = snap_objects(by_orientation["v"], "x0", x_tolerance)
677: (4)                snapped_h = snap_objects(by_orientation["h"], "top", y_tolerance)
678: (4)                return snapped_v + snapped_h
679: (0)            def resize_object(obj, key: str, value):
680: (4)                assert key in ("x0", "x1", "top", "bottom")
681: (4)                old_value = obj[key]
682: (4)                diff = value - old_value
683: (4)                new_items = [
684: (8)                    (key, value),
685: (4)                ]
686: (4)                if key == "x0":
687: (8)                    assert value <= obj["x1"]
688: (8)                    new_items.append(("width", obj["x1"] - value))
689: (4)                elif key == "x1":
690: (8)                    assert value >= obj["x0"]
691: (8)                    new_items.append(("width", value - obj["x0"]))
692: (4)                elif key == "top":
693: (8)                    assert value <= obj["bottom"]
694: (8)                    new_items.append(("doctop", obj["doctop"] + diff))
695: (8)                    new_items.append(("height", obj["height"] - diff))
696: (8)                    if "y1" in obj:
697: (12)                       new_items.append(("y1", obj["y1"] - diff))
698: (4)                elif key == "bottom":
699: (8)                    assert value >= obj["top"]
700: (8)                    new_items.append(("height", obj["height"] + diff))
701: (8)                    if "y0" in obj:
702: (12)                       new_items.append(("y0", obj["y0"] - diff))
703: (4)                return obj.__class__(tuple(obj.items()) + tuple(new_items))
704: (0)            def join_edge_group(edges, orientation: str,
tolerance=DEFAULT_JOIN_TOLERANCE):
705: (4)                """
706: (4)                Given a list of edges along the same infinite line, join those that
707: (4)                are within `tolerance` pixels of one another.
708: (4)                """
709: (4)                if orientation == "h":
710: (8)                    min_prop, max_prop = "x0", "x1"
711: (4)                elif orientation == "v":
712: (8)                    min_prop, max_prop = "top", "bottom"
713: (4)                else:
714: (8)                    raise ValueError("Orientation must be 'v' or 'h'")
715: (4)                sorted_edges = list(sorted(edges, key=itemgetter(min_prop)))
716: (4)                joined = [sorted_edges[0]]
717: (4)                for e in sorted_edges[1:]:
718: (8)                    last = joined[-1]
719: (8)                    if e[min_prop] <= (last[max_prop] + tolerance):
720: (12)                       if e[max_prop] > last[max_prop]:
721: (16)                           joined[-1] = resize_object(last, max_prop, e[max_prop])
722: (8)                    else:
723: (12)                       joined.append(e)
724: (4)                return joined
725: (0)            def merge_edges(
726: (4)                edges,
727: (4)                snap_x_tolerance,
728: (4)                snap_y_tolerance,
729: (4)                join_x_tolerance,
730: (4)                join_y_tolerance,
731: (0)            ):
732: (4)                """
733: (4)                Using the `snap_edges` and `join_edge_group` methods above,
734: (4)                merge a list of edges into a more "seamless" list.
735: (4)                """
736: (4)                def get_group(edge):
```

```
(8)                     if edge["orientation"] == "h":
(12)                        return ("h", edge["top"])
(8)                     else:
(12)                        return ("v", edge["x0"])
(4)                 if snap_x_tolerance > 0 or snap_y_tolerance > 0:
(8)                     edges = snap_edges(edges, snap_x_tolerance, snap_y_tolerance)
(4)                 _sorted = sorted(edges, key=get_group)
(4)                 edge_groups = itertools.groupby(_sorted, key=get_group)
(4)                 edge_gen = (
(8)                     join_edge_group(
(12)                        items, k[0], (join_x_tolerance if k[0] == "h" else
join_y_tolerance)
(8)                     )
(8)                     for k, items in edge_groups
(4)                 )
(4)                 edges = list(itertools.chain(*edge_gen))
(4)                 return edges
(0)             def bbox_to_rect(bbox) -> dict:
(4)                 """
(4)                 Return the rectangle (i.e a dict with keys "x0", "top", "x1",
(4)                 "bottom") for an object.
(4)                 """
(4)                 return {"x0": bbox[0], "top": bbox[1], "x1": bbox[2], "bottom": bbox[3]}
(0)             def objects_to_rect(objects) -> dict:
(4)                 """
(4)                 Given an iterable of objects, return the smallest rectangle (i.e. a
(4)                 dict with "x0", "top", "x1", and "bottom" keys) that contains them
(4)                 all.
(4)                 """
(4)                 return bbox_to_rect(objects_to_bbox(objects))
(0)             def merge_bboxes(bboxes):
(4)                 """
(4)                 Given an iterable of bounding boxes, return the smallest bounding box
(4)                 that contains them all.
(4)                 """
(4)                 x0, top, x1, bottom = zip(*bboxes)
(4)                 return (min(x0), min(top), max(x1), max(bottom))
(0)             def objects_to_bbox(objects):
(4)                 """
(4)                 Given an iterable of objects, return the smallest bounding box that
(4)                 contains them all.
(4)                 """
(4)                 return merge_bboxes(map(bbox_getter, objects))
(0)             def words_to_edges_h(words, word_threshold: int =
DEFAULT_MIN_WORDS_HORIZONTAL):
(4)                 """
(4)                 Find (imaginary) horizontal lines that connect the tops
(4)                 of at least `word_threshold` words.
(4)                 """
(4)                 by_top = cluster_objects(words, itemgetter("top"), 1)
(4)                 large_clusters = filter(lambda x: len(x) >= word_threshold, by_top)
(4)                 rects = list(map(objects_to_rect, large_clusters))
(4)                 if len(rects) == 0:
(8)                     return []
(4)                 min_x0 = min(map(itemgetter("x0"), rects))
(4)                 max_x1 = max(map(itemgetter("x1"), rects))
(4)                 edges = []
(4)                 for r in rects:
(8)                     edges += [
(12)                        {
(16)                            "x0": min_x0,
(16)                            "x1": max_x1,
(16)                            "top": r["top"],
(16)                            "bottom": r["top"],
(16)                            "width": max_x1 - min_x0,
(16)                            "orientation": "h",
(12)                        },
(12)                        {
(16)                            "x0": min_x0,
```

```
804: (16)                            "x1": max_x1,
805: (16)                            "top": r["bottom"],
806: (16)                            "bottom": r["bottom"],
807: (16)                            "width": max_x1 - min_x0,
808: (16)                            "orientation": "h",
809: (12)                        },
810: (8)                    ]
811: (4)                return edges
812: (0)            def get_bbox_overlap(a, b):
813: (4)                a_left, a_top, a_right, a_bottom = a
814: (4)                b_left, b_top, b_right, b_bottom = b
815: (4)                o_left = max(a_left, b_left)
816: (4)                o_right = min(a_right, b_right)
817: (4)                o_bottom = min(a_bottom, b_bottom)
818: (4)                o_top = max(a_top, b_top)
819: (4)                o_width = o_right - o_left
820: (4)                o_height = o_bottom - o_top
821: (4)                if o_height >= 0 and o_width >= 0 and o_height + o_width > 0:
822: (8)                    return (o_left, o_top, o_right, o_bottom)
823: (4)                else:
824: (8)                    return None
825: (0)            def words_to_edges_v(words, word_threshold: int = DEFAULT_MIN_WORDS_VERTICAL):
826: (4)                """
827: (4)                Find (imaginary) vertical lines that connect the left, right, or
828: (4)                center of at least `word_threshold` words.
829: (4)                """
830: (4)                by_x0 = cluster_objects(words, itemgetter("x0"), 1)
831: (4)                by_x1 = cluster_objects(words, itemgetter("x1"), 1)
832: (4)                def get_center(word):
833: (8)                    return float(word["x0"] + word["x1"]) / 2
834: (4)                by_center = cluster_objects(words, get_center, 1)
835: (4)                clusters = by_x0 + by_x1 + by_center
836: (4)                sorted_clusters = sorted(clusters, key=lambda x: -len(x))
837: (4)                large_clusters = filter(lambda x: len(x) >= word_threshold,
sorted_clusters)
838: (4)                bboxes = list(map(objects_to_bbox, large_clusters))
839: (4)                condensed_bboxes = []
840: (4)                for bbox in bboxes:
841: (8)                    overlap = any(get_bbox_overlap(bbox, c) for c in condensed_bboxes)
842: (8)                    if not overlap:
843: (12)                       condensed_bboxes.append(bbox)
844: (4)                if len(condensed_bboxes) == 0:
845: (8)                    return []
846: (4)                condensed_rects = map(bbox_to_rect, condensed_bboxes)
847: (4)                sorted_rects = list(sorted(condensed_rects, key=itemgetter("x0")))
848: (4)                max_x1 = max(map(itemgetter("x1"), sorted_rects))
849: (4)                min_top = min(map(itemgetter("top"), sorted_rects))
850: (4)                max_bottom = max(map(itemgetter("bottom"), sorted_rects))
851: (4)                return [
852: (8)                    {
853: (12)                       "x0": b["x0"],
854: (12)                       "x1": b["x0"],
855: (12)                       "top": min_top,
856: (12)                       "bottom": max_bottom,
857: (12)                       "height": max_bottom - min_top,
858: (12)                       "orientation": "v",
859: (8)                    }
860: (8)                    for b in sorted_rects
861: (4)                ] + [
862: (8)                    {
863: (12)                       "x0": max_x1,
864: (12)                       "x1": max_x1,
865: (12)                       "top": min_top,
866: (12)                       "bottom": max_bottom,
867: (12)                       "height": max_bottom - min_top,
868: (12)                       "orientation": "v",
869: (8)                    }
870: (4)                ]
871: (0)            def edges_to_intersections(edges, x_tolerance=1, y_tolerance=1) -> dict:
```

```
    """
    Given a list of edges, return the points at which they intersect
    within `tolerance` pixels.
    """
    intersections = {}
    v_edges, h_edges = [
        list(filter(lambda x: x["orientation"] == o, edges)) for o in ("v", "h")
    ]
    for v in sorted(v_edges, key=itemgetter("x0", "top")):
        for h in sorted(h_edges, key=itemgetter("top", "x0")):
            if (
                (v["top"] <= (h["top"] + y_tolerance))
                and (v["bottom"] >= (h["top"] - y_tolerance))
                and (v["x0"] >= (h["x0"] - x_tolerance))
                and (v["x0"] <= (h["x1"] + x_tolerance))
            ):
                vertex = (v["x0"], h["top"])
                if vertex not in intersections:
                    intersections[vertex] = {"v": [], "h": []}
                intersections[vertex]["v"].append(v)
                intersections[vertex]["h"].append(h)
    return intersections
def obj_to_bbox(obj):
    """
    Return the bounding box for an object.
    """
    return bbox_getter(obj)
def intersections_to_cells(intersections):
    """
    Given a list of points (`intersections`), return all rectangular "cells"
    that those points describe.
    `intersections` should be a dictionary with (x0, top) tuples as keys,
    and a list of edge objects as values. The edge objects should correspond
    to the edges that touch the intersection.
    """
    def edge_connects(p1, p2) -> bool:
        def edges_to_set(edges):
            return set(map(obj_to_bbox, edges))
        if p1[0] == p2[0]:
            common = edges_to_set(intersections[p1]["v"]).intersection(
                edges_to_set(intersections[p2]["v"])
            )
            if len(common):
                return True
        if p1[1] == p2[1]:
            common = edges_to_set(intersections[p1]["h"]).intersection(
                edges_to_set(intersections[p2]["h"])
            )
            if len(common):
                return True
        return False
    points = list(sorted(intersections.keys()))
    n_points = len(points)
    def find_smallest_cell(points, i: int):
        if i == n_points - 1:
            return None
        pt = points[i]
        rest = points[i + 1 :]
        below = [x for x in rest if x[0] == pt[0]]
        right = [x for x in rest if x[1] == pt[1]]
        for below_pt in below:
            if not edge_connects(pt, below_pt):
                continue
            for right_pt in right:
                if not edge_connects(pt, right_pt):
                    continue
                bottom_right = (right_pt[0], below_pt[1])
                if (
```

```
(20)                                (bottom_right in intersections)
(20)                                and edge_connects(bottom_right, right_pt)
(20)                                and edge_connects(bottom_right, below_pt)
(16)                            ):
(20)                                return (pt[0], pt[1], bottom_right[0], bottom_right[1])
(8)                     return None
(4)                 cell_gen = (find_smallest_cell(points, i) for i in range(len(points)))
(4)                 return list(filter(None, cell_gen))
(0)             def cells_to_tables(page, cells) -> list:
(4)                 """
(4)                 Given a list of bounding boxes (`cells`), return a list of tables that
(4)                 hold those cells most simply (and contiguously).
(4)                 """
(4)                 def bbox_to_corners(bbox) -> tuple:
(8)                     x0, top, x1, bottom = bbox
(8)                     return ((x0, top), (x0, bottom), (x1, top), (x1, bottom))
(4)                 remaining_cells = list(cells)
(4)                 current_corners = set()
(4)                 current_cells = []
(4)                 tables = []
(4)                 while len(remaining_cells):
(8)                     initial_cell_count = len(current_cells)
(8)                     for cell in list(remaining_cells):
(12)                        cell_corners = bbox_to_corners(cell)
(12)                        if len(current_cells) == 0:
(16)                            current_corners |= set(cell_corners)
(16)                            current_cells.append(cell)
(16)                            remaining_cells.remove(cell)
(12)                        else:
(16)                            corner_count = sum(c in current_corners for c in cell_corners)
(16)                            if corner_count > 0:
(20)                                current_corners |= set(cell_corners)
(20)                                current_cells.append(cell)
(20)                                remaining_cells.remove(cell)
(8)                     if len(current_cells) == initial_cell_count:
(12)                        tables.append(list(current_cells))
(12)                        current_corners.clear()
(12)                        current_cells.clear()
(4)                 if len(current_cells):
(8)                     tables.append(list(current_cells))
(4)                 for i in range(len(tables) - 1, -1, -1):
(8)                     r = EMPTY_RECT()
(8)                     x1_vals = set()
(8)                     x0_vals = set()
(8)                     for c in tables[i]:
(12)                        r |= c
(12)                        x1_vals.add(c[2])
(12)                        x0_vals.add(c[0])
(8)                     if (
(12)                        len(x1_vals) < 2
(12)                        or len(x0_vals) < 2
(12)                        or white_spaces.issuperset(
(16)                            page.get_textbox(
(20)                                r,
(20)                                textpage=TEXTPAGE,
(16)                            )
(12)                        )
(8)                     ):
(12)                        del tables[i]
(4)                 _sorted = sorted(tables, key=lambda t: min((c[1], c[0]) for c in t))
(4)                 return _sorted
(0)             class CellGroup:
(4)                 def __init__(self, cells):
(8)                     self.cells = cells
(8)                     self.bbox = (
(12)                        min(map(itemgetter(0), filter(None, cells))),
(12)                        min(map(itemgetter(1), filter(None, cells))),
(12)                        max(map(itemgetter(2), filter(None, cells))),
(12)                        max(map(itemgetter(3), filter(None, cells))),
```

```
1009: (8)                )
1010: (0)        class TableRow(CellGroup):
1011: (4)            pass
1012: (0)        class TableHeader:
1013: (4)            """PyMuPDF extension containing the identified table header."""
1014: (4)            def __init__(self, bbox, cells, names, above):
1015: (8)                self.bbox = bbox
1016: (8)                self.cells = cells
1017: (8)                self.names = names
1018: (8)                self.external = above
1019: (0)        class Table:
1020: (4)            def __init__(self, page, cells):
1021: (8)                self.page = page
1022: (8)                self.cells = cells
1023: (8)                self.header = self._get_header()  # PyMuPDF extension
1024: (4)            @property
1025: (4)            def bbox(self):
1026: (8)                c = self.cells
1027: (8)                return (
1028: (12)                   min(map(itemgetter(0), c)),
1029: (12)                   min(map(itemgetter(1), c)),
1030: (12)                   max(map(itemgetter(2), c)),
1031: (12)                   max(map(itemgetter(3), c)),
1032: (8)                )
1033: (4)            @property
1034: (4)            def rows(self) -> list:
1035: (8)                _sorted = sorted(self.cells, key=itemgetter(1, 0))
1036: (8)                xs = list(sorted(set(map(itemgetter(0), self.cells))))
1037: (8)                rows = []
1038: (8)                for y, row_cells in itertools.groupby(_sorted, itemgetter(1)):
1039: (12)                   xdict = {cell[0]: cell for cell in row_cells}
1040: (12)                   row = TableRow([xdict.get(x) for x in xs])
1041: (12)                   rows.append(row)
1042: (8)                return rows
1043: (4)            @property
1044: (4)            def row_count(self) -> int:  # PyMuPDF extension
1045: (8)                return len(self.rows)
1046: (4)            @property
1047: (4)            def col_count(self) -> int:  # PyMuPDF extension
1048: (8)                return max([len(r.cells) for r in self.rows])
1049: (4)            def extract(self, **kwargs) -> list:
1050: (8)                chars = CHARS
1051: (8)                table_arr = []
1052: (8)                def char_in_bbox(char, bbox) -> bool:
1053: (12)                   v_mid = (char["top"] + char["bottom"]) / 2
1054: (12)                   h_mid = (char["x0"] + char["x1"]) / 2
1055: (12)                   x0, top, x1, bottom = bbox
1056: (12)                   return bool(
1057: (16)                       (h_mid >= x0) and (h_mid < x1) and (v_mid >= top) and (v_mid <
bottom)
1058: (12)                   )
1059: (8)                for row in self.rows:
1060: (12)                   arr = []
1061: (12)                   row_chars = [char for char in chars if char_in_bbox(char,
row.bbox)]
1062: (12)                   for cell in row.cells:
1063: (16)                       if cell is None:
1064: (20)                           cell_text = None
1065: (16)                       else:
1066: (20)                           cell_chars = [
1067: (24)                               char for char in row_chars if char_in_bbox(char, cell)
1068: (20)                           ]
1069: (20)                           if len(cell_chars):
1070: (24)                               kwargs["x_shift"] = cell[0]
1071: (24)                               kwargs["y_shift"] = cell[1]
1072: (24)                               if "layout" in kwargs:
1073: (28)                                   kwargs["layout_width"] = cell[2] - cell[0]
1074: (28)                                   kwargs["layout_height"] = cell[3] - cell[1]
1075: (24)                               cell_text = extract_text(cell_chars, **kwargs)
```

```
(20)                                else:
(24)                                    cell_text = ""
(16)                            arr.append(cell_text)
(12)                        table_arr.append(arr)
(8)                     return table_arr
(4)                 def to_markdown(self, clean=True):
(8)                     """Output table content as a string in Github-markdown format.
(8)                     If clean is true, markdown syntax is removed from cell content."""
(8)                     output = "|"
(8)                     for i, name in enumerate(self.header.names):
(12)                        if name is None or name == "":  # generate a name if empty
(16)                            name = f"Col{i+1}"
(12)                        name = name.replace("\n", " ")  # remove any line breaks
(12)                        if clean:  # remove sensitive syntax
(16)                            name = html.escape(name.replace("-", "&#45;"))
(12)                        output += name + "|"
(8)                     output += "\n"
(8)                     output += "|" + "|".join("---" for i in range(self.col_count)) + "|\n"
(8)                     j = 0 if self.header.external else 1
(8)                     for row in self.extract()[j:]:
(12)                        line = "|"
(12)                        for i, cell in enumerate(row):
(16)                            cell = "" if cell is None else cell.replace("\n", " ")
(16)                            if clean:  # remove sensitive syntax
(20)                                cell = html.escape(cell.replace("-", "&#45;"))
(16)                            line += cell + "|"
(12)                        line += "\n"
(12)                        output += line
(8)                     return output + "\n"
(4)                 def to_pandas(self, **kwargs):
(8)                     """Return a pandas DataFrame version of the table."""
(8)                     try:
(12)                        import pandas as pd
(8)                     except ModuleNotFoundError:
(12)                        message("Package 'pandas' is not installed")
(12)                        raise
(8)                     pd_dict = {}
(8)                     extract = self.extract()
(8)                     hdr = self.header
(8)                     names = self.header.names
(8)                     hdr_len = len(names)
(8)                     for i in range(hdr_len):
(12)                        name = names[i]
(12)                        if not name:
(16)                            names[i] = f"Col{i}"
(8)                     if hdr_len != len(set(names)):
(12)                        for i in range(hdr_len):
(16)                            name = names[i]
(16)                            if name != f"Col{i}":
(20)                                names[i] = f"{i}-{name}"
(8)                     if not hdr.external:  # header is part of 'extract'
(12)                        extract = extract[1:]
(8)                     for i in range(hdr_len):
(12)                        key = names[i]
(12)                        value = []
(12)                        for j in range(len(extract)):
(16)                            value.append(extract[j][i])
(12)                        pd_dict[key] = value
(8)                     return pd.DataFrame(pd_dict)
(4)                 def _get_header(self, y_tolerance=3):
(8)                     """Identify the table header.
(8)                     *** PyMuPDF extension. ***
(8)                     Starting from the first line above the table upwards, check if it
(8)                     qualifies to be part of the table header.
(8)                     Criteria include:
(8)                     * A one-line table never has an extra header.
(8)                     * Column borders must not intersect any word. If this happens, all
(10)                      text of this line and above of it is ignored.
(8)                     * No excess inter-line distance: If a line further up has a distance
```

```
  1145: (10)                    of more than 1.5 times of its font size, it will be ignored and
  1146: (10)                    all lines above of it.
  1147: (8)                   * Must have same text properties.
  1148: (8)                   * Starting with the top table line, a bold text property cannot change
  1149: (10)                    back to non-bold.
  1150: (8)                   If not all criteria are met (or there is no text above the table),
  1151: (8)                   the first table row is assumed to be the header.
  1152: (8)                   """
  1153: (8)                   page = self.page
  1154: (8)                   y_delta = y_tolerance
  1155: (8)                   def top_row_is_bold(bbox):
  1156: (12)                      """Check if row 0 has bold text anywhere.
  1157: (12)                      If this is true, then any non-bold text in lines above disqualify
  1158: (12)                      these lines as header.
  1159: (12)                      bbox is the (potentially repaired) row 0 bbox.
  1160: (12)                      Returns True or False
  1161: (12)                      """
  1162: (12)                      for b in page.get_text("dict", flags=TEXTFLAGS_TEXT, clip=bbox)
  ["blocks"]:
  1163: (16)                          for l in b["lines"]:
  1164: (20)                              for s in l["spans"]:
  1165: (24)                                  if s["flags"] & 16:
  1166: (28)                                      return True
  1167: (12)                      return False
  1168: (8)                   try:
  1169: (12)                      row = self.rows[0]
  1170: (12)                      cells = row.cells
  1171: (12)                      bbox = Rect(row.bbox)
  1172: (8)                   except IndexError:  # this table has no rows
  1173: (12)                      return None
  1174: (8)                   header_top_row = TableHeader(bbox, cells, self.extract()[0], False)
  1175: (8)                   if len(self.rows) < 2:
  1176: (12)                      return header_top_row
  1177: (8)                   if len(cells) < 2:
  1178: (12)                      return header_top_row
  1179: (8)                   col_x = [
  1180: (12)                      c[2] if c is not None else None for c in cells[:-1]
  1181: (8)                   ]  # column (x) coordinates
  1182: (8)                   top_row_bold = top_row_is_bold(bbox)
  1183: (8)                   clip = +bbox  # take row 0 bbox
  1184: (8)                   clip.y0 = 0  # start at top of page
  1185: (8)                   clip.y1 = bbox.y0  # end at top of table
  1186: (8)                   spans = []  # the text spans inside clip
  1187: (8)                   for b in page.get_text("dict", clip=clip, flags=TEXTFLAGS_TEXT)
  ["blocks"]:
  1188: (12)                      for l in b["lines"]:
  1189: (16)                          for s in l["spans"]:
  1190: (20)                              if (
  1191: (24)                                  not s["flags"] & 1 and s["text"].strip()
  1192: (20)                              ):  # ignore superscripts and empty text
  1193: (24)                                  spans.append(s)
  1194: (8)                   select = []  # y1 coordinates above, sorted descending
  1195: (8)                   line_heights = []  # line heights above, sorted descending
  1196: (8)                   line_bolds = []  # bold indicator per line above, same sorting
  1197: (8)                   spans.sort(key=lambda s: s["bbox"][3], reverse=True)
  1198: (8)                   for i in range(len(spans)):
  1199: (12)                      s = spans[i]
  1200: (12)                      y1 = s["bbox"][3]  # span bottom
  1201: (12)                      h = y1 - s["bbox"][1]  # span bbox height
  1202: (12)                      bold = s["flags"] & 16
  1203: (12)                      if i == 0:
  1204: (16)                          select.append(y1)
  1205: (16)                          line_heights.append(h)
  1206: (16)                          line_bolds.append(bold)
  1207: (16)                          continue
  1208: (12)                      y0 = select[-1]
  1209: (12)                      h0 = line_heights[-1]
  1210: (12)                      bold0 = line_bolds[-1]
  1211: (12)                      if bold0 and not bold:
```

```
1212: (16)                          break  # stop if switching from bold to non-bold
1213: (12)                      if y0 - y1 <= y_delta or abs((y0 - h0) - s["bbox"][1]) <= y_delta:
1214: (16)                          s["bbox"] = (s["bbox"][0], y0 - h0, s["bbox"][2], y0)
1215: (16)                          spans[i] = s
1216: (16)                          if bold:
1217: (20)                              line_bolds[-1] = bold
1218: (16)                          continue
1219: (12)                      elif y0 - y1 > 1.5 * h0:
1220: (16)                          break  # stop if distance to previous line too large
1221: (12)                      select.append(y1)
1222: (12)                      line_heights.append(h)
1223: (12)                      line_bolds.append(bold)
1224: (8)                   if select == []:  # nothing above the table?
1225: (12)                      return header_top_row
1226: (8)                   select = select[:5]  # only accept up to 5 lines in any header
1227: (8)                   if bbox.y0 - select[0] >= line_heights[0]:
1228: (12)                      return header_top_row
1229: (8)                   if top_row_bold and not line_bolds[0]:
1230: (12)                      return header_top_row
1231: (8)                   if spans == []:  # nothing left above the table, return top row
1232: (12)                      return header_top_row
1233: (8)                   nclip = EMPTY_RECT()
1234: (8)                   for s in [s for s in spans if s["bbox"][3] >= select[-1]]:
1235: (12)                      nclip |= s["bbox"]
1236: (8)                   if not nclip.is_empty:
1237: (12)                      clip = nclip
1238: (8)                   clip.y1 = bbox.y0  # make sure we still include every word above
1239: (8)                   word_rects = [Rect(w[:4]) for w in page.get_text("words", clip=clip)]
1240: (8)                   word_tops = sorted(list(set([r[1] for r in word_rects])),
reverse=True)
1241: (8)                   select = []
1242: (8)                   for top in word_tops:
1243: (12)                      intersecting = [
1244: (16)                          (x, r)
1245: (16)                          for x in col_x
1246: (16)                          if x is not None
1247: (16)                          for r in word_rects
1248: (16)                          if r[1] == top and r[0] < x and r[2] > x
1249: (12)                      ]
1250: (12)                      if intersecting == []:
1251: (16)                          select.append(top)
1252: (12)                      else:  # detected a word crossing a column border
1253: (16)                          break
1254: (8)                   if select == []:  # nothing left over: return first row
1255: (12)                      return header_top_row
1256: (8)                   hdr_bbox = +clip  # compute the header cells
1257: (8)                   hdr_bbox.y0 = select[-1]  # hdr_bbox top is smallest top coord of
words
1258: (8)                   hdr_cells = [
1259: (12)                      (c[0], hdr_bbox.y0, c[2], hdr_bbox.y1) if c is not None else None
1260: (12)                      for c in cells
1261: (8)                   ]
1262: (8)                   hdr_bbox.x0 = self.bbox[0]
1263: (8)                   hdr_bbox.x1 = self.bbox[2]
1264: (8)                   hdr_names = [
1265: (12)                      (
1266: (16)                          page.get_textbox(c).replace("\n", " ").replace("  ", "
").strip()
1267: (16)                          if c is not None
1268: (16)                          else ""
1269: (12)                      )
1270: (12)                      for c in hdr_cells
1271: (8)                   ]
1272: (8)                   return TableHeader(tuple(hdr_bbox), hdr_cells, hdr_names, True)
1273: (0)           @dataclass
1274: (0)           class TableSettings:
1275: (4)               vertical_strategy: str = "lines"
1276: (4)               horizontal_strategy: str = "lines"
1277: (4)               explicit_vertical_lines: list = None
```

```
    explicit_horizontal_lines: list = None
    snap_tolerance: float = DEFAULT_SNAP_TOLERANCE
    snap_x_tolerance: float = UNSET
    snap_y_tolerance: float = UNSET
    join_tolerance: float = DEFAULT_JOIN_TOLERANCE
    join_x_tolerance: float = UNSET
    join_y_tolerance: float = UNSET
    edge_min_length: float = 3
    min_words_vertical: float = DEFAULT_MIN_WORDS_VERTICAL
    min_words_horizontal: float = DEFAULT_MIN_WORDS_HORIZONTAL
    intersection_tolerance: float = 3
    intersection_x_tolerance: float = UNSET
    intersection_y_tolerance: float = UNSET
    text_settings: dict = None
    def __post_init__(self) -> "TableSettings":
        """Clean up user-provided table settings.
        Validates that the table settings provided consists of acceptable values and
        returns a cleaned up version. The cleaned up version fills out the missing
        values with the default values in the provided settings.
        TODO: Can be further used to validate that the values are of the correct
            type. For example, raising a value error when a non-boolean input is
            provided for the key ``keep_blank_chars``.
        :param table_settings: User-provided table settings.
        :returns: A cleaned up version of the user-provided table settings.
        :raises ValueError: When an unrecognised key is provided.
        """
        for setting in NON_NEGATIVE_SETTINGS:
            if (getattr(self, setting) or 0) < 0:
                raise ValueError(f"Table setting '{setting}' cannot be negative")
        for orientation in ["horizontal", "vertical"]:
            strategy = getattr(self, orientation + "_strategy")
            if strategy not in TABLE_STRATEGIES:
                raise ValueError(
                    f"{orientation}_strategy must be one of"
                    f'{{{",".join(TABLE_STRATEGIES)}}}'
                )
        if self.text_settings is None:
            self.text_settings = {}
        for attr in ["x_tolerance", "y_tolerance"]:
            if attr not in self.text_settings:
                self.text_settings[attr] = self.text_settings.get("tolerance", 3)
        if "tolerance" in self.text_settings:
            del self.text_settings["tolerance"]
        for attr, fallback in [
            ("snap_x_tolerance", "snap_tolerance"),
            ("snap_y_tolerance", "snap_tolerance"),
            ("join_x_tolerance", "join_tolerance"),
            ("join_y_tolerance", "join_tolerance"),
            ("intersection_x_tolerance", "intersection_tolerance"),
            ("intersection_y_tolerance", "intersection_tolerance"),
        ]:
            if getattr(self, attr) is UNSET:
                setattr(self, attr, getattr(self, fallback))
        return self
    @classmethod
    def resolve(cls, settings=None):
        if settings is None:
            return cls()
        elif isinstance(settings, cls):
            return settings
        elif isinstance(settings, dict):
            core_settings = {}
            text_settings = {}
```

```
1341: (12)                      for k, v in settings.items():
1342: (16)                          if k[:5] == "text_":
1343: (20)                              text_settings[k[5:]] = v
1344: (16)                          else:
1345: (20)                              core_settings[k] = v
1346: (12)                      core_settings["text_settings"] = text_settings
1347: (12)                      return cls(**core_settings)
1348: (8)                   else:
1349: (12)                      raise ValueError(f"Cannot resolve settings: {settings}")
1350: (0)           class TableFinder:
1351: (4)               """
1352: (4)               Given a PDF page, find plausible table structures.
1353: (4)               Largely borrowed from Anssi Nurminen's master's thesis:
1354: (4)
http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/21520/Nurminen.pdf?sequence=3
1355: (4)               ... and inspired by Tabula:
1356: (4)               https://github.com/tabulapdf/tabula-extractor/issues/16
1357: (4)               """
1358: (4)               def __init__(self, page, settings=None):
1359: (8)                   self.page = page
1360: (8)                   self.settings = TableSettings.resolve(settings)
1361: (8)                   self.edges = self.get_edges()
1362: (8)                   self.intersections = edges_to_intersections(
1363: (12)                      self.edges,
1364: (12)                      self.settings.intersection_x_tolerance,
1365: (12)                      self.settings.intersection_y_tolerance,
1366: (8)                   )
1367: (8)                   self.cells = intersections_to_cells(self.intersections)
1368: (8)                   self.tables = [
1369: (12)                      Table(self.page, cell_group)
1370: (12)                      for cell_group in cells_to_tables(self.page, self.cells)
1371: (8)                   ]
1372: (4)               def get_edges(self) -> list:
1373: (8)                   settings = self.settings
1374: (8)                   for orientation in ["vertical", "horizontal"]:
1375: (12)                      strategy = getattr(settings, orientation + "_strategy")
1376: (12)                      if strategy == "explicit":
1377: (16)                          lines = getattr(settings, "explicit_" + orientation +
"_lines")
1378: (16)                          if len(lines) < 2:
1379: (20)                              raise ValueError(
1380: (24)                                  f"If {orientation}_strategy == 'explicit', "
1381: (24)                                  f"explicit_{orientation}_lines "
1382: (24)                                  f"must be specified as a list/tuple of two or more "
1383: (24)                                  f"floats/ints."
1384: (20)                              )
1385: (8)                   v_strat = settings.vertical_strategy
1386: (8)                   h_strat = settings.horizontal_strategy
1387: (8)                   if v_strat == "text" or h_strat == "text":
1388: (12)                      words = extract_words(CHARS, **(settings.text_settings or {}))
1389: (8)                   else:
1390: (12)                      words = []
1391: (8)                   v_explicit = []
1392: (8)                   for desc in settings.explicit_vertical_lines or []:
1393: (12)                      if isinstance(desc, dict):
1394: (16)                          for e in obj_to_edges(desc):
1395: (20)                              if e["orientation"] == "v":
1396: (24)                                  v_explicit.append(e)
1397: (12)                      else:
1398: (16)                          v_explicit.append(
1399: (20)                              {
1400: (24)                                  "x0": desc,
1401: (24)                                  "x1": desc,
1402: (24)                                  "top": self.page.rect[1],
1403: (24)                                  "bottom": self.page.rect[3],
1404: (24)                                  "height": self.page.rect[3] - self.page.rect[1],
1405: (24)                                  "orientation": "v",
1406: (20)                              }
1407: (16)                          )
```

```
1408: (8)                    if v_strat == "lines":
1409: (12)                       v_base = filter_edges(EDGES, "v")
1410: (8)                    elif v_strat == "lines_strict":
1411: (12)                       v_base = filter_edges(EDGES, "v", edge_type="line")
1412: (8)                    elif v_strat == "text":
1413: (12)                       v_base = words_to_edges_v(words,
word_threshold=settings.min_words_vertical)
1414: (8)                    elif v_strat == "explicit":
1415: (12)                       v_base = []
1416: (8)                    else:
1417: (12)                       v_base = []
1418: (8)                    v = v_base + v_explicit
1419: (8)                    h_explicit = []
1420: (8)                    for desc in settings.explicit_horizontal_lines or []:
1421: (12)                       if isinstance(desc, dict):
1422: (16)                           for e in obj_to_edges(desc):
1423: (20)                               if e["orientation"] == "h":
1424: (24)                                   h_explicit.append(e)
1425: (12)                       else:
1426: (16)                           h_explicit.append(
1427: (20)                               {
1428: (24)                                   "x0": self.page.rect[0],
1429: (24)                                   "x1": self.page.rect[2],
1430: (24)                                   "width": self.page.rect[2] - self.page.rect[0],
1431: (24)                                   "top": desc,
1432: (24)                                   "bottom": desc,
1433: (24)                                   "orientation": "h",
1434: (20)                               }
1435: (16)                           )
1436: (8)                    if h_strat == "lines":
1437: (12)                       h_base = filter_edges(EDGES, "h")
1438: (8)                    elif h_strat == "lines_strict":
1439: (12)                       h_base = filter_edges(EDGES, "h", edge_type="line")
1440: (8)                    elif h_strat == "text":
1441: (12)                       h_base = words_to_edges_h(
1442: (16)                           words, word_threshold=settings.min_words_horizontal
1443: (12)                       )
1444: (8)                    elif h_strat == "explicit":
1445: (12)                       h_base = []
1446: (8)                    else:
1447: (12)                       h_base = []
1448: (8)                    h = h_base + h_explicit
1449: (8)                    edges = list(v) + list(h)
1450: (8)                    edges = merge_edges(
1451: (12)                       edges,
1452: (12)                       snap_x_tolerance=settings.snap_x_tolerance,
1453: (12)                       snap_y_tolerance=settings.snap_y_tolerance,
1454: (12)                       join_x_tolerance=settings.join_x_tolerance,
1455: (12)                       join_y_tolerance=settings.join_y_tolerance,
1456: (8)                    )
1457: (8)                    return filter_edges(edges, min_length=settings.edge_min_length)
1458: (4)                def __getitem__(self, i):
1459: (8)                    tcount = len(self.tables)
1460: (8)                    if i >= tcount:
1461: (12)                       raise IndexError("table not on page")
1462: (8)                    while i < 0:
1463: (12)                       i += tcount
1464: (8)                    return self.tables[i]
1465: (0)            """
1466: (0)            Start of PyMuPDF interface code.
1467: (0)            The following functions are executed when "page.find_tables()" is called.
1468: (0)            * make_chars: Fills the CHARS list with text character information extracted
1469: (14)                          via "rawdict" text extraction. Items in CHARS are formatted
1470: (14)                          as expected by the table code.
1471: (0)            * make_edges: Fills the EDGES list with vector graphic information extracted
1472: (14)                          via "get_drawings". Items in EDGES are formatted as expected
1473: (14)                          by the table code.
1474: (0)            The lists CHARS and EDGES are used to replace respective document access
1475: (0)            of pdfplumber or, respectively pdfminer.
```

```
1476: (0)           The table code has been modified to use these lists instead of accessing
1477: (0)           page information themselves.
1478: (0)           """
1479: (0)           def make_chars(page, clip=None):
1480: (4)               """Extract text as "rawdict" to fill CHARS."""
1481: (4)               global CHARS, TEXTPAGE
1482: (4)               page_number = page.number + 1
1483: (4)               page_height = page.rect.height
1484: (4)               ctm = page.transformation_matrix
1485: (4)               TEXTPAGE = page.get_textpage(clip=clip, flags=TEXTFLAGS_TEXT)
1486: (4)               blocks = page.get_text("rawdict", textpage=TEXTPAGE)["blocks"]
1487: (4)               doctop_base = page_height * page.number
1488: (4)               for block in blocks:
1489: (8)                   for line in block["lines"]:
1490: (12)                      ldir = line["dir"]  # = (cosine, sine) of angle
1491: (12)                      ldir = (round(ldir[0], 4), round(ldir[1], 4))
1492: (12)                      matrix = Matrix(ldir[0], -ldir[1], ldir[1], ldir[0], 0, 0)
1493: (12)                      if ldir[1] == 0:
1494: (16)                          upright = True
1495: (12)                      else:
1496: (16)                          upright = False
1497: (12)                      for span in sorted(line["spans"], key=lambda s: s["bbox"][0]):
1498: (16)                          fontname = span["font"]
1499: (16)                          fontsize = span["size"]
1500: (16)                          color = sRGB_to_pdf(span["color"])
1501: (16)                          for char in sorted(span["chars"], key=lambda c: c["bbox"][0]):
1502: (20)                              bbox = Rect(char["bbox"])
1503: (20)                              bbox_ctm = bbox * ctm
1504: (20)                              origin = Point(char["origin"]) * ctm
1505: (20)                              matrix.e = origin.x
1506: (20)                              matrix.f = origin.y
1507: (20)                              text = char["c"]
1508: (20)                              char_dict = {
1509: (24)                                  "adv": bbox.x1 - bbox.x0 if upright else bbox.y1 -
bbox.y0,
1510: (24)                                  "bottom": bbox.y1,
1511: (24)                                  "doctop": bbox.y0 + doctop_base,
1512: (24)                                  "fontname": fontname,
1513: (24)                                  "height": bbox.y1 - bbox.y0,
1514: (24)                                  "matrix": tuple(matrix),
1515: (24)                                  "ncs": "DeviceRGB",
1516: (24)                                  "non_stroking_color": color,
1517: (24)                                  "non_stroking_pattern": None,
1518: (24)                                  "object_type": "char",
1519: (24)                                  "page_number": page_number,
1520: (24)                                  "size": fontsize if upright else bbox.y1 - bbox.y0,
1521: (24)                                  "stroking_color": color,
1522: (24)                                  "stroking_pattern": None,
1523: (24)                                  "text": text,
1524: (24)                                  "top": bbox.y0,
1525: (24)                                  "upright": upright,
1526: (24)                                  "width": bbox.x1 - bbox.x0,
1527: (24)                                  "x0": bbox.x0,
1528: (24)                                  "x1": bbox.x1,
1529: (24)                                  "y0": bbox_ctm.y0,
1530: (24)                                  "y1": bbox_ctm.y1,
1531: (20)                              }
1532: (20)                              CHARS.append(char_dict)
1533: (0)           def make_edges(page, clip=None, tset=None, add_lines=None):
1534: (4)               global EDGES
1535: (4)               snap_x = tset.snap_x_tolerance
1536: (4)               snap_y = tset.snap_y_tolerance
1537: (4)               min_length = tset.edge_min_length
1538: (4)               lines_strict = (
1539: (8)                   tset.vertical_strategy == "lines_strict"
1540: (8)                   or tset.horizontal_strategy == "lines_strict"
1541: (4)               )
1542: (4)               page_height = page.rect.height
1543: (4)               doctop_basis = page.number * page_height
```

```
1544: (4)                page_number = page.number + 1
1545: (4)                prect = page.rect
1546: (4)                if page.rotation in (90, 270):
1547: (8)                    w, h = prect.br
1548: (8)                    prect = Rect(0, 0, h, w)
1549: (4)                if clip is not None:
1550: (8)                    clip = Rect(clip)
1551: (4)                else:
1552: (8)                    clip = prect
1553: (4)                def are_neighbors(r1, r2):
1554: (8)                    """Detect whether r1, r2 are neighbors.
1555: (8)                    Defined as:
1556: (8)                    The minimum distance between points of r1 and points of r2 is not
1557: (8)                    larger than some delta.
1558: (8)                    This check supports empty rect-likes and thus also lines.
1559: (8)                    Note:
1560: (8)                    This type of check is MUCH faster than native Rect containment checks.
1561: (8)                    """
1562: (8)                    if (  # check if x-coordinates of r1 are within those of r2
1563: (12)                       r2.x0 - snap_x <= r1.x0 <= r2.x1 + snap_x
1564: (12)                       or r2.x0 - snap_x <= r1.x1 <= r2.x1 + snap_x
1565: (8)                    ) and (  # ... same for y-coordinates
1566: (12)                       r2.y0 - snap_y <= r1.y0 <= r2.y1 + snap_y
1567: (12)                       or r2.y0 - snap_y <= r1.y1 <= r2.y1 + snap_y
1568: (8)                    ):
1569: (12)                       return True
1570: (8)                    if (
1571: (12)                       r1.x0 - snap_x <= r2.x0 <= r1.x1 + snap_x
1572: (12)                       or r1.x0 - snap_x <= r2.x1 <= r1.x1 + snap_x
1573: (8)                    ) and (
1574: (12)                       r1.y0 - snap_y <= r2.y0 <= r1.y1 + snap_y
1575: (12)                       or r1.y0 - snap_y <= r2.y1 <= r1.y1 + snap_y
1576: (8)                    ):
1577: (12)                       return True
1578: (8)                    return False
1579: (4)                def clean_graphics():
1580: (8)                    """Detect and join rectangles of "connected" vector graphics."""
1581: (8)                    paths = []  # paths relevant for table detection
1582: (8)                    for p in page.get_drawings():
1583: (12)                       if (
1584: (16)                           p["type"] == "f"
1585: (16)                           and lines_strict
1586: (16)                           and p["rect"].width > snap_x
1587: (16)                           and p["rect"].height > snap_y
1588: (12)                       ):
1589: (16)                           continue
1590: (12)                       paths.append(p)
1591: (8)                    prects = sorted(set([p["rect"] for p in paths]), key=lambda r: (r.y1,
r.x0))
1592: (8)                    new_rects = []  # the final list of joined rectangles
1593: (8)                    while prects:  # the algorithm will empty this list
1594: (12)                       prect0 = prects[0]  # copy of first rectangle (performance
reasons!)
1595: (12)                       repeat = True
1596: (12)                       while repeat:  # this loop extends first rect in list
1597: (16)                           repeat = False  # set to true again if some other rect touches
1598: (16)                           for i in range(len(prects) - 1, 0, -1):  # run backwards
1599: (20)                               if are_neighbors(prect0, prects[i]):  # close enough to
rect 0?
1600: (24)                                   prect0 |= prects[i].tl  # extend rect 0
1601: (24)                                   prect0 |= prects[i].br  # extend rect 0
1602: (24)                                   del prects[i]  # delete this rect
1603: (24)                                   repeat = True  # keep checking the rest
1604: (12)                       if not white_spaces.issuperset(page.get_textbox(prect0,
textpage=TEXTPAGE)):
1605: (16)                           new_rects.append(prect0)
1606: (12)                       del prects[0]  # remove from rect list
1607: (8)                    return new_rects, paths
1608: (4)                bboxes, paths = clean_graphics()
```

```
(4)    def is_parallel(p1, p2):
(8)        """Check if line is roughly axis-parallel."""
(8)        if abs(p1.x - p2.x) <= snap_x or abs(p1.y - p2.y) <= snap_y:
(12)           return True
(8)        return False
(4)    def make_line(p, p1, p2, clip):
(8)        """Given 2 points, make a line dictionary for table detection."""
(8)        if not is_parallel(p1, p2):  # only accepting axis-parallel lines
(12)           return {}
(8)        x0 = min(p1.x, p2.x)
(8)        x1 = max(p1.x, p2.x)
(8)        y0 = min(p1.y, p2.y)
(8)        y1 = max(p1.y, p2.y)
(8)        if x0 > clip.x1 or x1 < clip.x0 or y0 > clip.y1 or y1 < clip.y0:
(12)           return {}
(8)        if x0 < clip.x0:
(12)           x0 = clip.x0  # adjust to clip boundary
(8)        if x1 > clip.x1:
(12)           x1 = clip.x1  # adjust to clip boundary
(8)        if y0 < clip.y0:
(12)           y0 = clip.y0  # adjust to clip boundary
(8)        if y1 > clip.y1:
(12)           y1 = clip.y1  # adjust to clip boundary
(8)        width = x1 - x0  # from adjusted values
(8)        height = y1 - y0  # from adjusted values
(8)        if width == height == 0:
(12)           return {}  # nothing left to deal with
(8)        line_dict = {
(12)           "x0": x0,
(12)           "y0": page_height - y0,
(12)           "x1": x1,
(12)           "y1": page_height - y1,
(12)           "width": width,
(12)           "height": height,
(12)           "pts": [(x0, y0), (x1, y1)],
(12)           "linewidth": p["width"],
(12)           "stroke": True,
(12)           "fill": False,
(12)           "evenodd": False,
(12)           "stroking_color": p["color"] if p["color"] else p["fill"],
(12)           "non_stroking_color": None,
(12)           "object_type": "line",
(12)           "page_number": page_number,
(12)           "stroking_pattern": None,
(12)           "non_stroking_pattern": None,
(12)           "top": y0,
(12)           "bottom": y1,
(12)           "doctop": y0 + doctop_basis,
(8)        }
(8)        return line_dict
(4)    for p in paths:
(8)        items = p["items"]  # items in this path
(8)        if p["closePath"] and items[0][0] == "l" and items[-1][0] == "l":
(12)           items.append(("l", items[-1][2], items[0][1]))
(8)        for i in items:
(12)           if i[0] not in ("l", "re", "qu"):
(16)               continue  # ignore anything else
(12)           if i[0] == "l":  # a line
(16)               p1, p2 = i[1:]
(16)               line_dict = make_line(p, p1, p2, clip)
(16)               if line_dict:
(20)                   EDGES.append(line_to_edge(line_dict))
(12)           elif i[0] == "re":
(16)               rect = i[1].normalize()  # normalize the rectangle
(16)               if (
(20)                   rect.width <= min_length and rect.width < rect.height
(16)               ):  # simulates a vertical line
(20)                   x = abs(rect.x1 + rect.x0) / 2  # take middle value for x
(20)                   p1 = Point(x, rect.y0)
```

```
(20)                                p2 = Point(x, rect.y1)
(20)                                line_dict = make_line(p, p1, p2, clip)
(20)                                if line_dict:
(24)                                    EDGES.append(line_to_edge(line_dict))
(20)                                continue
(16)                            if (
(20)                                rect.height <= min_length and rect.height < rect.width
(16)                            ):  # simulates a horizontal line
(20)                                y = abs(rect.y1 + rect.y0) / 2  # take middle value for y
(20)                                p1 = Point(rect.x0, y)
(20)                                p2 = Point(rect.x1, y)
(20)                                line_dict = make_line(p, p1, p2, clip)
(20)                                if line_dict:
(24)                                    EDGES.append(line_to_edge(line_dict))
(20)                                continue
(16)                            line_dict = make_line(p, rect.tl, rect.bl, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, rect.bl, rect.br, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, rect.br, rect.tr, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, rect.tr, rect.tl, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(12)                        else:  # must be a quad
(16)                            ul, ur, ll, lr = i[1]
(16)                            line_dict = make_line(p, ul, ll, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, ll, lr, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, lr, ur, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(16)                            line_dict = make_line(p, ur, ul, clip)
(16)                            if line_dict:
(20)                                EDGES.append(line_to_edge(line_dict))
(4)                 path = {"color": (0, 0, 0), "fill": None, "width": 1}
(4)                 for bbox in bboxes:  # add the border lines for all enveloping bboxes
(8)                     line_dict = make_line(path, bbox.tl, bbox.tr, clip)
(8)                     if line_dict:
(12)                        EDGES.append(line_to_edge(line_dict))
(8)                     line_dict = make_line(path, bbox.bl, bbox.br, clip)
(8)                     if line_dict:
(12)                        EDGES.append(line_to_edge(line_dict))
(8)                     line_dict = make_line(path, bbox.tl, bbox.bl, clip)
(8)                     if line_dict:
(12)                        EDGES.append(line_to_edge(line_dict))
(8)                     line_dict = make_line(path, bbox.tr, bbox.br, clip)
(8)                     if line_dict:
(12)                        EDGES.append(line_to_edge(line_dict))
(4)                 if add_lines is not None:  # add user-specified lines
(8)                     assert isinstance(add_lines, (tuple, list))
(4)                 else:
(8)                     add_lines = []
(4)                 for p1, p2 in add_lines:
(8)                     p1 = Point(p1)
(8)                     p2 = Point(p2)
(8)                     line_dict = make_line(path, p1, p2, clip)
(8)                     if line_dict:
(12)                        EDGES.append(line_to_edge(line_dict))
(0)             def page_rotation_set0(page):
(4)                 """Nullify page rotation.
(4)                 To correctly detect tables, page rotation must be zero.
(4)                 This function performs the necessary adjustments and returns information
```

```
1747: (4)               for reverting this changes.
1748: (4)               """
1749: (4)               mediabox = page.mediabox
1750: (4)               rot = page.rotation  # contains normalized rotation value
1751: (4)               mb = page.mediabox  # current mediabox
1752: (4)               if rot == 90:
1753: (8)                   mat0 = Matrix(1, 0, 0, 1, mb.y1 - mb.x1 - mb.x0 - mb.y0, 0)
1754: (4)               elif rot == 270:
1755: (8)                   mat0 = Matrix(1, 0, 0, 1, 0, mb.x1 - mb.y1 - mb.y0 - mb.x0)
1756: (4)               else:
1757: (8)                   mat0 = Matrix(1, 0, 0, 1, -2 * mb.x0, -2 * mb.y0)
1758: (4)               mat = mat0 * page.derotation_matrix
1759: (4)               cmd = b"%g %g %g %g %g %g cm " % tuple(mat)
1760: (4)               xref = TOOLS._insert_contents(page, cmd, 0)
1761: (4)               if rot in (90, 270):
1762: (8)                   x0, y0, x1, y1 = mb
1763: (8)                   mb.x0 = y0
1764: (8)                   mb.y0 = x0
1765: (8)                   mb.x1 = y1
1766: (8)                   mb.y1 = x1
1767: (8)                   page.set_mediabox(mb)
1768: (4)               page.set_rotation(0)
1769: (4)               doc = page.parent
1770: (4)               pno = page.number
1771: (4)               page = doc[pno]
1772: (4)               return page, xref, rot, mediabox
1773: (0)           def page_rotation_reset(page, xref, rot, mediabox):
1774: (4)               """Reset page rotation to original values.
1775: (4)               To be used before we return tables."""
1776: (4)               doc = page.parent  # document of the page
1777: (4)               doc.update_stream(xref, b" ")  # remove de-rotation matrix
1778: (4)               page.set_mediabox(mediabox)  # set mediabox to old value
1779: (4)               page.set_rotation(rot)  # set rotation to old value
1780: (4)               pno = page.number
1781: (4)               page = doc[pno]  # update page info
1782: (4)               return page
1783: (0)           def find_tables(
1784: (4)               page,
1785: (4)               clip=None,
1786: (4)               vertical_strategy: str = "lines",
1787: (4)               horizontal_strategy: str = "lines",
1788: (4)               vertical_lines: list = None,
1789: (4)               horizontal_lines: list = None,
1790: (4)               snap_tolerance: float = DEFAULT_SNAP_TOLERANCE,
1791: (4)               snap_x_tolerance: float = None,
1792: (4)               snap_y_tolerance: float = None,
1793: (4)               join_tolerance: float = DEFAULT_JOIN_TOLERANCE,
1794: (4)               join_x_tolerance: float = None,
1795: (4)               join_y_tolerance: float = None,
1796: (4)               edge_min_length: float = 3,
1797: (4)               min_words_vertical: float = DEFAULT_MIN_WORDS_VERTICAL,
1798: (4)               min_words_horizontal: float = DEFAULT_MIN_WORDS_HORIZONTAL,
1799: (4)               intersection_tolerance: float = 3,
1800: (4)               intersection_x_tolerance: float = None,
1801: (4)               intersection_y_tolerance: float = None,
1802: (4)               text_tolerance=3,
1803: (4)               text_x_tolerance=3,
1804: (4)               text_y_tolerance=3,
1805: (4)               strategy=None,  # offer abbreviation
1806: (4)               add_lines=None,  # optional user-specified lines
1807: (0)           ):
1808: (4)               global CHARS, EDGES
1809: (4)               CHARS = []
1810: (4)               EDGES = []
1811: (4)               old_small = bool(TOOLS.set_small_glyph_heights())  # save old value
1812: (4)               TOOLS.set_small_glyph_heights(True)  # we need minimum bboxes
1813: (4)               if page.rotation != 0:
1814: (8)                   page, old_xref, old_rot, old_mediabox = page_rotation_set0(page)
1815: (4)               else:
```

```
1816: (8)                   old_xref, old_rot, old_mediabox = None, None, None
1817: (4)               if snap_x_tolerance is None:
1818: (8)                   snap_x_tolerance = UNSET
1819: (4)               if snap_y_tolerance is None:
1820: (8)                   snap_y_tolerance = UNSET
1821: (4)               if join_x_tolerance is None:
1822: (8)                   join_x_tolerance = UNSET
1823: (4)               if join_y_tolerance is None:
1824: (8)                   join_y_tolerance = UNSET
1825: (4)               if intersection_x_tolerance is None:
1826: (8)                   intersection_x_tolerance = UNSET
1827: (4)               if intersection_y_tolerance is None:
1828: (8)                   intersection_y_tolerance = UNSET
1829: (4)               if strategy is not None:
1830: (8)                   vertical_strategy = strategy
1831: (8)                   horizontal_strategy = strategy
1832: (4)               settings = {
1833: (8)                   "vertical_strategy": vertical_strategy,
1834: (8)                   "horizontal_strategy": horizontal_strategy,
1835: (8)                   "explicit_vertical_lines": vertical_lines,
1836: (8)                   "explicit_horizontal_lines": horizontal_lines,
1837: (8)                   "snap_tolerance": snap_tolerance,
1838: (8)                   "snap_x_tolerance": snap_x_tolerance,
1839: (8)                   "snap_y_tolerance": snap_y_tolerance,
1840: (8)                   "join_tolerance": join_tolerance,
1841: (8)                   "join_x_tolerance": join_x_tolerance,
1842: (8)                   "join_y_tolerance": join_y_tolerance,
1843: (8)                   "edge_min_length": edge_min_length,
1844: (8)                   "min_words_vertical": min_words_vertical,
1845: (8)                   "min_words_horizontal": min_words_horizontal,
1846: (8)                   "intersection_tolerance": intersection_tolerance,
1847: (8)                   "intersection_x_tolerance": intersection_x_tolerance,
1848: (8)                   "intersection_y_tolerance": intersection_y_tolerance,
1849: (8)                   "text_tolerance": text_tolerance,
1850: (8)                   "text_x_tolerance": text_x_tolerance,
1851: (8)                   "text_y_tolerance": text_y_tolerance,
1852: (4)               }
1853: (4)               tset = TableSettings.resolve(settings=settings)
1854: (4)               page.table_settings = tset
1855: (4)               make_chars(page, clip=clip)  # create character list of page
1856: (4)               make_edges(
1857: (8)                   page, clip=clip, tset=tset, add_lines=add_lines
1858: (4)               )  # create lines and curves
1859: (4)               tables = TableFinder(page, settings=tset)
1860: (4)               TOOLS.set_small_glyph_heights(old_small)
1861: (4)               if old_xref is not None:
1862: (8)                   page = page_rotation_reset(page, old_xref, old_rot, old_mediabox)
1863: (4)               return tables
```

File 5 - utils.py:

```
1: (0)              import io
2: (0)              import math
3: (0)              import os
4: (0)              import typing
5: (0)              import weakref
6: (0)              try:
7: (4)                  from . import pymupdf
8: (0)              except Exception:
9: (4)                  import pymupdf
10: (0)             try:
11: (4)                 from . import mupdf
12: (0)             except Exception:
13: (4)                 import mupdf
14: (0)             _format_g = pymupdf.format_g
15: (0)             g_exceptions_verbose = pymupdf.g_exceptions_verbose
16: (0)             TESSDATA_PREFIX = os.environ.get("TESSDATA_PREFIX")
```

```
 17: (0)             point_like = "point_like"
 18: (0)             rect_like = "rect_like"
 19: (0)             matrix_like = "matrix_like"
 20: (0)             quad_like = "quad_like"
 21: (0)             AnyType = typing.Any
 22: (0)             OptInt = typing.Union[int, None]
 23: (0)             OptFloat = typing.Optional[float]
 24: (0)             OptStr = typing.Optional[str]
 25: (0)             OptDict = typing.Optional[dict]
 26: (0)             OptBytes = typing.Optional[typing.ByteString]
 27: (0)             OptSeq = typing.Optional[typing.Sequence]
 28: (0)             """
 29: (0)             This is a collection of functions to extend PyMupdf.
 30: (0)             """
 31: (0)             def write_text(
 32: (8)                     page: pymupdf.Page,
 33: (8)                     rect=None,
 34: (8)                     writers=None,
 35: (8)                     overlay=True,
 36: (8)                     color=None,
 37: (8)                     opacity=None,
 38: (8)                     keep_proportion=True,
 39: (8)                     rotate=0,
 40: (8)                     oc=0,
 41: (8)                     ) -> None:
 42: (4)                 """Write the text of one or more pymupdf.TextWriter objects.
 43: (4)                 Args:
 44: (8)                     rect: target rectangle. If None, the union of the text writers is
 used.
 45: (8)                     writers: one or more pymupdf.TextWriter objects.
 46: (8)                     overlay: put in foreground or background.
 47: (8)                     keep_proportion: maintain aspect ratio of rectangle sides.
 48: (8)                     rotate: arbitrary rotation angle.
 49: (8)                     oc: the xref of an optional content object
 50: (4)                 """
 51: (4)                 assert isinstance(page, pymupdf.Page)
 52: (4)                 if not writers:
 53: (8)                     raise ValueError("need at least one pymupdf.TextWriter")
 54: (4)                 if type(writers) is pymupdf.TextWriter:
 55: (8)                     if rotate == 0 and rect is None:
 56: (12)                        writers.write_text(page, opacity=opacity, color=color,
 overlay=overlay)
 57: (12)                        return None
 58: (8)                     else:
 59: (12)                        writers = (writers,)
 60: (4)                 clip = writers[0].text_rect
 61: (4)                 textdoc = pymupdf.Document()
 62: (4)                 tpage = textdoc.new_page(width=page.rect.width, height=page.rect.height)
 63: (4)                 for writer in writers:
 64: (8)                     clip |= writer.text_rect
 65: (8)                     writer.write_text(tpage, opacity=opacity, color=color)
 66: (4)                 if rect is None:
 67: (8)                     rect = clip
 68: (4)                 page.show_pdf_page(
 69: (8)                     rect,
 70: (8)                     textdoc,
 71: (8)                     0,
 72: (8)                     overlay=overlay,
 73: (8)                     keep_proportion=keep_proportion,
 74: (8)                     rotate=rotate,
 75: (8)                     clip=clip,
 76: (8)                     oc=oc,
 77: (4)                 )
 78: (4)                 textdoc = None
 79: (4)                 tpage = None
 80: (0)             def show_pdf_page(
 81: (8)                     page,
 82: (8)                     rect,
 83: (8)                     src,
```

```
84: (8)                     pno=0,
85: (8)                     keep_proportion=True,
86: (8)                     overlay=True,
87: (8)                     oc=0,
88: (8)                     rotate=0,
89: (8)                     clip=None,
90: (8)                     ) -> int:
91: (4)                 """Show page number 'pno' of PDF 'src' in rectangle 'rect'.
92: (4)                 Args:
93: (8)                     rect: (rect-like) where to place the source image
94: (8)                     src: (document) source PDF
95: (8)                     pno: (int) source page number
96: (8)                     keep_proportion: (bool) do not change width-height-ratio
97: (8)                     overlay: (bool) put in foreground
98: (8)                     oc: (xref) make visibility dependent on this OCG / OCMD (which must be
defined in the target PDF)
99: (8)                     rotate: (int) degrees (multiple of 90)
100: (8)                    clip: (rect-like) part of source page rectangle
101: (4)                Returns:
102: (8)                    xref of inserted object (for reuse)
103: (4)                """
104: (4)                def calc_matrix(sr, tr, keep=True, rotate=0):
105: (8)                    """Calculate transformation matrix from source to target rect.
106: (8)                    Notes:
107: (12)                       The product of four matrices in this sequence: (1) translate
correct
108: (12)                       source corner to origin, (2) rotate, (3) scale, (4) translate to
109: (12)                       target's top-left corner.
110: (8)                    Args:
111: (12)                       sr: source rect in PDF (!) coordinate system
112: (12)                       tr: target rect in PDF coordinate system
113: (12)                       keep: whether to keep source ratio of width to height
114: (12)                       rotate: rotation angle in degrees
115: (8)                    Returns:
116: (12)                       Transformation matrix.
117: (8)                    """
118: (8)                    smp = (sr.tl + sr.br) / 2.0
119: (8)                    tmp = (tr.tl + tr.br) / 2.0
120: (8)                    m = pymupdf.Matrix(1, 0, 0, 1, -smp.x, -smp.y) *
pymupdf.Matrix(rotate)
121: (8)                    sr1 = sr * m  # resulting source rect to calculate scale factors
122: (8)                    fw = tr.width / sr1.width  # scale the width
123: (8)                    fh = tr.height / sr1.height  # scale the height
124: (8)                    if keep:
125: (12)                       fw = fh = min(fw, fh)  # take min if keeping aspect ratio
126: (8)                    m *= pymupdf.Matrix(fw, fh)  # concat scale matrix
127: (8)                    m *= pymupdf.Matrix(1, 0, 0, 1, tmp.x, tmp.y)  # concat move to target
center
128: (8)                    return pymupdf.JM_TUPLE(m)
129: (4)                pymupdf.CheckParent(page)
130: (4)                doc = page.parent
131: (4)                if not doc.is_pdf or not src.is_pdf:
132: (8)                    raise ValueError("is no PDF")
133: (4)                if rect.is_empty or rect.is_infinite:
134: (8)                    raise ValueError("rect must be finite and not empty")
135: (4)                while pno < 0:  # support negative page numbers
136: (8)                    pno += src.page_count
137: (4)                src_page = src[pno]  # load source page
138: (4)                if src_page.get_contents() == []:
139: (8)                    raise ValueError("nothing to show - source page empty")
140: (4)                tar_rect = rect * ~page.transformation_matrix  # target rect in PDF
coordinates
141: (4)                src_rect = src_page.rect if not clip else src_page.rect & clip  # source
rect
142: (4)                if src_rect.is_empty or src_rect.is_infinite:
143: (8)                    raise ValueError("clip must be finite and not empty")
144: (4)                src_rect = src_rect * ~src_page.transformation_matrix  # ... in PDF coord
145: (4)                matrix = calc_matrix(src_rect, tar_rect, keep=keep_proportion,
rotate=rotate)
```

```
146: (4)                ilst = [i[1] for i in doc.get_page_xobjects(page.number)]
147: (4)                ilst += [i[7] for i in doc.get_page_images(page.number)]
148: (4)                ilst += [i[4] for i in doc.get_page_fonts(page.number)]
149: (4)                n = "fzFrm"
150: (4)                i = 0
151: (4)                _imgname = n + "0"
152: (4)                while _imgname in ilst:
153: (8)                    i += 1
154: (8)                    _imgname = n + str(i)
155: (4)                isrc = src._graft_id  # used as key for graftmaps
156: (4)                if doc._graft_id == isrc:
157: (8)                    raise ValueError("source document must not equal target")
158: (4)                gmap = doc.Graftmaps.get(isrc, None)
159: (4)                if gmap is None:
160: (8)                    gmap = pymupdf.Graftmap(doc)
161: (8)                    doc.Graftmaps[isrc] = gmap
162: (4)                pno_id = (isrc, pno)  # id of src[pno]
163: (4)                xref = doc.ShownPages.get(pno_id, 0)
164: (4)                if overlay:
165: (8)                    page.wrap_contents()  # ensure a balanced graphics state
166: (4)                xref = page._show_pdf_page(
167: (8)                    src_page,
168: (8)                    overlay=overlay,
169: (8)                    matrix=matrix,
170: (8)                    xref=xref,
171: (8)                    oc=oc,
172: (8)                    clip=src_rect,
173: (8)                    graftmap=gmap,
174: (8)                    _imgname=_imgname,
175: (4)                )
176: (4)                doc.ShownPages[pno_id] = xref
177: (4)                return xref
178: (0)            def replace_image(page: pymupdf.Page, xref: int, *, filename=None,
pixmap=None, stream=None):
179: (4)                """Replace the image referred to by xref.
180: (4)                Replace the image by changing the object definition stored under xref.
This
181: (4)                will leave the pages appearance instructions intact, so the new image is
182: (4)                being displayed with the same bbox, rotation etc.
183: (4)                By providing a small fully transparent image, an effect as if the image
had
184: (4)                been deleted can be achieved.
185: (4)                A typical use may include replacing large images by a smaller version,
186: (4)                e.g. with a lower resolution or graylevel instead of colored.
187: (4)                Args:
188: (8)                    xref: the xref of the image to replace.
189: (8)                    filename, pixmap, stream: exactly one of these must be provided. The
190: (12)                       meaning being the same as in Page.insert_image.
191: (4)                """
192: (4)                doc = page.parent  # the owning document
193: (4)                if not doc.xref_is_image(xref):
194: (8)                    raise ValueError("xref not an image")  # insert new image anywhere in
page
195: (4)                if bool(filename) + bool(stream) + bool(pixmap) != 1:
196: (8)                    raise ValueError("Exactly one of filename/stream/pixmap must be
given")
197: (4)                new_xref = page.insert_image(
198: (8)                    page.rect, filename=filename, stream=stream, pixmap=pixmap
199: (4)                )
200: (4)                doc.xref_copy(new_xref, xref)  # copy over new to old
201: (4)                last_contents_xref = page.get_contents()[-1]
202: (4)                doc.update_stream(last_contents_xref, b" ")
203: (0)            def delete_image(page: pymupdf.Page, xref: int):
204: (4)                """Delete the image referred to by xef.
205: (4)                Actually replaces by a small transparent Pixmap using method
Page.replace_image.
206: (4)                Args:
207: (8)                    xref: xref of the image to delete.
208: (4)                """
```

```
209: (4)                pix = pymupdf.Pixmap(pymupdf.csGRAY, (0, 0, 1, 1), 1)
210: (4)                pix.clear_with()  # clear all samples bytes to 0x00
211: (4)                page.replace_image(xref, pixmap=pix)
212: (0)            def insert_image(
213: (8)                    page,
214: (8)                    rect,
215: (8)                    *,
216: (8)                    alpha=-1,
217: (8)                    filename=None,
218: (8)                    height=0,
219: (8)                    keep_proportion=True,
220: (8)                    mask=None,
221: (8)                    oc=0,
222: (8)                    overlay=True,
223: (8)                    pixmap=None,
224: (8)                    rotate=0,
225: (8)                    stream=None,
226: (8)                    width=0,
227: (8)                    xref=0,
228: (8)                    ):
229: (4)                """Insert an image for display in a rectangle.
230: (4)                Args:
231: (8)                    rect: (rect_like) position of image on the page.
232: (8)                    alpha: (int, optional) set to 0 if image has no transparency.
233: (8)                    filename: (str, Path, file object) image filename.
234: (8)                    height: (int)
235: (8)                    keep_proportion: (bool) keep width / height ratio (default).
236: (8)                    mask: (bytes, optional) image consisting of alpha values to use.
237: (8)                    oc: (int) xref of OCG or OCMD to declare as Optional Content.
238: (8)                    overlay: (bool) put in foreground (default) or background.
239: (8)                    pixmap: (pymupdf.Pixmap) use this as image.
240: (8)                    rotate: (int) rotate by 0, 90, 180 or 270 degrees.
241: (8)                    stream: (bytes) use this as image.
242: (8)                    width: (int)
243: (8)                    xref: (int) use this as image.
244: (4)                'page' and 'rect' are positional, all other parameters are keywords.
245: (4)                If 'xref' is given, that image is used. Other input options are ignored.
246: (4)                Else, exactly one of pixmap, stream or filename must be given.
247: (4)                'alpha=0' for non-transparent images improves performance significantly.
248: (4)                Affects stream and filename only.
249: (4)                Optimum transparent insertions are possible by using filename / stream in
250: (4)                conjunction with a 'mask' image of alpha values.
251: (4)                Returns:
252: (8)                    xref (int) of inserted image. Re-use as argument for multiple
insertions.
253: (4)                """
254: (4)                pymupdf.CheckParent(page)
255: (4)                doc = page.parent
256: (4)                if not doc.is_pdf:
257: (8)                    raise ValueError("is no PDF")
258: (4)                if xref == 0 and (bool(filename) + bool(stream) + bool(pixmap) != 1):
259: (8)                    raise ValueError("xref=0 needs exactly one of filename, pixmap,
stream")
260: (4)                if filename:
261: (8)                    if type(filename) is str:
262: (12)                       pass
263: (8)                    elif hasattr(filename, "absolute"):
264: (12)                       filename = str(filename)
265: (8)                    elif hasattr(filename, "name"):
266: (12)                       filename = filename.name
267: (8)                    else:
268: (12)                       raise ValueError("bad filename")
269: (4)                if filename and not os.path.exists(filename):
270: (8)                    raise FileNotFoundError("No such file: '%s'" % filename)
271: (4)                elif stream and type(stream) not in (bytes, bytearray, io.BytesIO):
272: (8)                    raise ValueError("stream must be bytes-like / BytesIO")
273: (4)                elif pixmap and type(pixmap) is not pymupdf.Pixmap:
274: (8)                    raise ValueError("pixmap must be a pymupdf.Pixmap")
275: (4)                if mask and not (stream or filename):
```

```
(8)                 raise ValueError("mask requires stream or filename")
(4)             if mask and type(mask) not in (bytes, bytearray, io.BytesIO):
(8)                 raise ValueError("mask must be bytes-like / BytesIO")
(4)             while rotate < 0:
(8)                 rotate += 360
(4)             while rotate >= 360:
(8)                 rotate -= 360
(4)             if rotate not in (0, 90, 180, 270):
(8)                 raise ValueError("bad rotate value")
(4)             r = pymupdf.Rect(rect)
(4)             if r.is_empty or r.is_infinite:
(8)                 raise ValueError("rect must be finite and not empty")
(4)             clip = r * ~page.transformation_matrix
(4)             ilst = [i[7] for i in doc.get_page_images(page.number)]
(4)             ilst += [i[1] for i in doc.get_page_xobjects(page.number)]
(4)             ilst += [i[4] for i in doc.get_page_fonts(page.number)]
(4)             n = "fzImg"  # 'pymupdf image'
(4)             i = 0
(4)             _imgname = n + "0"  # first name candidate
(4)             while _imgname in ilst:
(8)                 i += 1
(8)                 _imgname = n + str(i)  # try new name
(4)             if overlay:
(8)                 page.wrap_contents()  # ensure a balanced graphics state
(4)             digests = doc.InsertedImages
(4)             xref, digests = page._insert_image(
(8)                 filename=filename,
(8)                 pixmap=pixmap,
(8)                 stream=stream,
(8)                 imask=mask,
(8)                 clip=clip,
(8)                 overlay=overlay,
(8)                 oc=oc,
(8)                 xref=xref,
(8)                 rotate=rotate,
(8)                 keep_proportion=keep_proportion,
(8)                 width=width,
(8)                 height=height,
(8)                 alpha=alpha,
(8)                 _imgname=_imgname,
(8)                 digests=digests,
(4)             )
(4)             if digests is not None:
(8)                 doc.InsertedImages = digests
(4)             return xref
(0)         def search_for(
(8)                 page,
(8)                 text,
(8)                 *,
(8)                 clip=None,
(8)                 quads=False,
(8)                 flags=pymupdf.TEXT_DEHYPHENATE
(12)                    | pymupdf.TEXT_PRESERVE_WHITESPACE
(12)                    | pymupdf.TEXT_PRESERVE_LIGATURES
(12)                    | pymupdf.TEXT_MEDIABOX_CLIP
(12)                    ,
(8)                 textpage=None,
(8)                 ) -> list:
(4)             """Search for a string on a page.
(4)             Args:
(8)                 text: string to be searched for
(8)                 clip: restrict search to this rectangle
(8)                 quads: (bool) return quads instead of rectangles
(8)                 flags: bit switches, default: join hyphened words
(8)                 textpage: a pre-created pymupdf.TextPage
(4)             Returns:
(8)                 a list of rectangles or quads, each containing one occurrence.
(4)             """
(4)             if clip is not None:
```

```
345: (8)                    clip = pymupdf.Rect(clip)
346: (4)                pymupdf.CheckParent(page)
347: (4)                tp = textpage
348: (4)                if tp is None:
349: (8)                    tp = page.get_textpage(clip=clip, flags=flags)  # create
pymupdf.TextPage
350: (4)                elif getattr(tp, "parent") != page:
351: (8)                    raise ValueError("not a textpage of this page")
352: (4)                rlist = tp.search(text, quads=quads)
353: (4)                if textpage is None:
354: (8)                    del tp
355: (4)                return rlist
356: (0)            def search_page_for(
357: (4)                doc: pymupdf.Document,
358: (4)                pno: int,
359: (4)                text: str,
360: (4)                quads: bool = False,
361: (4)                clip: rect_like = None,
362: (4)                flags: int = pymupdf.TEXT_DEHYPHENATE
363: (12)                       | pymupdf.TEXT_PRESERVE_LIGATURES
364: (12)                       | pymupdf.TEXT_PRESERVE_WHITESPACE
365: (12)                       | pymupdf.TEXT_MEDIABOX_CLIP
366: (12)                       ,
367: (4)                textpage: pymupdf.TextPage = None,
368: (0)            ) -> list:
369: (4)                """Search for a string on a page.
370: (4)                Args:
371: (8)                    pno: page number
372: (8)                    text: string to be searched for
373: (8)                    clip: restrict search to this rectangle
374: (8)                    quads: (bool) return quads instead of rectangles
375: (8)                    flags: bit switches, default: join hyphened words
376: (8)                    textpage: reuse a prepared textpage
377: (4)                Returns:
378: (8)                    a list of rectangles or quads, each containing an occurrence.
379: (4)                """
380: (4)                return doc[pno].search_for(
381: (8)                    text,
382: (8)                    quads=quads,
383: (8)                    clip=clip,
384: (8)                    flags=flags,
385: (8)                    textpage=textpage,
386: (4)                )
387: (0)            def get_text_blocks(
388: (4)                page: pymupdf.Page,
389: (4)                clip: rect_like = None,
390: (4)                flags: OptInt = None,
391: (4)                textpage: pymupdf.TextPage = None,
392: (4)                sort: bool = False,
393: (0)            ) -> list:
394: (4)                """Return the text blocks on a page.
395: (4)                Notes:
396: (8)                    Lines in a block are concatenated with line breaks.
397: (4)                Args:
398: (8)                    flags: (int) control the amount of data parsed into the textpage.
399: (4)                Returns:
400: (8)                    A list of the blocks. Each item contains the containing rectangle
401: (8)                    coordinates, text lines, running block number and block type.
402: (4)                """
403: (4)                pymupdf.CheckParent(page)
404: (4)                if flags is None:
405: (8)                    flags = pymupdf.TEXTFLAGS_BLOCKS
406: (4)                tp = textpage
407: (4)                if tp is None:
408: (8)                    tp = page.get_textpage(clip=clip, flags=flags)
409: (4)                elif getattr(tp, "parent") != page:
410: (8)                    raise ValueError("not a textpage of this page")
411: (4)                blocks = tp.extractBLOCKS()
412: (4)                if textpage is None:
```

```
413: (8)                    del tp
414: (4)                if sort is True:
415: (8)                    blocks.sort(key=lambda b: (b[3], b[0]))
416: (4)                return blocks
417: (0)            def get_text_words(
418: (4)                page: pymupdf.Page,
419: (4)                clip: rect_like = None,
420: (4)                flags: OptInt = None,
421: (4)                textpage: pymupdf.TextPage = None,
422: (4)                sort: bool = False,
423: (4)                delimiters=None,
424: (0)            ) -> list:
425: (4)                """Return the text words as a list with the bbox for each word.
426: (4)                Args:
427: (8)                    flags: (int) control the amount of data parsed into the textpage.
428: (8)                    delimiters: (str,list) characters to use as word delimiters
429: (4)                Returns:
430: (8)                    Word tuples (x0, y0, x1, y1, "word", bno, lno, wno).
431: (4)                """
432: (4)                pymupdf.CheckParent(page)
433: (4)                if flags is None:
434: (8)                    flags = pymupdf.TEXT_PRESERVE_WHITESPACE |
pymupdf.TEXT_PRESERVE_LIGATURES | pymupdf.TEXT_MEDIABOX_CLIP
435: (4)                tp = textpage
436: (4)                if tp is None:
437: (8)                    tp = page.get_textpage(clip=clip, flags=flags)
438: (4)                elif getattr(tp, "parent") != page:
439: (8)                    raise ValueError("not a textpage of this page")
440: (4)                words = tp.extractWORDS(delimiters)
441: (4)                if textpage is None:
442: (8)                    del tp
443: (4)                if sort is True:
444: (8)                    words.sort(key=lambda w: (w[3], w[0]))
445: (4)                return words
446: (0)            def get_textbox(
447: (4)                page: pymupdf.Page,
448: (4)                rect: rect_like,
449: (4)                textpage: pymupdf.TextPage = None,
450: (0)            ) -> str:
451: (4)                tp = textpage
452: (4)                if tp is None:
453: (8)                    tp = page.get_textpage()
454: (4)                elif getattr(tp, "parent") != page:
455: (8)                    raise ValueError("not a textpage of this page")
456: (4)                rc = tp.extractTextbox(rect)
457: (4)                if textpage is None:
458: (8)                    del tp
459: (4)                return rc
460: (0)            def get_text_selection(
461: (4)                page: pymupdf.Page,
462: (4)                p1: point_like,
463: (4)                p2: point_like,
464: (4)                clip: rect_like = None,
465: (4)                textpage: pymupdf.TextPage = None,
466: (0)            ):
467: (4)                pymupdf.CheckParent(page)
468: (4)                tp = textpage
469: (4)                if tp is None:
470: (8)                    tp = page.get_textpage(clip=clip, flags=pymupdf.TEXT_DEHYPHENATE)
471: (4)                elif getattr(tp, "parent") != page:
472: (8)                    raise ValueError("not a textpage of this page")
473: (4)                rc = tp.extractSelection(p1, p2)
474: (4)                if textpage is None:
475: (8)                    del tp
476: (4)                return rc
477: (0)            def get_textpage_ocr(
478: (4)                page: pymupdf.Page,
479: (4)                flags: int = 0,
480: (4)                language: str = "eng",
```

```
481: (4)                dpi: int = 72,
482: (4)                full: bool = False,
483: (4)                tessdata: str = None,
484: (0)            ) -> pymupdf.TextPage:
485: (4)                """Create a Textpage from combined results of normal and OCR text parsing.
486: (4)                Args:
487: (8)                    flags: (int) control content becoming part of the result.
488: (8)                    language: (str) specify expected language(s). Default is "eng"
(English).
489: (8)                    dpi: (int) resolution in dpi, default 72.
490: (8)                    full: (bool) whether to OCR the full page image, or only its images
(default)
491: (4)                """
492: (4)                pymupdf.CheckParent(page)
493: (4)                if not TESSDATA_PREFIX and not tessdata:
494: (8)                    raise RuntimeError("No OCR support: TESSDATA_PREFIX not set")
495: (4)                def full_ocr(page, dpi, language, flags):
496: (8)                    zoom = dpi / 72
497: (8)                    mat = pymupdf.Matrix(zoom, zoom)
498: (8)                    pix = page.get_pixmap(matrix=mat)
499: (8)                    ocr_pdf = pymupdf.Document(
500: (16)                           "pdf",
501: (16)                           pix.pdfocr_tobytes(
502: (20)                               compress=False,
503: (20)                               language=language,
504: (20)                               tessdata=tessdata,
505: (20)                               ),
506: (16)                           )
507: (8)                    ocr_page = ocr_pdf.load_page(0)
508: (8)                    unzoom = page.rect.width / ocr_page.rect.width
509: (8)                    ctm = pymupdf.Matrix(unzoom, unzoom) * page.derotation_matrix
510: (8)                    tpage = ocr_page.get_textpage(flags=flags, matrix=ctm)
511: (8)                    ocr_pdf.close()
512: (8)                    pix = None
513: (8)                    tpage.parent = weakref.proxy(page)
514: (8)                    return tpage
515: (4)                if full is True:
516: (8)                    return full_ocr(page, dpi, language, flags)
517: (4)                tpage = page.get_textpage(flags=flags)
518: (4)                for block in page.get_text("dict", flags=pymupdf.TEXT_PRESERVE_IMAGES)
["blocks"]:
519: (8)                    if block["type"] != 1:  # only look at images
520: (12)                       continue
521: (8)                    bbox = pymupdf.Rect(block["bbox"])
522: (8)                    if bbox.width <= 3 or bbox.height <= 3:  # ignore tiny stuff
523: (12)                       continue
524: (8)                    exception_types = (RuntimeError, mupdf.FzErrorBase)
525: (8)                    if pymupdf.mupdf_version_tuple < (1, 24):
526: (12)                       exception_types = RuntimeError
527: (8)                    try:
528: (12)                       pix = pymupdf.Pixmap(block["image"])  # get image pixmap
529: (12)                       if pix.n - pix.alpha != 3:  # we need to convert this to RGB!
530: (16)                           pix = pymupdf.Pixmap(pymupdf.csRGB, pix)
531: (12)                       if pix.alpha:  # must remove alpha channel
532: (16)                           pix = pymupdf.Pixmap(pix, 0)
533: (12)                       imgdoc = pymupdf.Document(
534: (20)                               "pdf",
535: (20)                               pix.pdfocr_tobytes(language=language, tessdata=tessdata),
536: (20)                               )  # pdf with OCRed page
537: (12)                       imgpage = imgdoc.load_page(0)  # read image as a page
538: (12)                       pix = None
539: (12)                       imgrect = imgpage.rect  # page size of image PDF
540: (12)                       shrink = pymupdf.Matrix(1 / imgrect.width, 1 / imgrect.height)
541: (12)                       mat = shrink * block["transform"]
542: (12)                       imgpage.extend_textpage(tpage, flags=0, matrix=mat)
543: (12)                       imgdoc.close()
544: (8)                    except exception_types:
545: (12)                       if g_exceptions_verbose:    pymupdf.exception_info()
546: (12)                       tpage = None
```

```
547: (12)                        pymupdf.message("Falling back to full page OCR")
548: (12)                        return full_ocr(page, dpi, language, flags)
549: (4)                return tpage
550: (0)            def get_image_info(page: pymupdf.Page, hashes: bool = False, xrefs: bool =
False) -> list:
551: (4)                """Extract image information only from a pymupdf.TextPage.
552: (4)                Args:
553: (8)                    hashes: (bool) include MD5 hash for each image.
554: (8)                    xrefs: (bool) try to find the xref for each image. Sets hashes to
true.
555: (4)                """
556: (4)                doc = page.parent
557: (4)                if xrefs and doc.is_pdf:
558: (8)                    hashes = True
559: (4)                if not doc.is_pdf:
560: (8)                    xrefs = False
561: (4)                imginfo = getattr(page, "_image_info", None)
562: (4)                if imginfo and not xrefs:
563: (8)                    return imginfo
564: (4)                if not imginfo:
565: (8)                    tp = page.get_textpage(flags=pymupdf.TEXT_PRESERVE_IMAGES)
566: (8)                    imginfo = tp.extractIMGINFO(hashes=hashes)
567: (8)                    del tp
568: (8)                    if hashes:
569: (12)                       page._image_info = imginfo
570: (4)                if not xrefs or not doc.is_pdf:
571: (8)                    return imginfo
572: (4)                imglist = page.get_images()
573: (4)                digests = {}
574: (4)                for item in imglist:
575: (8)                    xref = item[0]
576: (8)                    pix = pymupdf.Pixmap(doc, xref)
577: (8)                    digests[pix.digest] = xref
578: (8)                    del pix
579: (4)                for i in range(len(imginfo)):
580: (8)                    item = imginfo[i]
581: (8)                    xref = digests.get(item["digest"], 0)
582: (8)                    item["xref"] = xref
583: (8)                    imginfo[i] = item
584: (4)                return imginfo
585: (0)            def get_image_rects(page: pymupdf.Page, name, transform=False) -> list:
586: (4)                """Return list of image positions on a page.
587: (4)                Args:
588: (8)                    name: (str, list, int) image identification. May be reference name, an
589: (14)                         item of the page's image list or an xref.
590: (8)                    transform: (bool) whether to also return the transformation matrix.
591: (4)                Returns:
592: (8)                    A list of pymupdf.Rect objects or tuples of (pymupdf.Rect,
pymupdf.Matrix)
593: (8)                    for all image locations on the page.
594: (4)                """
595: (4)                if type(name) in (list, tuple):
596: (8)                    xref = name[0]
597: (4)                elif type(name) is int:
598: (8)                    xref = name
599: (4)                else:
600: (8)                    imglist = [i for i in page.get_images() if i[7] == name]
601: (8)                    if imglist == []:
602: (12)                       raise ValueError("bad image name")
603: (8)                    elif len(imglist) != 1:
604: (12)                       raise ValueError("multiple image names found")
605: (8)                    xref = imglist[0][0]
606: (4)                pix = pymupdf.Pixmap(page.parent, xref)  # make pixmap of the image to
compute MD5
607: (4)                digest = pix.digest
608: (4)                del pix
609: (4)                infos = page.get_image_info(hashes=True)
610: (4)                if not transform:
611: (8)                    bboxes = [pymupdf.Rect(im["bbox"]) for im in infos if im["digest"] ==
```

```
digest]
612: (4)                else:
613: (8)                    bboxes = [
614: (12)                       (pymupdf.Rect(im["bbox"]), pymupdf.Matrix(im["transform"]))
615: (12)                       for im in infos
616: (12)                       if im["digest"] == digest
617: (8)                    ]
618: (4)                return bboxes
619: (0)            def get_text(
620: (8)                    page: pymupdf.Page,
621: (8)                    option: str = "text",
622: (8)                    clip: rect_like = None,
623: (8)                    flags: OptInt = None,
624: (8)                    textpage: pymupdf.TextPage = None,
625: (8)                    sort: bool = False,
626: (8)                    delimiters=None,
627: (8)                    ):
628: (4)                """Extract text from a page or an annotation.
629: (4)                This is a unifying wrapper for various methods of the pymupdf.TextPage
class.
630: (4)                Args:
631: (8)                    option: (str) text, words, blocks, html, dict, json, rawdict, xhtml or
xml.
632: (8)                    clip: (rect-like) restrict output to this area.
633: (8)                    flags: bit switches to e.g. exclude images or decompose ligatures.
634: (8)                    textpage: reuse this pymupdf.TextPage and make no new one. If
specified,
635: (12)                       'flags' and 'clip' are ignored.
636: (4)                Returns:
637: (8)                    the output of methods get_text_words / get_text_blocks or
pymupdf.TextPage
638: (8)                    methods extractText, extractHTML, extractDICT, extractJSON,
extractRAWDICT,
639: (8)                    extractXHTML or etractXML respectively.
640: (8)                    Default and misspelling choice is "text".
641: (4)                """
642: (4)                formats = {
643: (8)                    "text": pymupdf.TEXTFLAGS_TEXT,
644: (8)                    "html": pymupdf.TEXTFLAGS_HTML,
645: (8)                    "json": pymupdf.TEXTFLAGS_DICT,
646: (8)                    "rawjson": pymupdf.TEXTFLAGS_RAWDICT,
647: (8)                    "xml": pymupdf.TEXTFLAGS_XML,
648: (8)                    "xhtml": pymupdf.TEXTFLAGS_XHTML,
649: (8)                    "dict": pymupdf.TEXTFLAGS_DICT,
650: (8)                    "rawdict": pymupdf.TEXTFLAGS_RAWDICT,
651: (8)                    "words": pymupdf.TEXTFLAGS_WORDS,
652: (8)                    "blocks": pymupdf.TEXTFLAGS_BLOCKS,
653: (4)                }
654: (4)                option = option.lower()
655: (4)                if option not in formats:
656: (8)                    option = "text"
657: (4)                if flags is None:
658: (8)                    flags = formats[option]
659: (4)                if option == "words":
660: (8)                    return get_text_words(
661: (12)                       page,
662: (12)                       clip=clip,
663: (12)                       flags=flags,
664: (12)                       textpage=textpage,
665: (12)                       sort=sort,
666: (12)                       delimiters=delimiters,
667: (8)                    )
668: (4)                if option == "blocks":
669: (8)                    return get_text_blocks(
670: (12)                       page, clip=clip, flags=flags, textpage=textpage, sort=sort
671: (8)                    )
672: (4)                pymupdf.CheckParent(page)
673: (4)                cb = None
674: (4)                if option in ("html", "xml", "xhtml"):  # no clipping for MuPDF functions
```

```
675: (8)                    clip = page.cropbox
676: (4)                if clip is not None:
677: (8)                    clip = pymupdf.Rect(clip)
678: (8)                    cb = None
679: (4)                elif type(page) is pymupdf.Page:
680: (8)                    cb = page.cropbox
681: (4)                tp = textpage
682: (4)                if tp is None:
683: (8)                    tp = page.get_textpage(clip=clip, flags=flags)
684: (4)                elif getattr(tp, "parent") != page:
685: (8)                    raise ValueError("not a textpage of this page")
686: (4)                if option == "json":
687: (8)                    t = tp.extractJSON(cb=cb, sort=sort)
688: (4)                elif option == "rawjson":
689: (8)                    t = tp.extractRAWJSON(cb=cb, sort=sort)
690: (4)                elif option == "dict":
691: (8)                    t = tp.extractDICT(cb=cb, sort=sort)
692: (4)                elif option == "rawdict":
693: (8)                    t = tp.extractRAWDICT(cb=cb, sort=sort)
694: (4)                elif option == "html":
695: (8)                    t = tp.extractHTML()
696: (4)                elif option == "xml":
697: (8)                    t = tp.extractXML()
698: (4)                elif option == "xhtml":
699: (8)                    t = tp.extractXHTML()
700: (4)                else:
701: (8)                    t = tp.extractText(sort=sort)
702: (4)                if textpage is None:
703: (8)                    del tp
704: (4)                return t
705: (0)            def get_page_text(
706: (4)                doc: pymupdf.Document,
707: (4)                pno: int,
708: (4)                option: str = "text",
709: (4)                clip: rect_like = None,
710: (4)                flags: OptInt = None,
711: (4)                textpage: pymupdf.TextPage = None,
712: (4)                sort: bool = False,
713: (0)            ) -> typing.Any:
714: (4)                """Extract a document page's text by page number.
715: (4)                Notes:
716: (8)                    Convenience function calling page.get_text().
717: (4)                Args:
718: (8)                    pno: page number
719: (8)                    option: (str) text, words, blocks, html, dict, json, rawdict, xhtml or
xml.
720: (4)                Returns:
721: (8)                    output from page.TextPage().
722: (4)                """
723: (4)                return doc[pno].get_text(option, clip=clip, flags=flags, sort=sort)
724: (0)            def get_pixmap(
725: (8)                    page: pymupdf.Page,
726: (8)                    *,
727: (8)                    matrix: matrix_like=pymupdf.Identity,
728: (8)                    dpi=None,
729: (8)                    colorspace: pymupdf.Colorspace=pymupdf.csRGB,
730: (8)                    clip: rect_like=None,
731: (8)                    alpha: bool=False,
732: (8)                    annots: bool=True,
733: (8)                    ) -> pymupdf.Pixmap:
734: (4)                """Create pixmap of page.
735: (4)                Keyword args:
736: (8)                    matrix: Matrix for transformation (default: Identity).
737: (8)                    dpi: desired dots per inch. If given, matrix is ignored.
738: (8)                    colorspace: (str/Colorspace) cmyk, rgb, gray - case ignored, default
csRGB.
739: (8)                    clip: (irect-like) restrict rendering to this area.
740: (8)                    alpha: (bool) whether to include alpha channel
741: (8)                    annots: (bool) whether to also render annotations
```

```
742: (4)                """
743: (4)                if dpi:
744: (8)                    zoom = dpi / 72
745: (8)                    matrix = pymupdf.Matrix(zoom, zoom)
746: (4)                if type(colorspace) is str:
747: (8)                    if colorspace.upper() == "GRAY":
748: (12)                       colorspace = pymupdf.csGRAY
749: (8)                    elif colorspace.upper() == "CMYK":
750: (12)                       colorspace = pymupdf.csCMYK
751: (8)                    else:
752: (12)                       colorspace = pymupdf.csRGB
753: (4)                if colorspace.n not in (1, 3, 4):
754: (8)                    raise ValueError("unsupported colorspace")
755: (4)                dl = page.get_displaylist(annots=annots)
756: (4)                pix = dl.get_pixmap(matrix=matrix, colorspace=colorspace, alpha=alpha,
clip=clip)
757: (4)                dl = None
758: (4)                if dpi:
759: (8)                    pix.set_dpi(dpi, dpi)
760: (4)                return pix
761: (0)            def get_page_pixmap(
762: (4)                doc: pymupdf.Document,
763: (4)                pno: int,
764: (4)                *,
765: (4)                matrix: matrix_like = pymupdf.Identity,
766: (4)                dpi=None,
767: (4)                colorspace: pymupdf.Colorspace = pymupdf.csRGB,
768: (4)                clip: rect_like = None,
769: (4)                alpha: bool = False,
770: (4)                annots: bool = True,
771: (0)            ) -> pymupdf.Pixmap:
772: (4)                """Create pixmap of document page by page number.
773: (4)                Notes:
774: (8)                    Convenience function calling page.get_pixmap.
775: (4)                Args:
776: (8)                    pno: (int) page number
777: (8)                    matrix: pymupdf.Matrix for transformation (default: pymupdf.Identity).
778: (8)                    colorspace: (str,pymupdf.Colorspace) rgb, rgb, gray - case ignored,
default csRGB.
779: (8)                    clip: (irect-like) restrict rendering to this area.
780: (8)                    alpha: (bool) include alpha channel
781: (8)                    annots: (bool) also render annotations
782: (4)                """
783: (4)                return doc[pno].get_pixmap(
784: (12)                       matrix=matrix,
785: (12)                       dpi=dpi, colorspace=colorspace,
786: (12)                       clip=clip,
787: (12)                       alpha=alpha,
788: (12)                       annots=annots
789: (12)                       )
790: (0)            def getLinkDict(ln, document=None) -> dict:
791: (4)                if isinstance(ln, pymupdf.Outline):
792: (8)                    dest = ln.destination(document)
793: (4)                elif isinstance(ln, pymupdf.Link):
794: (8)                    dest = ln.dest
795: (4)                else:
796: (8)                    assert 0, f'Unexpected {type(ln)=}.'
797: (4)                nl = {"kind": dest.kind, "xref": 0}
798: (4)                try:
799: (8)                    if hasattr(ln, 'rect'):
800: (12)                       nl["from"] = ln.rect
801: (4)                except Exception:
802: (8)                    if g_exceptions_verbose >= 2:   pymupdf.exception_info()
803: (8)                    pass
804: (4)                pnt = pymupdf.Point(0, 0)
805: (4)                if dest.flags & pymupdf.LINK_FLAG_L_VALID:
806: (8)                    pnt.x = dest.lt.x
807: (4)                if dest.flags & pymupdf.LINK_FLAG_T_VALID:
808: (8)                    pnt.y = dest.lt.y
```

```
809: (4)                if dest.kind == pymupdf.LINK_URI:
810: (8)                    nl["uri"] = dest.uri
811: (4)                elif dest.kind == pymupdf.LINK_GOTO:
812: (8)                    nl["page"] = dest.page
813: (8)                    nl["to"] = pnt
814: (8)                    if dest.flags & pymupdf.LINK_FLAG_R_IS_ZOOM:
815: (12)                       nl["zoom"] = dest.rb.x
816: (8)                    else:
817: (12)                       nl["zoom"] = 0.0
818: (4)                elif dest.kind == pymupdf.LINK_GOTOR:
819: (8)                    nl["file"] = dest.file_spec.replace("\\", "/")
820: (8)                    nl["page"] = dest.page
821: (8)                    if dest.page < 0:
822: (12)                       nl["to"] = dest.dest
823: (8)                    else:
824: (12)                       nl["to"] = pnt
825: (12)                       if dest.flags & pymupdf.LINK_FLAG_R_IS_ZOOM:
826: (16)                           nl["zoom"] = dest.rb.x
827: (12)                       else:
828: (16)                           nl["zoom"] = 0.0
829: (4)                elif dest.kind == pymupdf.LINK_LAUNCH:
830: (8)                    nl["file"] = dest.file_spec.replace("\\", "/")
831: (4)                elif dest.kind == pymupdf.LINK_NAMED:
832: (8)                    assert not (dest.named.keys() & nl.keys())
833: (8)                    nl.update(dest.named)
834: (8)                    if 'to' in nl:
835: (12)                       nl['to'] = pymupdf.Point(nl['to'])
836: (4)                else:
837: (8)                    nl["page"] = dest.page
838: (4)                return nl
839: (0)            def get_links(page: pymupdf.Page) -> list:
840: (4)                """Create a list of all links contained in a PDF page.
841: (4)                Notes:
842: (8)                    see PyMuPDF ducmentation for details.
843: (4)                """
844: (4)                pymupdf.CheckParent(page)
845: (4)                ln = page.first_link
846: (4)                links = []
847: (4)                while ln:
848: (8)                    nl = getLinkDict(ln, page.parent)
849: (8)                    links.append(nl)
850: (8)                    ln = ln.next
851: (4)                if links != [] and page.parent.is_pdf:
852: (8)                    linkxrefs = [x for x in
853: (16)                           pymupdf.JM_get_annot_xref_list2(page)
854: (16)                           if x[1] == pymupdf.PDF_ANNOT_LINK  # pylint: disable=no-member
855: (16)                           ]
856: (8)                    if len(linkxrefs) == len(links):
857: (12)                       for i in range(len(linkxrefs)):
858: (16)                           links[i]["xref"] = linkxrefs[i][0]
859: (16)                           links[i]["id"] = linkxrefs[i][2]
860: (4)                return links
861: (0)            def get_toc(
862: (4)                doc: pymupdf.Document,
863: (4)                simple: bool = True,
864: (0)            ) -> list:
865: (4)                """Create a table of contents.
866: (4)                Args:
867: (8)                    simple: a bool to control output. Returns a list, where each entry
consists of outline level, title, page number and link destination (if simple = False). For
details see PyMuPDF's documentation.
868: (4)                """
869: (4)                def recurse(olItem, liste, lvl):
870: (8)                    """Recursively follow the outline item chain and record item
information in a list."""
871: (8)                    while olItem and olItem.this.m_internal:
872: (12)                       if olItem.title:
873: (16)                           title = olItem.title
874: (12)                       else:
```

```
(16)                            title = " "
(12)                        if not olItem.is_external:
(16)                            if olItem.uri:
(20)                                if olItem.page == -1:
(24)                                    resolve = doc.resolve_link(olItem.uri)
(24)                                    page = resolve[0] + 1
(20)                                else:
(24)                                    page = olItem.page + 1
(16)                            else:
(20)                                page = -1
(12)                        else:
(16)                            page = -1
(12)                        if not simple:
(16)                            link = getLinkDict(olItem, doc)
(16)                            liste.append([lvl, title, page, link])
(12)                        else:
(16)                            liste.append([lvl, title, page])
(12)                        if olItem.down:
(16)                            liste = recurse(olItem.down, liste, lvl + 1)
(12)                        olItem = olItem.next
(8)                     return liste
(4)                 if doc.is_closed:
(8)                     raise ValueError("document closed")
(4)                 doc.init_doc()
(4)                 olItem = doc.outline
(4)                 if not olItem:
(8)                     return []
(4)                 lvl = 1
(4)                 liste = []
(4)                 toc = recurse(olItem, liste, lvl)
(4)                 if doc.is_pdf and simple is False:
(8)                     doc._extend_toc_items(toc)
(4)                 return toc
(0)             def del_toc_item(
(4)                 doc: pymupdf.Document,
(4)                 idx: int,
(0)             ) -> None:
(4)                 """Delete TOC / bookmark item by index."""
(4)                 xref = doc.get_outline_xrefs()[idx]
(4)                 doc._remove_toc_item(xref)
(0)             def set_toc_item(
(4)                 doc: pymupdf.Document,
(4)                 idx: int,
(4)                 dest_dict: OptDict = None,
(4)                 kind: OptInt = None,
(4)                 pno: OptInt = None,
(4)                 uri: OptStr = None,
(4)                 title: OptStr = None,
(4)                 to: point_like = None,
(4)                 filename: OptStr = None,
(4)                 zoom: float = 0,
(0)             ) -> None:
(4)                 """Update TOC item by index.
(4)                 It allows changing the item's title and link destination.
(4)                 Args:
(8)                     idx:
(12)                        (int) desired index of the TOC list, as created by get_toc.
(8)                     dest_dict:
(12)                        (dict) destination dictionary as created by get_toc(False).
(12)                        Outrules all other parameters. If None, the remaining parameters
(12)                        are used to make a dest dictionary.
(8)                     kind:
(12)                        (int) kind of link (pymupdf.LINK_GOTO, etc.). If None, then only
(12)                        the title will be updated. If pymupdf.LINK_NONE, the TOC item will
(12)                        be deleted.
(8)                     pno:
(12)                        (int) page number (1-based like in get_toc). Required if
(12)                        pymupdf.LINK_GOTO.
(8)                     uri:
```

```
944: (12)                        (str) the URL, required if pymupdf.LINK_URI.
945: (8)                    title:
946: (12)                        (str) the new title. No change if None.
947: (8)                    to:
948: (12)                        (point-like) destination on the target page. If omitted, (72, 36)
949: (12)                        will be used as target coordinates.
950: (8)                    filename:
951: (12)                        (str) destination filename, required for pymupdf.LINK_GOTOR and
952: (12)                        pymupdf.LINK_LAUNCH.
953: (8)                    name:
954: (12)                        (str) a destination name for pymupdf.LINK_NAMED.
955: (8)                    zoom:
956: (12)                        (float) a zoom factor for the target location (pymupdf.LINK_GOTO).
957: (4)                """
958: (4)                xref = doc.get_outline_xrefs()[idx]
959: (4)                page_xref = 0
960: (4)                if type(dest_dict) is dict:
961: (8)                    if dest_dict["kind"] == pymupdf.LINK_GOTO:
962: (12)                       pno = dest_dict["page"]
963: (12)                       page_xref = doc.page_xref(pno)
964: (12)                       page_height = doc.page_cropbox(pno).height
965: (12)                       to = dest_dict.get('to', pymupdf.Point(72, 36))
966: (12)                       to.y = page_height - to.y
967: (12)                       dest_dict["to"] = to
968: (8)                    action = getDestStr(page_xref, dest_dict)
969: (8)                    if not action.startswith("/A"):
970: (12)                       raise ValueError("bad bookmark dest")
971: (8)                    color = dest_dict.get("color")
972: (8)                    if color:
973: (12)                       color = list(map(float, color))
974: (12)                       if len(color) != 3 or min(color) < 0 or max(color) > 1:
975: (16)                           raise ValueError("bad color value")
976: (8)                    bold = dest_dict.get("bold", False)
977: (8)                    italic = dest_dict.get("italic", False)
978: (8)                    flags = italic + 2 * bold
979: (8)                    collapse = dest_dict.get("collapse")
980: (8)                    return doc._update_toc_item(
981: (12)                       xref,
982: (12)                       action=action[2:],
983: (12)                       title=title,
984: (12)                       color=color,
985: (12)                       flags=flags,
986: (12)                       collapse=collapse,
987: (8)                    )
988: (4)                if kind == pymupdf.LINK_NONE:  # delete bookmark item
989: (8)                    return doc.del_toc_item(idx)
990: (4)                if kind is None and title is None:  # treat as no-op
991: (8)                    return None
992: (4)                if kind is None:  # only update title text
993: (8)                    return doc._update_toc_item(xref, action=None, title=title)
994: (4)                if kind == pymupdf.LINK_GOTO:
995: (8)                    if pno is None or pno not in range(1, doc.page_count + 1):
996: (12)                       raise ValueError("bad page number")
997: (8)                    page_xref = doc.page_xref(pno - 1)
998: (8)                    page_height = doc.page_cropbox(pno - 1).height
999: (8)                    if to is None:
1000: (12)                      to = pymupdf.Point(72, page_height - 36)
1001: (8)                   else:
1002: (12)                      to = pymupdf.Point(to)
1003: (12)                      to.y = page_height - to.y
1004: (4)               ddict = {
1005: (8)                   "kind": kind,
1006: (8)                   "to": to,
1007: (8)                   "uri": uri,
1008: (8)                   "page": pno,
1009: (8)                   "file": filename,
1010: (8)                   "zoom": zoom,
1011: (4)               }
1012: (4)               action = getDestStr(page_xref, ddict)
```

```
1013: (4)               if action == "" or not action.startswith("/A"):
1014: (8)                   raise ValueError("bad bookmark dest")
1015: (4)               return doc._update_toc_item(xref, action=action[2:], title=title)
1016: (0)           def get_area(*args) -> float:
1017: (4)               """Calculate area of rectangle.\nparameter is one of 'px' (default), 'in',
'cm', or 'mm'."""
1018: (4)               rect = args[0]
1019: (4)               if len(args) > 1:
1020: (8)                   unit = args[1]
1021: (4)               else:
1022: (8)                   unit = "px"
1023: (4)               u = {"px": (1, 1), "in": (1.0, 72.0), "cm": (2.54, 72.0), "mm": (25.4,
72.0)}
1024: (4)               f = (u[unit][0] / u[unit][1]) ** 2
1025: (4)               return f * rect.width * rect.height
1026: (0)           def set_metadata(doc: pymupdf.Document, m: dict = None) -> None:
1027: (4)               """Update the PDF /Info object.
1028: (4)               Args:
1029: (8)                   m: a dictionary like doc.metadata.
1030: (4)               """
1031: (4)               if not doc.is_pdf:
1032: (8)                   raise ValueError("is no PDF")
1033: (4)               if doc.is_closed or doc.is_encrypted:
1034: (8)                   raise ValueError("document closed or encrypted")
1035: (4)               if m is None:
1036: (8)                   m = {}
1037: (4)               elif type(m) is not dict:
1038: (8)                   raise ValueError("bad metadata")
1039: (4)               keymap = {
1040: (8)                   "author": "Author",
1041: (8)                   "producer": "Producer",
1042: (8)                   "creator": "Creator",
1043: (8)                   "title": "Title",
1044: (8)                   "format": None,
1045: (8)                   "encryption": None,
1046: (8)                   "creationDate": "CreationDate",
1047: (8)                   "modDate": "ModDate",
1048: (8)                   "subject": "Subject",
1049: (8)                   "keywords": "Keywords",
1050: (8)                   "trapped": "Trapped",
1051: (4)               }
1052: (4)               valid_keys = set(keymap.keys())
1053: (4)               diff_set = set(m.keys()).difference(valid_keys)
1054: (4)               if diff_set != set():
1055: (8)                   msg = "bad dict key(s): %s" % diff_set
1056: (8)                   raise ValueError(msg)
1057: (4)               t, temp = doc.xref_get_key(-1, "Info")
1058: (4)               if t != "xref":
1059: (8)                   info_xref = 0
1060: (4)               else:
1061: (8)                   info_xref = int(temp.replace("0 R", ""))
1062: (4)               if m == {} and info_xref == 0:  # nothing to do
1063: (8)                   return
1064: (4)               if info_xref == 0:  # no prev metadata: get new xref
1065: (8)                   info_xref = doc.get_new_xref()
1066: (8)                   doc.update_object(info_xref, "<<>>")  # fill it with empty object
1067: (8)                   doc.xref_set_key(-1, "Info", "%i 0 R" % info_xref)
1068: (4)               elif m == {}:  # remove existing metadata
1069: (8)                   doc.xref_set_key(-1, "Info", "null")
1070: (8)                   doc.init_doc()
1071: (8)                   return
1072: (4)               for key, val in [(k, v) for k, v in m.items() if keymap[k] is not None]:
1073: (8)                   pdf_key = keymap[key]
1074: (8)                   if not bool(val) or val in ("none", "null"):
1075: (12)                      val = "null"
1076: (8)                   else:
1077: (12)                      val = pymupdf.get_pdf_str(val)
1078: (8)                   doc.xref_set_key(info_xref, pdf_key, val)
1079: (4)               doc.init_doc()
```

```
1080: (4)               return
1081: (0)           def getDestStr(xref: int, ddict: dict) -> str:
1082: (4)               """Calculate the PDF action string.
1083: (4)               Notes:
1084: (8)                   Supports Link annotations and outline items (bookmarks).
1085: (4)               """
1086: (4)               if not ddict:
1087: (8)                   return ""
1088: (4)               str_goto = lambda a, b, c, d: f"/A<</S/GoTo/D[{a} 0 R/XYZ {_format_g((b,
c, d))}]>>"
1089: (4)               str_gotor1 = lambda a, b, c, d, e, f: f"/A<</S/GoToR/D[{a} /XYZ
{_format_g((b, c, d))}]/F<</F{e}/UF{f}/Type/Filespec>>>>"
1090: (4)               str_gotor2 = lambda a, b, c:
f"/A<</S/GoToR/D{a}/F<</F{b}/UF{c}/Type/Filespec>>>>"
1091: (4)               str_launch = lambda a, b:
f"/A<</S/Launch/F<</F{a}/UF{b}/Type/Filespec>>>>"
1092: (4)               str_uri = lambda a: f"/A<</S/URI/URI{a}>>"
1093: (4)               if type(ddict) in (int, float):
1094: (8)                   dest = str_goto(xref, 0, ddict, 0)
1095: (8)                   return dest
1096: (4)               d_kind = ddict.get("kind", pymupdf.LINK_NONE)
1097: (4)               if d_kind == pymupdf.LINK_NONE:
1098: (8)                   return ""
1099: (4)               if ddict["kind"] == pymupdf.LINK_GOTO:
1100: (8)                   d_zoom = ddict.get("zoom", 0)
1101: (8)                   to = ddict.get("to", pymupdf.Point(0, 0))
1102: (8)                   d_left, d_top = to
1103: (8)                   dest = str_goto(xref, d_left, d_top, d_zoom)
1104: (8)                   return dest
1105: (4)               if ddict["kind"] == pymupdf.LINK_URI:
1106: (8)                   dest = str_uri(pymupdf.get_pdf_str(ddict["uri"]),)
1107: (8)                   return dest
1108: (4)               if ddict["kind"] == pymupdf.LINK_LAUNCH:
1109: (8)                   fspec = pymupdf.get_pdf_str(ddict["file"])
1110: (8)                   dest = str_launch(fspec, fspec)
1111: (8)                   return dest
1112: (4)               if ddict["kind"] == pymupdf.LINK_GOTOR and ddict["page"] < 0:
1113: (8)                   fspec = pymupdf.get_pdf_str(ddict["file"])
1114: (8)                   dest = str_gotor2(pymupdf.get_pdf_str(ddict["to"]), fspec, fspec)
1115: (8)                   return dest
1116: (4)               if ddict["kind"] == pymupdf.LINK_GOTOR and ddict["page"] >= 0:
1117: (8)                   fspec = pymupdf.get_pdf_str(ddict["file"])
1118: (8)                   dest = str_gotor1(
1119: (12)                      ddict["page"],
1120: (12)                      ddict["to"].x,
1121: (12)                      ddict["to"].y,
1122: (12)                      ddict["zoom"],
1123: (12)                      fspec,
1124: (12)                      fspec,
1125: (8)                   )
1126: (8)                   return dest
1127: (4)               return ""
1128: (0)           def set_toc(
1129: (4)               doc: pymupdf.Document,
1130: (4)               toc: list,
1131: (4)               collapse: int = 1,
1132: (0)           ) -> int:
1133: (4)               """Create new outline tree (table of contents, TOC).
1134: (4)               Args:
1135: (8)                   toc: (list, tuple) each entry must contain level, title, page and
1136: (12)                      optionally top margin on the page. None or '()' remove the TOC.
1137: (8)                   collapse: (int) collapses entries beyond this level. Zero or None
1138: (12)                      shows all entries unfolded.
1139: (4)               Returns:
1140: (8)                   the number of inserted items, or the number of removed items
respectively.
1141: (4)               """
1142: (4)               if doc.is_closed or doc.is_encrypted:
1143: (8)                   raise ValueError("document closed or encrypted")
```

```
(4)     if not doc.is_pdf:
(8)         raise ValueError("is no PDF")
(4)     if not toc:  # remove all entries
(8)         return len(doc._delToC())
(4)     if type(toc) not in (list, tuple):
(8)         raise ValueError("'toc' must be list or tuple")
(4)     toclen = len(toc)
(4)     page_count = doc.page_count
(4)     t0 = toc[0]
(4)     if type(t0) not in (list, tuple):
(8)         raise ValueError("items must be sequences of 3 or 4 items")
(4)     if t0[0] != 1:
(8)         raise ValueError("hierarchy level of item 0 must be 1")
(4)     for i in list(range(toclen - 1)):
(8)         t1 = toc[i]
(8)         t2 = toc[i + 1]
(8)         if not -1 <= t1[2] <= page_count:
(12)            raise ValueError("row %i: page number out of range" % i)
(8)         if (type(t2) not in (list, tuple)) or len(t2) not in (3, 4):
(12)            raise ValueError("bad row %i" % (i + 1))
(8)         if (type(t2[0]) is not int) or t2[0] < 1:
(12)            raise ValueError("bad hierarchy level in row %i" % (i + 1))
(8)         if t2[0] > t1[0] + 1:
(12)            raise ValueError("bad hierarchy level in row %i" % (i + 1))
(4)     old_xrefs = doc._delToC()  # del old outlines, get their xref numbers
(4)     old_xrefs = []
(4)     xref = [0] + old_xrefs
(4)     xref[0] = doc._getOLRootNumber()  # entry zero is outline root xref number
(4)     if toclen > len(old_xrefs):  # too few old xrefs?
(8)         for i in range((toclen - len(old_xrefs))):
(12)            xref.append(doc.get_new_xref())  # acquire new ones
(4)     lvltab = {0: 0}  # to store last entry per hierarchy level
(4)     olitems = [{"count": 0, "first": -1, "last": -1, "xref": xref[0]}]
(4)     for i in range(toclen):
(8)         o = toc[i]
(8)         lvl = o[0]  # level
(8)         title = pymupdf.get_pdf_str(o[1])  # title
(8)         pno = min(doc.page_count - 1, max(0, o[2] - 1))  # page number
(8)         page_xref = doc.page_xref(pno)
(8)         page_height = doc.page_cropbox(pno).height
(8)         top = pymupdf.Point(72, page_height - 36)
(8)         dest_dict = {"to": top, "kind": pymupdf.LINK_GOTO}  # fall back target
(8)         if o[2] < 0:
(12)            dest_dict["kind"] = pymupdf.LINK_NONE
(8)         if len(o) > 3:  # some target is specified
(12)            if type(o[3]) in (int, float):  # convert a number to a point
(16)                dest_dict["to"] = pymupdf.Point(72, page_height - o[3])
(12)            else:  # if something else, make sure we have a dict
(16)                dest_dict = o[3].copy() if type(o[3]) is dict else dest_dict
(16)                if "to" not in dest_dict:  # target point not in dict?
(20)                    dest_dict["to"] = top  # put default in
(16)                else:  # transform target to PDF coordinates
(20)                    page = doc[pno]
(20)                    point = pymupdf.Point(dest_dict["to"])
(20)                    point.y = page.cropbox.height - point.y
(20)                    point = point * page.rotation_matrix
(20)                    dest_dict["to"] = (point.x, point.y)
(8)         d = {}
(8)         d["first"] = -1
(8)         d["count"] = 0
(8)         d["last"] = -1
(8)         d["prev"] = -1
(8)         d["next"] = -1
(8)         d["dest"] = getDestStr(page_xref, dest_dict)
(8)         d["top"] = dest_dict["to"]
(8)         d["title"] = title
(8)         d["parent"] = lvltab[lvl - 1]
(8)         d["xref"] = xref[i + 1]
(8)         d["color"] = dest_dict.get("color")
```

```
1213: (8)                    d["flags"] = dest_dict.get("italic", 0) + 2 * dest_dict.get("bold", 0)
1214: (8)                    lvltab[lvl] = i + 1
1215: (8)                    parent = olitems[lvltab[lvl - 1]]  # the parent entry
1216: (8)                    if (
1217: (12)                       dest_dict.get("collapse") or collapse and lvl > collapse
1218: (8)                    ):  # suppress expansion
1219: (12)                       parent["count"] -= 1  # make /Count negative
1220: (8)                    else:
1221: (12)                       parent["count"] += 1  # positive /Count
1222: (8)                    if parent["first"] == -1:
1223: (12)                       parent["first"] = i + 1
1224: (12)                       parent["last"] = i + 1
1225: (8)                    else:
1226: (12)                       d["prev"] = parent["last"]
1227: (12)                       prev = olitems[parent["last"]]
1228: (12)                       prev["next"] = i + 1
1229: (12)                       parent["last"] = i + 1
1230: (8)                    olitems.append(d)
1231: (4)                for i, ol in enumerate(olitems):
1232: (8)                    txt = "<<"
1233: (8)                    if ol["count"] != 0:
1234: (12)                       txt += "/Count %i" % ol["count"]
1235: (8)                    try:
1236: (12)                       txt += ol["dest"]
1237: (8)                    except Exception:
1238: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1239: (12)                       pass
1240: (8)                    try:
1241: (12)                       if ol["first"] > -1:
1242: (16)                           txt += "/First %i 0 R" % xref[ol["first"]]
1243: (8)                    except Exception:
1244: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1245: (12)                       pass
1246: (8)                    try:
1247: (12)                       if ol["last"] > -1:
1248: (16)                           txt += "/Last %i 0 R" % xref[ol["last"]]
1249: (8)                    except Exception:
1250: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1251: (12)                       pass
1252: (8)                    try:
1253: (12)                       if ol["next"] > -1:
1254: (16)                           txt += "/Next %i 0 R" % xref[ol["next"]]
1255: (8)                    except Exception:
1256: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1257: (12)                       pass
1258: (8)                    try:
1259: (12)                       if ol["parent"] > -1:
1260: (16)                           txt += "/Parent %i 0 R" % xref[ol["parent"]]
1261: (8)                    except Exception:
1262: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1263: (12)                       pass
1264: (8)                    try:
1265: (12)                       if ol["prev"] > -1:
1266: (16)                           txt += "/Prev %i 0 R" % xref[ol["prev"]]
1267: (8)                    except Exception:
1268: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1269: (12)                       pass
1270: (8)                    try:
1271: (12)                       txt += "/Title" + ol["title"]
1272: (8)                    except Exception:
1273: (12)                       if g_exceptions_verbose >= 2:   pymupdf.exception_info()
1274: (12)                       pass
1275: (8)                    if ol.get("color") and len(ol["color"]) == 3:
1276: (12)                       txt += f"/C[ {_format_g(tuple(ol['color']))}]"
1277: (8)                    if ol.get("flags", 0) > 0:
1278: (12)                       txt += "/F %i" % ol["flags"]
1279: (8)                    if i == 0:  # special: this is the outline root
1280: (12)                       txt += "/Type/Outlines"  # so add the /Type entry
1281: (8)                    txt += ">>"
```

```
1282: (8)                doc.update_object(xref[i], txt)  # insert the PDF object
1283: (4)            doc.init_doc()
1284: (4)            return toclen
1285: (0)        def do_links(
1286: (4)            doc1: pymupdf.Document,
1287: (4)            doc2: pymupdf.Document,
1288: (4)            from_page: int = -1,
1289: (4)            to_page: int = -1,
1290: (4)            start_at: int = -1,
1291: (0)        ) -> None:
1292: (4)            """Insert links contained in copied page range into destination PDF.
1293: (4)            Parameter values **must** equal those of method insert_pdf(), which must
1294: (4)            have been previously executed.
1295: (4)            """
1296: (4)            def cre_annot(lnk, xref_dst, pno_src, ctm):
1297: (8)                """Create annotation object string for a passed-in link."""
1298: (8)                r = lnk["from"] * ctm  # rect in PDF coordinates
1299: (8)                rect = _format_g(tuple(r))
1300: (8)                if lnk["kind"] == pymupdf.LINK_GOTO:
1301: (12)                    txt = pymupdf.annot_skel["goto1"]  # annot_goto
1302: (12)                    idx = pno_src.index(lnk["page"])
1303: (12)                    p = lnk["to"] * ctm  # target point in PDF coordinates
1304: (12)                    annot = txt(xref_dst[idx], p.x, p.y, lnk["zoom"], rect)
1305: (8)                elif lnk["kind"] == pymupdf.LINK_GOTOR:
1306: (12)                    if lnk["page"] >= 0:
1307: (16)                        txt = pymupdf.annot_skel["gotor1"]  # annot_gotor
1308: (16)                        pnt = lnk.get("to", pymupdf.Point(0, 0))  # destination point
1309: (16)                        if type(pnt) is not pymupdf.Point:
1310: (20)                            pnt = pymupdf.Point(0, 0)
1311: (16)                        annot = txt(
1312: (20)                            lnk["page"],
1313: (20)                            pnt.x,
1314: (20)                            pnt.y,
1315: (20)                            lnk["zoom"],
1316: (20)                            lnk["file"],
1317: (20)                            lnk["file"],
1318: (20)                            rect,
1319: (16)                        )
1320: (12)                    else:
1321: (16)                        txt = pymupdf.annot_skel["gotor2"]  # annot_gotor_n
1322: (16)                        to = pymupdf.get_pdf_str(lnk["to"])
1323: (16)                        to = to[1:-1]
1324: (16)                        f = lnk["file"]
1325: (16)                        annot = txt(to, f, rect)
1326: (8)                elif lnk["kind"] == pymupdf.LINK_LAUNCH:
1327: (12)                    txt = pymupdf.annot_skel["launch"]  # annot_launch
1328: (12)                    annot = txt(lnk["file"], lnk["file"], rect)
1329: (8)                elif lnk["kind"] == pymupdf.LINK_URI:
1330: (12)                    txt = pymupdf.annot_skel["uri"]  # annot_uri
1331: (12)                    annot = txt(lnk["uri"], rect)
1332: (8)                else:
1333: (12)                    annot = ""
1334: (8)                return annot
1335: (4)            if from_page < 0:
1336: (8)                fp = 0
1337: (4)            elif from_page >= doc2.page_count:
1338: (8)                fp = doc2.page_count - 1
1339: (4)            else:
1340: (8)                fp = from_page
1341: (4)            if to_page < 0 or to_page >= doc2.page_count:
1342: (8)                tp = doc2.page_count - 1
1343: (4)            else:
1344: (8)                tp = to_page
1345: (4)            if start_at < 0:
1346: (8)                raise ValueError("'start_at' must be >= 0")
1347: (4)            sa = start_at
1348: (4)            incr = 1 if fp <= tp else -1  # page range could be reversed
1349: (4)            pno_src = list(range(fp, tp + incr, incr))
1350: (4)            pno_dst = [sa + i for i in range(len(pno_src))]
```

```
1351: (4)               xref_src = []
1352: (4)               xref_dst = []
1353: (4)               for i in range(len(pno_src)):
1354: (8)                   p_src = pno_src[i]
1355: (8)                   p_dst = pno_dst[i]
1356: (8)                   old_xref = doc2.page_xref(p_src)
1357: (8)                   new_xref = doc1.page_xref(p_dst)
1358: (8)                   xref_src.append(old_xref)
1359: (8)                   xref_dst.append(new_xref)
1360: (4)               for i in range(len(xref_src)):
1361: (8)                   page_src = doc2[pno_src[i]]  # load source page
1362: (8)                   links = page_src.get_links()  # get all its links
1363: (8)                   if len(links) == 0:  # no links there
1364: (12)                      page_src = None
1365: (12)                      continue
1366: (8)                   ctm = ~page_src.transformation_matrix  # calc page transformation
matrix
1367: (8)                   page_dst = doc1[pno_dst[i]]  # load destination page
1368: (8)                   link_tab = []  # store all link definitions here
1369: (8)                   for l in links:
1370: (12)                      if l["kind"] == pymupdf.LINK_GOTO and (l["page"] not in pno_src):
1371: (16)                          continue  # GOTO link target not in copied pages
1372: (12)                      annot_text = cre_annot(l, xref_dst, pno_src, ctm)
1373: (12)                      if annot_text:
1374: (16)                          link_tab.append(annot_text)
1375: (8)                   if link_tab != []:
1376: (12)                      page_dst._addAnnot_FromString( tuple(link_tab))
1377: (0)           def getLinkText(page: pymupdf.Page, lnk: dict) -> str:
1378: (4)               ctm = page.transformation_matrix
1379: (4)               ictm = ~ctm
1380: (4)               r = lnk["from"]
1381: (4)               rect = _format_g(tuple(r * ictm))
1382: (4)               annot = ""
1383: (4)               if lnk["kind"] == pymupdf.LINK_GOTO:
1384: (8)                   if lnk["page"] >= 0:
1385: (12)                      txt = pymupdf.annot_skel["goto1"]  # annot_goto
1386: (12)                      pno = lnk["page"]
1387: (12)                      xref = page.parent.page_xref(pno)
1388: (12)                      pnt = lnk.get("to", pymupdf.Point(0, 0))  # destination point
1389: (12)                      dest_page = page.parent[pno]
1390: (12)                      dest_ctm = dest_page.transformation_matrix
1391: (12)                      dest_ictm = ~dest_ctm
1392: (12)                      ipnt = pnt * dest_ictm
1393: (12)                      annot = txt(xref, ipnt.x, ipnt.y, lnk.get("zoom", 0), rect)
1394: (8)                   else:
1395: (12)                      txt = pymupdf.annot_skel["goto2"]  # annot_goto_n
1396: (12)                      annot = txt(pymupdf.get_pdf_str(lnk["to"]), rect)
1397: (4)               elif lnk["kind"] == pymupdf.LINK_GOTOR:
1398: (8)                   if lnk["page"] >= 0:
1399: (12)                      txt = pymupdf.annot_skel["gotor1"]  # annot_gotor
1400: (12)                      pnt = lnk.get("to", pymupdf.Point(0, 0))  # destination point
1401: (12)                      if type(pnt) is not pymupdf.Point:
1402: (16)                          pnt = pymupdf.Point(0, 0)
1403: (12)                      annot = txt(
1404: (16)                          lnk["page"],
1405: (16)                          pnt.x,
1406: (16)                          pnt.y,
1407: (16)                          lnk.get("zoom", 0),
1408: (16)                          lnk["file"],
1409: (16)                          lnk["file"],
1410: (16)                          rect,
1411: (12)                      )
1412: (8)                   else:
1413: (12)                      txt = pymupdf.annot_skel["gotor2"]  # annot_gotor_n
1414: (12)                      annot = txt(pymupdf.get_pdf_str(lnk["to"]), lnk["file"], rect)
1415: (4)               elif lnk["kind"] == pymupdf.LINK_LAUNCH:
1416: (8)                   txt = pymupdf.annot_skel["launch"]  # annot_launch
1417: (8)                   annot = txt(lnk["file"], lnk["file"], rect)
1418: (4)               elif lnk["kind"] == pymupdf.LINK_URI:
```

```
1419: (8)                   txt = pymupdf.annot_skel["uri"]  # txt = annot_uri
1420: (8)                   annot = txt(lnk["uri"], rect)
1421: (4)               elif lnk["kind"] == pymupdf.LINK_NAMED:
1422: (8)                   txt = pymupdf.annot_skel["named"]  # annot_named
1423: (8)                   lname = lnk.get("name")  # check presence of key
1424: (8)                   if lname is None:  # if missing, fall back to alternative
1425: (12)                      lname = lnk["nameddest"]
1426: (8)                   annot = txt(lname, rect)
1427: (4)               if not annot:
1428: (8)                   return annot
1429: (4)               link_names = dict(  # existing ids and their xref
1430: (8)                   [(x[0], x[2]) for x in page.annot_xrefs() if x[1] ==
pymupdf.PDF_ANNOT_LINK]   # pylint: disable=no-member
1431: (4)               )
1432: (4)               old_name = lnk.get("id", "")  # id value in the argument
1433: (4)               if old_name and (lnk["xref"], old_name) in link_names.items():
1434: (8)                   name = old_name  # no new name if this is an update only
1435: (4)               else:
1436: (8)                   i = 0
1437: (8)                   stem = pymupdf.TOOLS.set_annot_stem() + "-L%i"
1438: (8)                   while True:
1439: (12)                      name = stem % i
1440: (12)                      if name not in link_names.values():
1441: (16)                          break
1442: (12)                      i += 1
1443: (4)               annot = annot.replace("/Link", "/Link/NM(%s)" % name)
1444: (4)               return annot
1445: (0)           def delete_widget(page: pymupdf.Page, widget: pymupdf.Widget) ->
pymupdf.Widget:
1446: (4)               """Delete widget from page and return the next one."""
1447: (4)               pymupdf.CheckParent(page)
1448: (4)               annot = getattr(widget, "_annot", None)
1449: (4)               if annot is None:
1450: (8)                   raise ValueError("bad type: widget")
1451: (4)               nextwidget = widget.next
1452: (4)               page.delete_annot(annot)
1453: (4)               widget._annot.parent = None
1454: (4)               keylist = list(widget.__dict__.keys())
1455: (4)               for key in keylist:
1456: (8)                   del widget.__dict__[key]
1457: (4)               return nextwidget
1458: (0)           def update_link(page: pymupdf.Page, lnk: dict) -> None:
1459: (4)               """Update a link on the current page."""
1460: (4)               pymupdf.CheckParent(page)
1461: (4)               annot = getLinkText(page, lnk)
1462: (4)               if annot == "":
1463: (8)                   raise ValueError("link kind not supported")
1464: (4)               page.parent.update_object(lnk["xref"], annot, page=page)
1465: (0)           def insert_link(page: pymupdf.Page, lnk: dict, mark: bool = True) -> None:
1466: (4)               """Insert a new link for the current page."""
1467: (4)               pymupdf.CheckParent(page)
1468: (4)               annot = getLinkText(page, lnk)
1469: (4)               if annot == "":
1470: (8)                   raise ValueError("link kind not supported")
1471: (4)               page._addAnnot_FromString((annot,))
1472: (0)           def insert_textbox(
1473: (4)               page: pymupdf.Page,
1474: (4)               rect: rect_like,
1475: (4)               buffer: typing.Union[str, list],
1476: (4)               fontname: str = "helv",
1477: (4)               fontfile: OptStr = None,
1478: (4)               set_simple: int = 0,
1479: (4)               encoding: int = 0,
1480: (4)               fontsize: float = 11,
1481: (4)               lineheight: OptFloat = None,
1482: (4)               color: OptSeq = None,
1483: (4)               fill: OptSeq = None,
1484: (4)               expandtabs: int = 1,
1485: (4)               align: int = 0,
```

```
(4)                 rotate: int = 0,
(4)                 render_mode: int = 0,
(4)                 border_width: float = 0.05,
(4)                 morph: OptSeq = None,
(4)                 overlay: bool = True,
(4)                 stroke_opacity: float = 1,
(4)                 fill_opacity: float = 1,
(4)                 oc: int = 0,
(0)             ) -> float:
(4)                 """Insert text into a given rectangle.
(4)                 Notes:
(8)                     Creates a Shape object, uses its same-named method and commits it.
(4)                 Parameters:
(8)                     rect: (rect-like) area to use for text.
(8)                     buffer: text to be inserted
(8)                     fontname: a Base-14 font, font name or '/name'
(8)                     fontfile: name of a font file
(8)                     fontsize: font size
(8)                     lineheight: overwrite the font property
(8)                     color: RGB color triple
(8)                     expandtabs: handles tabulators with string function
(8)                     align: left, center, right, justified
(8)                     rotate: 0, 90, 180, or 270 degrees
(8)                     morph: morph box with a matrix and a fixpoint
(8)                     overlay: put text in foreground or background
(4)                 Returns:
(8)                     unused or deficit rectangle area (float)
(4)                 """
(4)                 img = page.new_shape()
(4)                 rc = img.insert_textbox(
(8)                     rect,
(8)                     buffer,
(8)                     fontsize=fontsize,
(8)                     lineheight=lineheight,
(8)                     fontname=fontname,
(8)                     fontfile=fontfile,
(8)                     set_simple=set_simple,
(8)                     encoding=encoding,
(8)                     color=color,
(8)                     fill=fill,
(8)                     expandtabs=expandtabs,
(8)                     render_mode=render_mode,
(8)                     border_width=border_width,
(8)                     align=align,
(8)                     rotate=rotate,
(8)                     morph=morph,
(8)                     stroke_opacity=stroke_opacity,
(8)                     fill_opacity=fill_opacity,
(8)                     oc=oc,
(4)                 )
(4)                 if rc >= 0:
(8)                     img.commit(overlay)
(4)                 return rc
(0)             def insert_text(
(4)                 page: pymupdf.Page,
(4)                 point: point_like,
(4)                 text: typing.Union[str, list],
(4)                 fontsize: float = 11,
(4)                 lineheight: OptFloat = None,
(4)                 fontname: str = "helv",
(4)                 fontfile: OptStr = None,
(4)                 set_simple: int = 0,
(4)                 encoding: int = 0,
(4)                 color: OptSeq = None,
(4)                 fill: OptSeq = None,
(4)                 border_width: float = 0.05,
(4)                 render_mode: int = 0,
(4)                 rotate: int = 0,
(4)                 morph: OptSeq = None,
```

```
1555: (4)               overlay: bool = True,
1556: (4)               stroke_opacity: float = 1,
1557: (4)               fill_opacity: float = 1,
1558: (4)               oc: int = 0,
1559: (0)           ):
1560: (4)               img = page.new_shape()
1561: (4)               rc = img.insert_text(
1562: (8)                   point,
1563: (8)                   text,
1564: (8)                   fontsize=fontsize,
1565: (8)                   lineheight=lineheight,
1566: (8)                   fontname=fontname,
1567: (8)                   fontfile=fontfile,
1568: (8)                   set_simple=set_simple,
1569: (8)                   encoding=encoding,
1570: (8)                   color=color,
1571: (8)                   fill=fill,
1572: (8)                   border_width=border_width,
1573: (8)                   render_mode=render_mode,
1574: (8)                   rotate=rotate,
1575: (8)                   morph=morph,
1576: (8)                   stroke_opacity=stroke_opacity,
1577: (8)                   fill_opacity=fill_opacity,
1578: (8)                   oc=oc,
1579: (4)               )
1580: (4)               if rc >= 0:
1581: (8)                   img.commit(overlay)
1582: (4)               return rc
1583: (0)           def insert_htmlbox(
1584: (4)               page,
1585: (4)               rect,
1586: (4)               text,
1587: (4)               *,
1588: (4)               css=None,
1589: (4)               scale_low=0,
1590: (4)               archive=None,
1591: (4)               rotate=0,
1592: (4)               oc=0,
1593: (4)               opacity=1,
1594: (4)               overlay=True,
1595: (0)           ) -> float:
1596: (4)               """Insert text with optional HTML tags and stylings into a rectangle.
1597: (4)               Args:
1598: (8)                   rect: (rect-like) rectangle into which the text should be placed.
1599: (8)                   text: (str) text with optional HTML tags and stylings.
1600: (8)                   css: (str) CSS styling commands.
1601: (8)                   scale_low: (float) force-fit content by scaling it down. Must be in
1602: (12)                      range [0, 1]. If 1, no scaling will take place. If 0, arbitrary
1603: (12)                      down-scaling is acceptable. A value of 0.1 would mean that content
1604: (12)                      may be scaled down by at most 90%.
1605: (8)                   archive: Archive object pointing to locations of used fonts or images
1606: (8)                   rotate: (int) rotate the text in the box by a multiple of 90 degrees.
1607: (8)                   oc: (int) the xref of an OCG / OCMD (Optional Content).
1608: (8)                   opacity: (float) set opacity of inserted content.
1609: (8)                   overlay: (bool) put text on top of page content.
1610: (4)               Returns:
1611: (8)                   A tuple of floats (spare_height, scale).
1612: (8)                   spare_height: -1 if content did not fit, else >= 0. It is the height
of the
1613: (15)                         unused (still available) rectangle stripe. Positive only if
1614: (15)                         scale_min = 1 (no down scaling).
1615: (8)                   scale: downscaling factor, 0 < scale <= 1. Set to 0 if spare_height =
-1 (no fit).
1616: (4)               """
1617: (4)               if not rotate % 90 == 0:
1618: (8)                   raise ValueError("bad rotation angle")
1619: (4)               while rotate < 0:
1620: (8)                   rotate += 360
1621: (4)               while rotate >= 360:
```

```
1622: (8)                   rotate -= 360
1623: (4)               if not 0 <= scale_low <= 1:
1624: (8)                   raise ValueError("'scale_low' must be in [0, 1]")
1625: (4)               if css is None:
1626: (8)                   css = ""
1627: (4)               rect = pymupdf.Rect(rect)
1628: (4)               if rotate in (90, 270):
1629: (8)                   temp_rect = pymupdf.Rect(0, 0, rect.height, rect.width)
1630: (4)               else:
1631: (8)                   temp_rect = pymupdf.Rect(0, 0, rect.width, rect.height)
1632: (4)               mycss = "body {margin:1px;}" + css  # append user CSS
1633: (4)               if isinstance(text, str):  # if a string, convert to a Story
1634: (8)                   story = pymupdf.Story(html=text, user_css=mycss, archive=archive)
1635: (4)               elif isinstance(text, pymupdf.Story):
1636: (8)                   story = text
1637: (4)               else:
1638: (8)                   raise ValueError("'text' must be a string or a Story")
1639: (4)               scale_max = None if scale_low == 0 else 1 / scale_low
1640: (4)               fit = story.fit_scale(temp_rect, scale_min=1, scale_max=scale_max)
1641: (4)               if fit.big_enough is False:  # there was no fit
1642: (8)                   return (-1, scale_low)
1643: (4)               filled = fit.filled
1644: (4)               scale = 1 / fit.parameter  # shrink factor
1645: (4)               spare_height = fit.rect.y1 - filled[3]  # unused room at rectangle bottom
1646: (4)               if scale != 1 or spare_height < 0:  # if scaling occurred, set
spare_height to 0
1647: (8)                   spare_height = 0
1648: (4)               def rect_function(*args):
1649: (8)                   return fit.rect, fit.rect, pymupdf.Identity
1650: (4)               doc = story.write_with_links(rect_function)
1651: (4)               if 0 <= opacity < 1:
1652: (8)                   tpage = doc[0]  # load page
1653: (8)                   alp0 = tpage._set_opacity(CA=opacity, ca=opacity)
1654: (8)                   s = f"/{alp0} gs\n"  # generate graphic state command
1655: (8)                   pymupdf.TOOLS._insert_contents(tpage, s.encode(), 0)
1656: (4)               page.show_pdf_page(rect, doc, 0, rotate=rotate, oc=oc, overlay=overlay)
1657: (4)               mp1 = (fit.rect.tl + fit.rect.br) / 2 * scale
1658: (4)               mp2 = (rect.tl + rect.br) / 2
1659: (4)               mat = (
1660: (8)                   pymupdf.Matrix(scale, 0, 0, scale, -mp1.x, -mp1.y)
1661: (8)                   * pymupdf.Matrix(-rotate)
1662: (8)                   * pymupdf.Matrix(1, 0, 0, 1, mp2.x, mp2.y)
1663: (4)               )
1664: (4)               for link in doc[0].get_links():
1665: (8)                   link["from"] *= mat
1666: (8)                   page.insert_link(link)
1667: (4)               return spare_height, scale
1668: (0)           def new_page(
1669: (4)               doc: pymupdf.Document,
1670: (4)               pno: int = -1,
1671: (4)               width: float = 595,
1672: (4)               height: float = 842,
1673: (0)           ) -> pymupdf.Page:
1674: (4)               """Create and return a new page object.
1675: (4)               Args:
1676: (8)                   pno: (int) insert before this page. Default: after last page.
1677: (8)                   width: (float) page width in points. Default: 595 (ISO A4 width).
1678: (8)                   height: (float) page height in points. Default 842 (ISO A4 height).
1679: (4)               Returns:
1680: (8)                   A pymupdf.Page object.
1681: (4)               """
1682: (4)               doc._newPage(pno, width=width, height=height)
1683: (4)               return doc[pno]
1684: (0)           def insert_page(
1685: (4)               doc: pymupdf.Document,
1686: (4)               pno: int,
1687: (4)               text: typing.Union[str, list, None] = None,
1688: (4)               fontsize: float = 11,
1689: (4)               width: float = 595,
```

```
1690: (4)               height: float = 842,
1691: (4)               fontname: str = "helv",
1692: (4)               fontfile: OptStr = None,
1693: (4)               color: OptSeq = (0,),
1694: (0)           ) -> int:
1695: (4)               """Create a new PDF page and insert some text.
1696: (4)               Notes:
1697: (8)                   Function combining pymupdf.Document.new_page() and
pymupdf.Page.insert_text().
1698: (8)                   For parameter details see these methods.
1699: (4)               """
1700: (4)               page = doc.new_page(pno=pno, width=width, height=height)
1701: (4)               if not bool(text):
1702: (8)                   return 0
1703: (4)               rc = page.insert_text(
1704: (8)                   (50, 72),
1705: (8)                   text,
1706: (8)                   fontsize=fontsize,
1707: (8)                   fontname=fontname,
1708: (8)                   fontfile=fontfile,
1709: (8)                   color=color,
1710: (4)               )
1711: (4)               return rc
1712: (0)           def draw_line(
1713: (4)               page: pymupdf.Page,
1714: (4)               p1: point_like,
1715: (4)               p2: point_like,
1716: (4)               color: OptSeq = (0,),
1717: (4)               dashes: OptStr = None,
1718: (4)               width: float = 1,
1719: (4)               lineCap: int = 0,
1720: (4)               lineJoin: int = 0,
1721: (4)               overlay: bool = True,
1722: (4)               morph: OptSeq = None,
1723: (4)               stroke_opacity: float = 1,
1724: (4)               fill_opacity: float = 1,
1725: (4)               oc=0,
1726: (0)           ) -> pymupdf.Point:
1727: (4)               """Draw a line from point p1 to point p2."""
1728: (4)               img = page.new_shape()
1729: (4)               p = img.draw_line(pymupdf.Point(p1), pymupdf.Point(p2))
1730: (4)               img.finish(
1731: (8)                   color=color,
1732: (8)                   dashes=dashes,
1733: (8)                   width=width,
1734: (8)                   closePath=False,
1735: (8)                   lineCap=lineCap,
1736: (8)                   lineJoin=lineJoin,
1737: (8)                   morph=morph,
1738: (8)                   stroke_opacity=stroke_opacity,
1739: (8)                   fill_opacity=fill_opacity,
1740: (8)                   oc=oc,
1741: (4)               )
1742: (4)               img.commit(overlay)
1743: (4)               return p
1744: (0)           def draw_squiggle(
1745: (4)               page: pymupdf.Page,
1746: (4)               p1: point_like,
1747: (4)               p2: point_like,
1748: (4)               breadth: float = 2,
1749: (4)               color: OptSeq = (0,),
1750: (4)               dashes: OptStr = None,
1751: (4)               width: float = 1,
1752: (4)               lineCap: int = 0,
1753: (4)               lineJoin: int = 0,
1754: (4)               overlay: bool = True,
1755: (4)               morph: OptSeq = None,
1756: (4)               stroke_opacity: float = 1,
1757: (4)               fill_opacity: float = 1,
```

```
1758: (4)                oc: int = 0,
1759: (0)            ) -> pymupdf.Point:
1760: (4)                """Draw a squiggly line from point p1 to point p2."""
1761: (4)                img = page.new_shape()
1762: (4)                p = img.draw_squiggle(pymupdf.Point(p1), pymupdf.Point(p2),
breadth=breadth)
1763: (4)                img.finish(
1764: (8)                    color=color,
1765: (8)                    dashes=dashes,
1766: (8)                    width=width,
1767: (8)                    closePath=False,
1768: (8)                    lineCap=lineCap,
1769: (8)                    lineJoin=lineJoin,
1770: (8)                    morph=morph,
1771: (8)                    stroke_opacity=stroke_opacity,
1772: (8)                    fill_opacity=fill_opacity,
1773: (8)                    oc=oc,
1774: (4)                )
1775: (4)                img.commit(overlay)
1776: (4)                return p
1777: (0)            def draw_zigzag(
1778: (4)                page: pymupdf.Page,
1779: (4)                p1: point_like,
1780: (4)                p2: point_like,
1781: (4)                breadth: float = 2,
1782: (4)                color: OptSeq = (0,),
1783: (4)                dashes: OptStr = None,
1784: (4)                width: float = 1,
1785: (4)                lineCap: int = 0,
1786: (4)                lineJoin: int = 0,
1787: (4)                overlay: bool = True,
1788: (4)                morph: OptSeq = None,
1789: (4)                stroke_opacity: float = 1,
1790: (4)                fill_opacity: float = 1,
1791: (4)                oc: int = 0,
1792: (0)            ) -> pymupdf.Point:
1793: (4)                """Draw a zigzag line from point p1 to point p2."""
1794: (4)                img = page.new_shape()
1795: (4)                p = img.draw_zigzag(pymupdf.Point(p1), pymupdf.Point(p2), breadth=breadth)
1796: (4)                img.finish(
1797: (8)                    color=color,
1798: (8)                    dashes=dashes,
1799: (8)                    width=width,
1800: (8)                    closePath=False,
1801: (8)                    lineCap=lineCap,
1802: (8)                    lineJoin=lineJoin,
1803: (8)                    morph=morph,
1804: (8)                    stroke_opacity=stroke_opacity,
1805: (8)                    fill_opacity=fill_opacity,
1806: (8)                    oc=oc,
1807: (4)                )
1808: (4)                img.commit(overlay)
1809: (4)                return p
1810: (0)            def draw_rect(
1811: (8)                    page: pymupdf.Page,
1812: (8)                    rect: rect_like,
1813: (8)                    color: OptSeq = (0,),
1814: (8)                    fill: OptSeq = None,
1815: (8)                    dashes: OptStr = None,
1816: (8)                    width: float = 1,
1817: (8)                    lineCap: int = 0,
1818: (8)                    lineJoin: int = 0,
1819: (8)                    morph: OptSeq = None,
1820: (8)                    overlay: bool = True,
1821: (8)                    stroke_opacity: float = 1,
1822: (8)                    fill_opacity: float = 1,
1823: (8)                    oc: int = 0,
1824: (8)                    radius=None,
1825: (8)                    ) -> pymupdf.Point:
```

```
1826: (4)                '''
1827: (4)                Draw a rectangle. See Shape class method for details.
1828: (4)                '''
1829: (4)                img = page.new_shape()
1830: (4)                Q = img.draw_rect(pymupdf.Rect(rect), radius=radius)
1831: (4)                img.finish(
1832: (8)                    color=color,
1833: (8)                    fill=fill,
1834: (8)                    dashes=dashes,
1835: (8)                    width=width,
1836: (8)                    lineCap=lineCap,
1837: (8)                    lineJoin=lineJoin,
1838: (8)                    morph=morph,
1839: (8)                    stroke_opacity=stroke_opacity,
1840: (8)                    fill_opacity=fill_opacity,
1841: (8)                    oc=oc,
1842: (4)                )
1843: (4)                img.commit(overlay)
1844: (4)                return Q
1845: (0)            def draw_quad(
1846: (4)                page: pymupdf.Page,
1847: (4)                quad: quad_like,
1848: (4)                color: OptSeq = (0,),
1849: (4)                fill: OptSeq = None,
1850: (4)                dashes: OptStr = None,
1851: (4)                width: float = 1,
1852: (4)                lineCap: int = 0,
1853: (4)                lineJoin: int = 0,
1854: (4)                morph: OptSeq = None,
1855: (4)                overlay: bool = True,
1856: (4)                stroke_opacity: float = 1,
1857: (4)                fill_opacity: float = 1,
1858: (4)                oc: int = 0,
1859: (0)            ) -> pymupdf.Point:
1860: (4)                """Draw a quadrilateral."""
1861: (4)                img = page.new_shape()
1862: (4)                Q = img.draw_quad(pymupdf.Quad(quad))
1863: (4)                img.finish(
1864: (8)                    color=color,
1865: (8)                    fill=fill,
1866: (8)                    dashes=dashes,
1867: (8)                    width=width,
1868: (8)                    lineCap=lineCap,
1869: (8)                    lineJoin=lineJoin,
1870: (8)                    morph=morph,
1871: (8)                    stroke_opacity=stroke_opacity,
1872: (8)                    fill_opacity=fill_opacity,
1873: (8)                    oc=oc,
1874: (4)                )
1875: (4)                img.commit(overlay)
1876: (4)                return Q
1877: (0)            def draw_polyline(
1878: (4)                page: pymupdf.Page,
1879: (4)                points: list,
1880: (4)                color: OptSeq = (0,),
1881: (4)                fill: OptSeq = None,
1882: (4)                dashes: OptStr = None,
1883: (4)                width: float = 1,
1884: (4)                morph: OptSeq = None,
1885: (4)                lineCap: int = 0,
1886: (4)                lineJoin: int = 0,
1887: (4)                overlay: bool = True,
1888: (4)                closePath: bool = False,
1889: (4)                stroke_opacity: float = 1,
1890: (4)                fill_opacity: float = 1,
1891: (4)                oc: int = 0,
1892: (0)            ) -> pymupdf.Point:
1893: (4)                """Draw multiple connected line segments."""
1894: (4)                img = page.new_shape()
```

```
1895: (4)                Q = img.draw_polyline(points)
1896: (4)                img.finish(
1897: (8)                    color=color,
1898: (8)                    fill=fill,
1899: (8)                    dashes=dashes,
1900: (8)                    width=width,
1901: (8)                    lineCap=lineCap,
1902: (8)                    lineJoin=lineJoin,
1903: (8)                    morph=morph,
1904: (8)                    closePath=closePath,
1905: (8)                    stroke_opacity=stroke_opacity,
1906: (8)                    fill_opacity=fill_opacity,
1907: (8)                    oc=oc,
1908: (4)                )
1909: (4)                img.commit(overlay)
1910: (4)                return Q
1911: (0)            def draw_circle(
1912: (4)                page: pymupdf.Page,
1913: (4)                center: point_like,
1914: (4)                radius: float,
1915: (4)                color: OptSeq = (0,),
1916: (4)                fill: OptSeq = None,
1917: (4)                morph: OptSeq = None,
1918: (4)                dashes: OptStr = None,
1919: (4)                width: float = 1,
1920: (4)                lineCap: int = 0,
1921: (4)                lineJoin: int = 0,
1922: (4)                overlay: bool = True,
1923: (4)                stroke_opacity: float = 1,
1924: (4)                fill_opacity: float = 1,
1925: (4)                oc: int = 0,
1926: (0)            ) -> pymupdf.Point:
1927: (4)                """Draw a circle given its center and radius."""
1928: (4)                img = page.new_shape()
1929: (4)                Q = img.draw_circle(pymupdf.Point(center), radius)
1930: (4)                img.finish(
1931: (8)                    color=color,
1932: (8)                    fill=fill,
1933: (8)                    dashes=dashes,
1934: (8)                    width=width,
1935: (8)                    lineCap=lineCap,
1936: (8)                    lineJoin=lineJoin,
1937: (8)                    morph=morph,
1938: (8)                    stroke_opacity=stroke_opacity,
1939: (8)                    fill_opacity=fill_opacity,
1940: (8)                    oc=oc,
1941: (4)                )
1942: (4)                img.commit(overlay)
1943: (4)                return Q
1944: (0)            def draw_oval(
1945: (4)                page: pymupdf.Page,
1946: (4)                rect: typing.Union[rect_like, quad_like],
1947: (4)                color: OptSeq = (0,),
1948: (4)                fill: OptSeq = None,
1949: (4)                dashes: OptStr = None,
1950: (4)                morph: OptSeq = None,
1951: (4)                width: float = 1,
1952: (4)                lineCap: int = 0,
1953: (4)                lineJoin: int = 0,
1954: (4)                overlay: bool = True,
1955: (4)                stroke_opacity: float = 1,
1956: (4)                fill_opacity: float = 1,
1957: (4)                oc: int = 0,
1958: (0)            ) -> pymupdf.Point:
1959: (4)                """Draw an oval given its containing rectangle or quad."""
1960: (4)                img = page.new_shape()
1961: (4)                Q = img.draw_oval(rect)
1962: (4)                img.finish(
1963: (8)                    color=color,
```

```
1964: (8)                   fill=fill,
1965: (8)                   dashes=dashes,
1966: (8)                   width=width,
1967: (8)                   lineCap=lineCap,
1968: (8)                   lineJoin=lineJoin,
1969: (8)                   morph=morph,
1970: (8)                   stroke_opacity=stroke_opacity,
1971: (8)                   fill_opacity=fill_opacity,
1972: (8)                   oc=oc,
1973: (4)               )
1974: (4)               img.commit(overlay)
1975: (4)               return Q
1976: (0)           def draw_curve(
1977: (4)               page: pymupdf.Page,
1978: (4)               p1: point_like,
1979: (4)               p2: point_like,
1980: (4)               p3: point_like,
1981: (4)               color: OptSeq = (0,),
1982: (4)               fill: OptSeq = None,
1983: (4)               dashes: OptStr = None,
1984: (4)               width: float = 1,
1985: (4)               morph: OptSeq = None,
1986: (4)               closePath: bool = False,
1987: (4)               lineCap: int = 0,
1988: (4)               lineJoin: int = 0,
1989: (4)               overlay: bool = True,
1990: (4)               stroke_opacity: float = 1,
1991: (4)               fill_opacity: float = 1,
1992: (4)               oc: int = 0,
1993: (0)           ) -> pymupdf.Point:
1994: (4)               """Draw a special Bezier curve from p1 to p3, generating control points on
lines p1 to p2 and p2 to p3."""
1995: (4)               img = page.new_shape()
1996: (4)               Q = img.draw_curve(pymupdf.Point(p1), pymupdf.Point(p2),
pymupdf.Point(p3))
1997: (4)               img.finish(
1998: (8)                   color=color,
1999: (8)                   fill=fill,
2000: (8)                   dashes=dashes,
2001: (8)                   width=width,
2002: (8)                   lineCap=lineCap,
2003: (8)                   lineJoin=lineJoin,
2004: (8)                   morph=morph,
2005: (8)                   closePath=closePath,
2006: (8)                   stroke_opacity=stroke_opacity,
2007: (8)                   fill_opacity=fill_opacity,
2008: (8)                   oc=oc,
2009: (4)               )
2010: (4)               img.commit(overlay)
2011: (4)               return Q
2012: (0)           def draw_bezier(
2013: (4)               page: pymupdf.Page,
2014: (4)               p1: point_like,
2015: (4)               p2: point_like,
2016: (4)               p3: point_like,
2017: (4)               p4: point_like,
2018: (4)               color: OptSeq = (0,),
2019: (4)               fill: OptSeq = None,
2020: (4)               dashes: OptStr = None,
2021: (4)               width: float = 1,
2022: (4)               morph: OptStr = None,
2023: (4)               closePath: bool = False,
2024: (4)               lineCap: int = 0,
2025: (4)               lineJoin: int = 0,
2026: (4)               overlay: bool = True,
2027: (4)               stroke_opacity: float = 1,
2028: (4)               fill_opacity: float = 1,
2029: (4)               oc: int = 0,
2030: (0)           ) -> pymupdf.Point:
```

```
2031: (4)               """Draw a general cubic Bezier curve from p1 to p4 using control points p2
and p3."""
2032: (4)               img = page.new_shape()
2033: (4)               Q = img.draw_bezier(pymupdf.Point(p1), pymupdf.Point(p2),
pymupdf.Point(p3), pymupdf.Point(p4))
2034: (4)               img.finish(
2035: (8)                   color=color,
2036: (8)                   fill=fill,
2037: (8)                   dashes=dashes,
2038: (8)                   width=width,
2039: (8)                   lineCap=lineCap,
2040: (8)                   lineJoin=lineJoin,
2041: (8)                   morph=morph,
2042: (8)                   closePath=closePath,
2043: (8)                   stroke_opacity=stroke_opacity,
2044: (8)                   fill_opacity=fill_opacity,
2045: (8)                   oc=oc,
2046: (4)               )
2047: (4)               img.commit(overlay)
2048: (4)               return Q
2049: (0)           def draw_sector(
2050: (4)               page: pymupdf.Page,
2051: (4)               center: point_like,
2052: (4)               point: point_like,
2053: (4)               beta: float,
2054: (4)               color: OptSeq = (0,),
2055: (4)               fill: OptSeq = None,
2056: (4)               dashes: OptStr = None,
2057: (4)               fullSector: bool = True,
2058: (4)               morph: OptSeq = None,
2059: (4)               width: float = 1,
2060: (4)               closePath: bool = False,
2061: (4)               lineCap: int = 0,
2062: (4)               lineJoin: int = 0,
2063: (4)               overlay: bool = True,
2064: (4)               stroke_opacity: float = 1,
2065: (4)               fill_opacity: float = 1,
2066: (4)               oc: int = 0,
2067: (0)           ) -> pymupdf.Point:
2068: (4)               """Draw a circle sector given circle center, one arc end point and the
angle of the arc.
2069: (4)               Parameters:
2070: (8)                   center -- center of circle
2071: (8)                   point -- arc end point
2072: (8)                   beta -- angle of arc (degrees)
2073: (8)                   fullSector -- connect arc ends with center
2074: (4)               """
2075: (4)               img = page.new_shape()
2076: (4)               Q = img.draw_sector(pymupdf.Point(center), pymupdf.Point(point), beta,
fullSector=fullSector)
2077: (4)               img.finish(
2078: (8)                   color=color,
2079: (8)                   fill=fill,
2080: (8)                   dashes=dashes,
2081: (8)                   width=width,
2082: (8)                   lineCap=lineCap,
2083: (8)                   lineJoin=lineJoin,
2084: (8)                   morph=morph,
2085: (8)                   closePath=closePath,
2086: (8)                   stroke_opacity=stroke_opacity,
2087: (8)                   fill_opacity=fill_opacity,
2088: (8)                   oc=oc,
2089: (4)               )
2090: (4)               img.commit(overlay)
2091: (4)               return Q
2092: (0)           def getColorList() -> list:
2093: (4)               """
2094: (4)               Returns a list of just the colour names used by this module.
2095: (4)               :rtype: list of strings
```

```
2096: (4)                """
2097: (4)                return [x[0] for x in getColorInfoList()]
2098: (0)            def getColorInfoList() -> list:
2099: (4)                """
2100: (4)                Returns the list of colour name/value tuples used by this module.
2101: (4)                :rtype: list of tuples
2102: (4)                """
2103: (4)                return [
2104: (8)                    ("ALICEBLUE", 240, 248, 255),
2105: (8)                    ("ANTIQUEWHITE", 250, 235, 215),
2106: (8)                    ("ANTIQUEWHITE1", 255, 239, 219),
2107: (8)                    ("ANTIQUEWHITE2", 238, 223, 204),
2108: (8)                    ("ANTIQUEWHITE3", 205, 192, 176),
2109: (8)                    ("ANTIQUEWHITE4", 139, 131, 120),
2110: (8)                    ("AQUAMARINE", 127, 255, 212),
2111: (8)                    ("AQUAMARINE1", 127, 255, 212),
2112: (8)                    ("AQUAMARINE2", 118, 238, 198),
2113: (8)                    ("AQUAMARINE3", 102, 205, 170),
2114: (8)                    ("AQUAMARINE4", 69, 139, 116),
2115: (8)                    ("AZURE", 240, 255, 255),
2116: (8)                    ("AZURE1", 240, 255, 255),
2117: (8)                    ("AZURE2", 224, 238, 238),
2118: (8)                    ("AZURE3", 193, 205, 205),
2119: (8)                    ("AZURE4", 131, 139, 139),
2120: (8)                    ("BEIGE", 245, 245, 220),
2121: (8)                    ("BISQUE", 255, 228, 196),
2122: (8)                    ("BISQUE1", 255, 228, 196),
2123: (8)                    ("BISQUE2", 238, 213, 183),
2124: (8)                    ("BISQUE3", 205, 183, 158),
2125: (8)                    ("BISQUE4", 139, 125, 107),
2126: (8)                    ("BLACK", 0, 0, 0),
2127: (8)                    ("BLANCHEDALMOND", 255, 235, 205),
2128: (8)                    ("BLUE", 0, 0, 255),
2129: (8)                    ("BLUE1", 0, 0, 255),
2130: (8)                    ("BLUE2", 0, 0, 238),
2131: (8)                    ("BLUE3", 0, 0, 205),
2132: (8)                    ("BLUE4", 0, 0, 139),
2133: (8)                    ("BLUEVIOLET", 138, 43, 226),
2134: (8)                    ("BROWN", 165, 42, 42),
2135: (8)                    ("BROWN1", 255, 64, 64),
2136: (8)                    ("BROWN2", 238, 59, 59),
2137: (8)                    ("BROWN3", 205, 51, 51),
2138: (8)                    ("BROWN4", 139, 35, 35),
2139: (8)                    ("BURLYWOOD", 222, 184, 135),
2140: (8)                    ("BURLYWOOD1", 255, 211, 155),
2141: (8)                    ("BURLYWOOD2", 238, 197, 145),
2142: (8)                    ("BURLYWOOD3", 205, 170, 125),
2143: (8)                    ("BURLYWOOD4", 139, 115, 85),
2144: (8)                    ("CADETBLUE", 95, 158, 160),
2145: (8)                    ("CADETBLUE1", 152, 245, 255),
2146: (8)                    ("CADETBLUE2", 142, 229, 238),
2147: (8)                    ("CADETBLUE3", 122, 197, 205),
2148: (8)                    ("CADETBLUE4", 83, 134, 139),
2149: (8)                    ("CHARTREUSE", 127, 255, 0),
2150: (8)                    ("CHARTREUSE1", 127, 255, 0),
2151: (8)                    ("CHARTREUSE2", 118, 238, 0),
2152: (8)                    ("CHARTREUSE3", 102, 205, 0),
2153: (8)                    ("CHARTREUSE4", 69, 139, 0),
2154: (8)                    ("CHOCOLATE", 210, 105, 30),
2155: (8)                    ("CHOCOLATE1", 255, 127, 36),
2156: (8)                    ("CHOCOLATE2", 238, 118, 33),
2157: (8)                    ("CHOCOLATE3", 205, 102, 29),
2158: (8)                    ("CHOCOLATE4", 139, 69, 19),
2159: (8)                    ("COFFEE", 156, 79, 0),
2160: (8)                    ("CORAL", 255, 127, 80),
2161: (8)                    ("CORAL1", 255, 114, 86),
2162: (8)                    ("CORAL2", 238, 106, 80),
2163: (8)                    ("CORAL3", 205, 91, 69),
2164: (8)                    ("CORAL4", 139, 62, 47),
```

```
(8)                     ("CORNFLOWERBLUE", 100, 149, 237),
(8)                     ("CORNSILK", 255, 248, 220),
(8)                     ("CORNSILK1", 255, 248, 220),
(8)                     ("CORNSILK2", 238, 232, 205),
(8)                     ("CORNSILK3", 205, 200, 177),
(8)                     ("CORNSILK4", 139, 136, 120),
(8)                     ("CYAN", 0, 255, 255),
(8)                     ("CYAN1", 0, 255, 255),
(8)                     ("CYAN2", 0, 238, 238),
(8)                     ("CYAN3", 0, 205, 205),
(8)                     ("CYAN4", 0, 139, 139),
(8)                     ("DARKBLUE", 0, 0, 139),
(8)                     ("DARKCYAN", 0, 139, 139),
(8)                     ("DARKGOLDENROD", 184, 134, 11),
(8)                     ("DARKGOLDENROD1", 255, 185, 15),
(8)                     ("DARKGOLDENROD2", 238, 173, 14),
(8)                     ("DARKGOLDENROD3", 205, 149, 12),
(8)                     ("DARKGOLDENROD4", 139, 101, 8),
(8)                     ("DARKGREEN", 0, 100, 0),
(8)                     ("DARKGRAY", 169, 169, 169),
(8)                     ("DARKKHAKI", 189, 183, 107),
(8)                     ("DARKMAGENTA", 139, 0, 139),
(8)                     ("DARKOLIVEGREEN", 85, 107, 47),
(8)                     ("DARKOLIVEGREEN1", 202, 255, 112),
(8)                     ("DARKOLIVEGREEN2", 188, 238, 104),
(8)                     ("DARKOLIVEGREEN3", 162, 205, 90),
(8)                     ("DARKOLIVEGREEN4", 110, 139, 61),
(8)                     ("DARKORANGE", 255, 140, 0),
(8)                     ("DARKORANGE1", 255, 127, 0),
(8)                     ("DARKORANGE2", 238, 118, 0),
(8)                     ("DARKORANGE3", 205, 102, 0),
(8)                     ("DARKORANGE4", 139, 69, 0),
(8)                     ("DARKORCHID", 153, 50, 204),
(8)                     ("DARKORCHID1", 191, 62, 255),
(8)                     ("DARKORCHID2", 178, 58, 238),
(8)                     ("DARKORCHID3", 154, 50, 205),
(8)                     ("DARKORCHID4", 104, 34, 139),
(8)                     ("DARKRED", 139, 0, 0),
(8)                     ("DARKSALMON", 233, 150, 122),
(8)                     ("DARKSEAGREEN", 143, 188, 143),
(8)                     ("DARKSEAGREEN1", 193, 255, 193),
(8)                     ("DARKSEAGREEN2", 180, 238, 180),
(8)                     ("DARKSEAGREEN3", 155, 205, 155),
(8)                     ("DARKSEAGREEN4", 105, 139, 105),
(8)                     ("DARKSLATEBLUE", 72, 61, 139),
(8)                     ("DARKSLATEGRAY", 47, 79, 79),
(8)                     ("DARKTURQUOISE", 0, 206, 209),
(8)                     ("DARKVIOLET", 148, 0, 211),
(8)                     ("DEEPPINK", 255, 20, 147),
(8)                     ("DEEPPINK1", 255, 20, 147),
(8)                     ("DEEPPINK2", 238, 18, 137),
(8)                     ("DEEPPINK3", 205, 16, 118),
(8)                     ("DEEPPINK4", 139, 10, 80),
(8)                     ("DEEPSKYBLUE", 0, 191, 255),
(8)                     ("DEEPSKYBLUE1", 0, 191, 255),
(8)                     ("DEEPSKYBLUE2", 0, 178, 238),
(8)                     ("DEEPSKYBLUE3", 0, 154, 205),
(8)                     ("DEEPSKYBLUE4", 0, 104, 139),
(8)                     ("DIMGRAY", 105, 105, 105),
(8)                     ("DODGERBLUE", 30, 144, 255),
(8)                     ("DODGERBLUE1", 30, 144, 255),
(8)                     ("DODGERBLUE2", 28, 134, 238),
(8)                     ("DODGERBLUE3", 24, 116, 205),
(8)                     ("DODGERBLUE4", 16, 78, 139),
(8)                     ("FIREBRICK", 178, 34, 34),
(8)                     ("FIREBRICK1", 255, 48, 48),
(8)                     ("FIREBRICK2", 238, 44, 44),
(8)                     ("FIREBRICK3", 205, 38, 38),
(8)                     ("FIREBRICK4", 139, 26, 26),
```

```
(8)                     ("FLORALWHITE", 255, 250, 240),
(8)                     ("FORESTGREEN", 34, 139, 34),
(8)                     ("GAINSBORO", 220, 220, 220),
(8)                     ("GHOSTWHITE", 248, 248, 255),
(8)                     ("GOLD", 255, 215, 0),
(8)                     ("GOLD1", 255, 215, 0),
(8)                     ("GOLD2", 238, 201, 0),
(8)                     ("GOLD3", 205, 173, 0),
(8)                     ("GOLD4", 139, 117, 0),
(8)                     ("GOLDENROD", 218, 165, 32),
(8)                     ("GOLDENROD1", 255, 193, 37),
(8)                     ("GOLDENROD2", 238, 180, 34),
(8)                     ("GOLDENROD3", 205, 155, 29),
(8)                     ("GOLDENROD4", 139, 105, 20),
(8)                     ("GREEN YELLOW", 173, 255, 47),
(8)                     ("GREEN", 0, 255, 0),
(8)                     ("GREEN1", 0, 255, 0),
(8)                     ("GREEN2", 0, 238, 0),
(8)                     ("GREEN3", 0, 205, 0),
(8)                     ("GREEN4", 0, 139, 0),
(8)                     ("GREENYELLOW", 173, 255, 47),
(8)                     ("GRAY", 190, 190, 190),
(8)                     ("GRAY0", 0, 0, 0),
(8)                     ("GRAY1", 3, 3, 3),
(8)                     ("GRAY10", 26, 26, 26),
(8)                     ("GRAY100", 255, 255, 255),
(8)                     ("GRAY11", 28, 28, 28),
(8)                     ("GRAY12", 31, 31, 31),
(8)                     ("GRAY13", 33, 33, 33),
(8)                     ("GRAY14", 36, 36, 36),
(8)                     ("GRAY15", 38, 38, 38),
(8)                     ("GRAY16", 41, 41, 41),
(8)                     ("GRAY17", 43, 43, 43),
(8)                     ("GRAY18", 46, 46, 46),
(8)                     ("GRAY19", 48, 48, 48),
(8)                     ("GRAY2", 5, 5, 5),
(8)                     ("GRAY20", 51, 51, 51),
(8)                     ("GRAY21", 54, 54, 54),
(8)                     ("GRAY22", 56, 56, 56),
(8)                     ("GRAY23", 59, 59, 59),
(8)                     ("GRAY24", 61, 61, 61),
(8)                     ("GRAY25", 64, 64, 64),
(8)                     ("GRAY26", 66, 66, 66),
(8)                     ("GRAY27", 69, 69, 69),
(8)                     ("GRAY28", 71, 71, 71),
(8)                     ("GRAY29", 74, 74, 74),
(8)                     ("GRAY3", 8, 8, 8),
(8)                     ("GRAY30", 77, 77, 77),
(8)                     ("GRAY31", 79, 79, 79),
(8)                     ("GRAY32", 82, 82, 82),
(8)                     ("GRAY33", 84, 84, 84),
(8)                     ("GRAY34", 87, 87, 87),
(8)                     ("GRAY35", 89, 89, 89),
(8)                     ("GRAY36", 92, 92, 92),
(8)                     ("GRAY37", 94, 94, 94),
(8)                     ("GRAY38", 97, 97, 97),
(8)                     ("GRAY39", 99, 99, 99),
(8)                     ("GRAY4", 10, 10, 10),
(8)                     ("GRAY40", 102, 102, 102),
(8)                     ("GRAY41", 105, 105, 105),
(8)                     ("GRAY42", 107, 107, 107),
(8)                     ("GRAY43", 110, 110, 110),
(8)                     ("GRAY44", 112, 112, 112),
(8)                     ("GRAY45", 115, 115, 115),
(8)                     ("GRAY46", 117, 117, 117),
(8)                     ("GRAY47", 120, 120, 120),
(8)                     ("GRAY48", 122, 122, 122),
(8)                     ("GRAY49", 125, 125, 125),
(8)                     ("GRAY5", 13, 13, 13),
```

```
(8)                     ("GRAY50", 127, 127, 127),
(8)                     ("GRAY51", 130, 130, 130),
(8)                     ("GRAY52", 133, 133, 133),
(8)                     ("GRAY53", 135, 135, 135),
(8)                     ("GRAY54", 138, 138, 138),
(8)                     ("GRAY55", 140, 140, 140),
(8)                     ("GRAY56", 143, 143, 143),
(8)                     ("GRAY57", 145, 145, 145),
(8)                     ("GRAY58", 148, 148, 148),
(8)                     ("GRAY59", 150, 150, 150),
(8)                     ("GRAY6", 15, 15, 15),
(8)                     ("GRAY60", 153, 153, 153),
(8)                     ("GRAY61", 156, 156, 156),
(8)                     ("GRAY62", 158, 158, 158),
(8)                     ("GRAY63", 161, 161, 161),
(8)                     ("GRAY64", 163, 163, 163),
(8)                     ("GRAY65", 166, 166, 166),
(8)                     ("GRAY66", 168, 168, 168),
(8)                     ("GRAY67", 171, 171, 171),
(8)                     ("GRAY68", 173, 173, 173),
(8)                     ("GRAY69", 176, 176, 176),
(8)                     ("GRAY7", 18, 18, 18),
(8)                     ("GRAY70", 179, 179, 179),
(8)                     ("GRAY71", 181, 181, 181),
(8)                     ("GRAY72", 184, 184, 184),
(8)                     ("GRAY73", 186, 186, 186),
(8)                     ("GRAY74", 189, 189, 189),
(8)                     ("GRAY75", 191, 191, 191),
(8)                     ("GRAY76", 194, 194, 194),
(8)                     ("GRAY77", 196, 196, 196),
(8)                     ("GRAY78", 199, 199, 199),
(8)                     ("GRAY79", 201, 201, 201),
(8)                     ("GRAY8", 20, 20, 20),
(8)                     ("GRAY80", 204, 204, 204),
(8)                     ("GRAY81", 207, 207, 207),
(8)                     ("GRAY82", 209, 209, 209),
(8)                     ("GRAY83", 212, 212, 212),
(8)                     ("GRAY84", 214, 214, 214),
(8)                     ("GRAY85", 217, 217, 217),
(8)                     ("GRAY86", 219, 219, 219),
(8)                     ("GRAY87", 222, 222, 222),
(8)                     ("GRAY88", 224, 224, 224),
(8)                     ("GRAY89", 227, 227, 227),
(8)                     ("GRAY9", 23, 23, 23),
(8)                     ("GRAY90", 229, 229, 229),
(8)                     ("GRAY91", 232, 232, 232),
(8)                     ("GRAY92", 235, 235, 235),
(8)                     ("GRAY93", 237, 237, 237),
(8)                     ("GRAY94", 240, 240, 240),
(8)                     ("GRAY95", 242, 242, 242),
(8)                     ("GRAY96", 245, 245, 245),
(8)                     ("GRAY97", 247, 247, 247),
(8)                     ("GRAY98", 250, 250, 250),
(8)                     ("GRAY99", 252, 252, 252),
(8)                     ("HONEYDEW", 240, 255, 240),
(8)                     ("HONEYDEW1", 240, 255, 240),
(8)                     ("HONEYDEW2", 224, 238, 224),
(8)                     ("HONEYDEW3", 193, 205, 193),
(8)                     ("HONEYDEW4", 131, 139, 131),
(8)                     ("HOTPINK", 255, 105, 180),
(8)                     ("HOTPINK1", 255, 110, 180),
(8)                     ("HOTPINK2", 238, 106, 167),
(8)                     ("HOTPINK3", 205, 96, 144),
(8)                     ("HOTPINK4", 139, 58, 98),
(8)                     ("INDIANRED", 205, 92, 92),
(8)                     ("INDIANRED1", 255, 106, 106),
(8)                     ("INDIANRED2", 238, 99, 99),
(8)                     ("INDIANRED3", 205, 85, 85),
(8)                     ("INDIANRED4", 139, 58, 58),
```

```
(8)                     ("IVORY", 255, 255, 240),
(8)                     ("IVORY1", 255, 255, 240),
(8)                     ("IVORY2", 238, 238, 224),
(8)                     ("IVORY3", 205, 205, 193),
(8)                     ("IVORY4", 139, 139, 131),
(8)                     ("KHAKI", 240, 230, 140),
(8)                     ("KHAKI1", 255, 246, 143),
(8)                     ("KHAKI2", 238, 230, 133),
(8)                     ("KHAKI3", 205, 198, 115),
(8)                     ("KHAKI4", 139, 134, 78),
(8)                     ("LAVENDER", 230, 230, 250),
(8)                     ("LAVENDERBLUSH", 255, 240, 245),
(8)                     ("LAVENDERBLUSH1", 255, 240, 245),
(8)                     ("LAVENDERBLUSH2", 238, 224, 229),
(8)                     ("LAVENDERBLUSH3", 205, 193, 197),
(8)                     ("LAVENDERBLUSH4", 139, 131, 134),
(8)                     ("LAWNGREEN", 124, 252, 0),
(8)                     ("LEMONCHIFFON", 255, 250, 205),
(8)                     ("LEMONCHIFFON1", 255, 250, 205),
(8)                     ("LEMONCHIFFON2", 238, 233, 191),
(8)                     ("LEMONCHIFFON3", 205, 201, 165),
(8)                     ("LEMONCHIFFON4", 139, 137, 112),
(8)                     ("LIGHTBLUE", 173, 216, 230),
(8)                     ("LIGHTBLUE1", 191, 239, 255),
(8)                     ("LIGHTBLUE2", 178, 223, 238),
(8)                     ("LIGHTBLUE3", 154, 192, 205),
(8)                     ("LIGHTBLUE4", 104, 131, 139),
(8)                     ("LIGHTCORAL", 240, 128, 128),
(8)                     ("LIGHTCYAN", 224, 255, 255),
(8)                     ("LIGHTCYAN1", 224, 255, 255),
(8)                     ("LIGHTCYAN2", 209, 238, 238),
(8)                     ("LIGHTCYAN3", 180, 205, 205),
(8)                     ("LIGHTCYAN4", 122, 139, 139),
(8)                     ("LIGHTGOLDENROD", 238, 221, 130),
(8)                     ("LIGHTGOLDENROD1", 255, 236, 139),
(8)                     ("LIGHTGOLDENROD2", 238, 220, 130),
(8)                     ("LIGHTGOLDENROD3", 205, 190, 112),
(8)                     ("LIGHTGOLDENROD4", 139, 129, 76),
(8)                     ("LIGHTGOLDENRODYELLOW", 250, 250, 210),
(8)                     ("LIGHTGREEN", 144, 238, 144),
(8)                     ("LIGHTGRAY", 211, 211, 211),
(8)                     ("LIGHTPINK", 255, 182, 193),
(8)                     ("LIGHTPINK1", 255, 174, 185),
(8)                     ("LIGHTPINK2", 238, 162, 173),
(8)                     ("LIGHTPINK3", 205, 140, 149),
(8)                     ("LIGHTPINK4", 139, 95, 101),
(8)                     ("LIGHTSALMON", 255, 160, 122),
(8)                     ("LIGHTSALMON1", 255, 160, 122),
(8)                     ("LIGHTSALMON2", 238, 149, 114),
(8)                     ("LIGHTSALMON3", 205, 129, 98),
(8)                     ("LIGHTSALMON4", 139, 87, 66),
(8)                     ("LIGHTSEAGREEN", 32, 178, 170),
(8)                     ("LIGHTSKYBLUE", 135, 206, 250),
(8)                     ("LIGHTSKYBLUE1", 176, 226, 255),
(8)                     ("LIGHTSKYBLUE2", 164, 211, 238),
(8)                     ("LIGHTSKYBLUE3", 141, 182, 205),
(8)                     ("LIGHTSKYBLUE4", 96, 123, 139),
(8)                     ("LIGHTSLATEBLUE", 132, 112, 255),
(8)                     ("LIGHTSLATEGRAY", 119, 136, 153),
(8)                     ("LIGHTSTEELBLUE", 176, 196, 222),
(8)                     ("LIGHTSTEELBLUE1", 202, 225, 255),
(8)                     ("LIGHTSTEELBLUE2", 188, 210, 238),
(8)                     ("LIGHTSTEELBLUE3", 162, 181, 205),
(8)                     ("LIGHTSTEELBLUE4", 110, 123, 139),
(8)                     ("LIGHTYELLOW", 255, 255, 224),
(8)                     ("LIGHTYELLOW1", 255, 255, 224),
(8)                     ("LIGHTYELLOW2", 238, 238, 209),
(8)                     ("LIGHTYELLOW3", 205, 205, 180),
(8)                     ("LIGHTYELLOW4", 139, 139, 122),
```

```
2441: (8)                    ("LIMEGREEN", 50, 205, 50),
2442: (8)                    ("LINEN", 250, 240, 230),
2443: (8)                    ("MAGENTA", 255, 0, 255),
2444: (8)                    ("MAGENTA1", 255, 0, 255),
2445: (8)                    ("MAGENTA2", 238, 0, 238),
2446: (8)                    ("MAGENTA3", 205, 0, 205),
2447: (8)                    ("MAGENTA4", 139, 0, 139),
2448: (8)                    ("MAROON", 176, 48, 96),
2449: (8)                    ("MAROON1", 255, 52, 179),
2450: (8)                    ("MAROON2", 238, 48, 167),
2451: (8)                    ("MAROON3", 205, 41, 144),
2452: (8)                    ("MAROON4", 139, 28, 98),
2453: (8)                    ("MEDIUMAQUAMARINE", 102, 205, 170),
2454: (8)                    ("MEDIUMBLUE", 0, 0, 205),
2455: (8)                    ("MEDIUMORCHID", 186, 85, 211),
2456: (8)                    ("MEDIUMORCHID1", 224, 102, 255),
2457: (8)                    ("MEDIUMORCHID2", 209, 95, 238),
2458: (8)                    ("MEDIUMORCHID3", 180, 82, 205),
2459: (8)                    ("MEDIUMORCHID4", 122, 55, 139),
2460: (8)                    ("MEDIUMPURPLE", 147, 112, 219),
2461: (8)                    ("MEDIUMPURPLE1", 171, 130, 255),
2462: (8)                    ("MEDIUMPURPLE2", 159, 121, 238),
2463: (8)                    ("MEDIUMPURPLE3", 137, 104, 205),
2464: (8)                    ("MEDIUMPURPLE4", 93, 71, 139),
2465: (8)                    ("MEDIUMSEAGREEN", 60, 179, 113),
2466: (8)                    ("MEDIUMSLATEBLUE", 123, 104, 238),
2467: (8)                    ("MEDIUMSPRINGGREEN", 0, 250, 154),
2468: (8)                    ("MEDIUMTURQUOISE", 72, 209, 204),
2469: (8)                    ("MEDIUMVIOLETRED", 199, 21, 133),
2470: (8)                    ("MIDNIGHTBLUE", 25, 25, 112),
2471: (8)                    ("MINTCREAM", 245, 255, 250),
2472: (8)                    ("MISTYROSE", 255, 228, 225),
2473: (8)                    ("MISTYROSE1", 255, 228, 225),
2474: (8)                    ("MISTYROSE2", 238, 213, 210),
2475: (8)                    ("MISTYROSE3", 205, 183, 181),
2476: (8)                    ("MISTYROSE4", 139, 125, 123),
2477: (8)                    ("MOCCASIN", 255, 228, 181),
2478: (8)                    ("MUPDFBLUE", 37, 114, 172),
2479: (8)                    ("NAVAJOWHITE", 255, 222, 173),
2480: (8)                    ("NAVAJOWHITE1", 255, 222, 173),
2481: (8)                    ("NAVAJOWHITE2", 238, 207, 161),
2482: (8)                    ("NAVAJOWHITE3", 205, 179, 139),
2483: (8)                    ("NAVAJOWHITE4", 139, 121, 94),
2484: (8)                    ("NAVY", 0, 0, 128),
2485: (8)                    ("NAVYBLUE", 0, 0, 128),
2486: (8)                    ("OLDLACE", 253, 245, 230),
2487: (8)                    ("OLIVEDRAB", 107, 142, 35),
2488: (8)                    ("OLIVEDRAB1", 192, 255, 62),
2489: (8)                    ("OLIVEDRAB2", 179, 238, 58),
2490: (8)                    ("OLIVEDRAB3", 154, 205, 50),
2491: (8)                    ("OLIVEDRAB4", 105, 139, 34),
2492: (8)                    ("ORANGE", 255, 165, 0),
2493: (8)                    ("ORANGE1", 255, 165, 0),
2494: (8)                    ("ORANGE2", 238, 154, 0),
2495: (8)                    ("ORANGE3", 205, 133, 0),
2496: (8)                    ("ORANGE4", 139, 90, 0),
2497: (8)                    ("ORANGERED", 255, 69, 0),
2498: (8)                    ("ORANGERED1", 255, 69, 0),
2499: (8)                    ("ORANGERED2", 238, 64, 0),
2500: (8)                    ("ORANGERED3", 205, 55, 0),
2501: (8)                    ("ORANGERED4", 139, 37, 0),
2502: (8)                    ("ORCHID", 218, 112, 214),
2503: (8)                    ("ORCHID1", 255, 131, 250),
2504: (8)                    ("ORCHID2", 238, 122, 233),
2505: (8)                    ("ORCHID3", 205, 105, 201),
2506: (8)                    ("ORCHID4", 139, 71, 137),
2507: (8)                    ("PALEGOLDENROD", 238, 232, 170),
2508: (8)                    ("PALEGREEN", 152, 251, 152),
2509: (8)                    ("PALEGREEN1", 154, 255, 154),
```

```
2510: (8)                    ("PALEGREEN2", 144, 238, 144),
2511: (8)                    ("PALEGREEN3", 124, 205, 124),
2512: (8)                    ("PALEGREEN4", 84, 139, 84),
2513: (8)                    ("PALETURQUOISE", 175, 238, 238),
2514: (8)                    ("PALETURQUOISE1", 187, 255, 255),
2515: (8)                    ("PALETURQUOISE2", 174, 238, 238),
2516: (8)                    ("PALETURQUOISE3", 150, 205, 205),
2517: (8)                    ("PALETURQUOISE4", 102, 139, 139),
2518: (8)                    ("PALEVIOLETRED", 219, 112, 147),
2519: (8)                    ("PALEVIOLETRED1", 255, 130, 171),
2520: (8)                    ("PALEVIOLETRED2", 238, 121, 159),
2521: (8)                    ("PALEVIOLETRED3", 205, 104, 137),
2522: (8)                    ("PALEVIOLETRED4", 139, 71, 93),
2523: (8)                    ("PAPAYAWHIP", 255, 239, 213),
2524: (8)                    ("PEACHPUFF", 255, 218, 185),
2525: (8)                    ("PEACHPUFF1", 255, 218, 185),
2526: (8)                    ("PEACHPUFF2", 238, 203, 173),
2527: (8)                    ("PEACHPUFF3", 205, 175, 149),
2528: (8)                    ("PEACHPUFF4", 139, 119, 101),
2529: (8)                    ("PERU", 205, 133, 63),
2530: (8)                    ("PINK", 255, 192, 203),
2531: (8)                    ("PINK1", 255, 181, 197),
2532: (8)                    ("PINK2", 238, 169, 184),
2533: (8)                    ("PINK3", 205, 145, 158),
2534: (8)                    ("PINK4", 139, 99, 108),
2535: (8)                    ("PLUM", 221, 160, 221),
2536: (8)                    ("PLUM1", 255, 187, 255),
2537: (8)                    ("PLUM2", 238, 174, 238),
2538: (8)                    ("PLUM3", 205, 150, 205),
2539: (8)                    ("PLUM4", 139, 102, 139),
2540: (8)                    ("POWDERBLUE", 176, 224, 230),
2541: (8)                    ("PURPLE", 160, 32, 240),
2542: (8)                    ("PURPLE1", 155, 48, 255),
2543: (8)                    ("PURPLE2", 145, 44, 238),
2544: (8)                    ("PURPLE3", 125, 38, 205),
2545: (8)                    ("PURPLE4", 85, 26, 139),
2546: (8)                    ("PY_COLOR", 240, 255, 210),
2547: (8)                    ("RED", 255, 0, 0),
2548: (8)                    ("RED1", 255, 0, 0),
2549: (8)                    ("RED2", 238, 0, 0),
2550: (8)                    ("RED3", 205, 0, 0),
2551: (8)                    ("RED4", 139, 0, 0),
2552: (8)                    ("ROSYBROWN", 188, 143, 143),
2553: (8)                    ("ROSYBROWN1", 255, 193, 193),
2554: (8)                    ("ROSYBROWN2", 238, 180, 180),
2555: (8)                    ("ROSYBROWN3", 205, 155, 155),
2556: (8)                    ("ROSYBROWN4", 139, 105, 105),
2557: (8)                    ("ROYALBLUE", 65, 105, 225),
2558: (8)                    ("ROYALBLUE1", 72, 118, 255),
2559: (8)                    ("ROYALBLUE2", 67, 110, 238),
2560: (8)                    ("ROYALBLUE3", 58, 95, 205),
2561: (8)                    ("ROYALBLUE4", 39, 64, 139),
2562: (8)                    ("SADDLEBROWN", 139, 69, 19),
2563: (8)                    ("SALMON", 250, 128, 114),
2564: (8)                    ("SALMON1", 255, 140, 105),
2565: (8)                    ("SALMON2", 238, 130, 98),
2566: (8)                    ("SALMON3", 205, 112, 84),
2567: (8)                    ("SALMON4", 139, 76, 57),
2568: (8)                    ("SANDYBROWN", 244, 164, 96),
2569: (8)                    ("SEAGREEN", 46, 139, 87),
2570: (8)                    ("SEAGREEN1", 84, 255, 159),
2571: (8)                    ("SEAGREEN2", 78, 238, 148),
2572: (8)                    ("SEAGREEN3", 67, 205, 128),
2573: (8)                    ("SEAGREEN4", 46, 139, 87),
2574: (8)                    ("SEASHELL", 255, 245, 238),
2575: (8)                    ("SEASHELL1", 255, 245, 238),
2576: (8)                    ("SEASHELL2", 238, 229, 222),
2577: (8)                    ("SEASHELL3", 205, 197, 191),
2578: (8)                    ("SEASHELL4", 139, 134, 130),
```

```
2579: (8)                    ("SIENNA", 160, 82, 45),
2580: (8)                    ("SIENNA1", 255, 130, 71),
2581: (8)                    ("SIENNA2", 238, 121, 66),
2582: (8)                    ("SIENNA3", 205, 104, 57),
2583: (8)                    ("SIENNA4", 139, 71, 38),
2584: (8)                    ("SKYBLUE", 135, 206, 235),
2585: (8)                    ("SKYBLUE1", 135, 206, 255),
2586: (8)                    ("SKYBLUE2", 126, 192, 238),
2587: (8)                    ("SKYBLUE3", 108, 166, 205),
2588: (8)                    ("SKYBLUE4", 74, 112, 139),
2589: (8)                    ("SLATEBLUE", 106, 90, 205),
2590: (8)                    ("SLATEBLUE1", 131, 111, 255),
2591: (8)                    ("SLATEBLUE2", 122, 103, 238),
2592: (8)                    ("SLATEBLUE3", 105, 89, 205),
2593: (8)                    ("SLATEBLUE4", 71, 60, 139),
2594: (8)                    ("SLATEGRAY", 112, 128, 144),
2595: (8)                    ("SNOW", 255, 250, 250),
2596: (8)                    ("SNOW1", 255, 250, 250),
2597: (8)                    ("SNOW2", 238, 233, 233),
2598: (8)                    ("SNOW3", 205, 201, 201),
2599: (8)                    ("SNOW4", 139, 137, 137),
2600: (8)                    ("SPRINGGREEN", 0, 255, 127),
2601: (8)                    ("SPRINGGREEN1", 0, 255, 127),
2602: (8)                    ("SPRINGGREEN2", 0, 238, 118),
2603: (8)                    ("SPRINGGREEN3", 0, 205, 102),
2604: (8)                    ("SPRINGGREEN4", 0, 139, 69),
2605: (8)                    ("STEELBLUE", 70, 130, 180),
2606: (8)                    ("STEELBLUE1", 99, 184, 255),
2607: (8)                    ("STEELBLUE2", 92, 172, 238),
2608: (8)                    ("STEELBLUE3", 79, 148, 205),
2609: (8)                    ("STEELBLUE4", 54, 100, 139),
2610: (8)                    ("TAN", 210, 180, 140),
2611: (8)                    ("TAN1", 255, 165, 79),
2612: (8)                    ("TAN2", 238, 154, 73),
2613: (8)                    ("TAN3", 205, 133, 63),
2614: (8)                    ("TAN4", 139, 90, 43),
2615: (8)                    ("THISTLE", 216, 191, 216),
2616: (8)                    ("THISTLE1", 255, 225, 255),
2617: (8)                    ("THISTLE2", 238, 210, 238),
2618: (8)                    ("THISTLE3", 205, 181, 205),
2619: (8)                    ("THISTLE4", 139, 123, 139),
2620: (8)                    ("TOMATO", 255, 99, 71),
2621: (8)                    ("TOMATO1", 255, 99, 71),
2622: (8)                    ("TOMATO2", 238, 92, 66),
2623: (8)                    ("TOMATO3", 205, 79, 57),
2624: (8)                    ("TOMATO4", 139, 54, 38),
2625: (8)                    ("TURQUOISE", 64, 224, 208),
2626: (8)                    ("TURQUOISE1", 0, 245, 255),
2627: (8)                    ("TURQUOISE2", 0, 229, 238),
2628: (8)                    ("TURQUOISE3", 0, 197, 205),
2629: (8)                    ("TURQUOISE4", 0, 134, 139),
2630: (8)                    ("VIOLET", 238, 130, 238),
2631: (8)                    ("VIOLETRED", 208, 32, 144),
2632: (8)                    ("VIOLETRED1", 255, 62, 150),
2633: (8)                    ("VIOLETRED2", 238, 58, 140),
2634: (8)                    ("VIOLETRED3", 205, 50, 120),
2635: (8)                    ("VIOLETRED4", 139, 34, 82),
2636: (8)                    ("WHEAT", 245, 222, 179),
2637: (8)                    ("WHEAT1", 255, 231, 186),
2638: (8)                    ("WHEAT2", 238, 216, 174),
2639: (8)                    ("WHEAT3", 205, 186, 150),
2640: (8)                    ("WHEAT4", 139, 126, 102),
2641: (8)                    ("WHITE", 255, 255, 255),
2642: (8)                    ("WHITESMOKE", 245, 245, 245),
2643: (8)                    ("YELLOW", 255, 255, 0),
2644: (8)                    ("YELLOW1", 255, 255, 0),
2645: (8)                    ("YELLOW2", 238, 238, 0),
2646: (8)                    ("YELLOW3", 205, 205, 0),
2647: (8)                    ("YELLOW4", 139, 139, 0),
```

```
(8)                 ("YELLOWGREEN", 154, 205, 50),
(4)             ]
(0)         def getColorInfoDict() -> dict:
(4)             d = {}
(4)             for item in getColorInfoList():
(8)                 d[item[0].lower()] = item[1:]
(4)             return d
(0)         def getColor(name: str) -> tuple:
(4)             """Retrieve RGB color in PDF format by name.
(4)             Returns:
(8)                 a triple of floats in range 0 to 1. In case of name-not-found, "white"
is returned.
(4)             """
(4)             try:
(8)                 c = getColorInfoList()[getColorList().index(name.upper())]
(8)                 return (c[1] / 255.0, c[2] / 255.0, c[3] / 255.0)
(4)             except Exception:
(8)                 pymupdf.exception_info()
(8)                 return (1, 1, 1)
(0)         def getColorHSV(name: str) -> tuple:
(4)             """Retrieve the hue, saturation, value triple of a color name.
(4)             Returns:
(8)                 a triple (degree, percent, percent). If not found (-1, -1, -1) is
returned.
(4)             """
(4)             try:
(8)                 x = getColorInfoList()[getColorList().index(name.upper())]
(4)             except Exception:
(8)                 if g_exceptions_verbose:    pymupdf.exception_info()
(8)                 return (-1, -1, -1)
(4)             r = x[1] / 255.0
(4)             g = x[2] / 255.0
(4)             b = x[3] / 255.0
(4)             cmax = max(r, g, b)
(4)             V = round(cmax * 100, 1)
(4)             cmin = min(r, g, b)
(4)             delta = cmax - cmin
(4)             if delta == 0:
(8)                 hue = 0
(4)             elif cmax == r:
(8)                 hue = 60.0 * (((g - b) / delta) % 6)
(4)             elif cmax == g:
(8)                 hue = 60.0 * (((b - r) / delta) + 2)
(4)             else:
(8)                 hue = 60.0 * (((r - g) / delta) + 4)
(4)             H = int(round(hue))
(4)             if cmax == 0:
(8)                 sat = 0
(4)             else:
(8)                 sat = delta / cmax
(4)             S = int(round(sat * 100))
(4)             return (H, S, V)
(0)         def _get_font_properties(doc: pymupdf.Document, xref: int) -> tuple:
(4)             fontname, ext, stype, buffer = doc.extract_font(xref)
(4)             asc = 0.8
(4)             dsc = -0.2
(4)             if ext == "":
(8)                 return fontname, ext, stype, asc, dsc
(4)             if buffer:
(8)                 try:
(12)                    font = pymupdf.Font(fontbuffer=buffer)
(12)                    asc = font.ascender
(12)                    dsc = font.descender
(12)                    bbox = font.bbox
(12)                    if asc - dsc < 1:
(16)                        if bbox.y0 < dsc:
(20)                            dsc = bbox.y0
(16)                        asc = 1 - dsc
(8)                 except Exception:
```

```
2715: (12)                      pymupdf.exception_info()
2716: (12)                      asc *= 1.2
2717: (12)                      dsc *= 1.2
2718: (8)                   return fontname, ext, stype, asc, dsc
2719: (4)               if ext != "n/a":
2720: (8)                   try:
2721: (12)                      font = pymupdf.Font(fontname)
2722: (12)                      asc = font.ascender
2723: (12)                      dsc = font.descender
2724: (8)                   except Exception:
2725: (12)                      pymupdf.exception_info()
2726: (12)                      asc *= 1.2
2727: (12)                      dsc *= 1.2
2728: (4)               else:
2729: (8)                   asc *= 1.2
2730: (8)                   dsc *= 1.2
2731: (4)               return fontname, ext, stype, asc, dsc
2732: (0)           def get_char_widths(
2733: (4)               doc: pymupdf.Document, xref: int, limit: int = 256, idx: int = 0,
fontdict: OptDict = None
2734: (0)           ) -> list:
2735: (4)               """Get list of glyph information of a font.
2736: (4)               Notes:
2737: (8)                   Must be provided by its XREF number. If we already dealt with the
2738: (8)                   font, it will be recorded in doc.FontInfos. Otherwise we insert an
2739: (8)                   entry there.
2740: (8)                   Finally we return the glyphs for the font. This is a list of
2741: (8)                   (glyph, width) where glyph is an integer controlling the char
2742: (8)                   appearance, and width is a float controlling the char's spacing:
2743: (8)                   width * fontsize is the actual space.
2744: (8)                   For 'simple' fonts, glyph == ord(char) will usually be true.
2745: (8)                   Exceptions are 'Symbol' and 'ZapfDingbats'. We are providing data for
these directly here.
2746: (4)               """
2747: (4)               fontinfo = pymupdf.CheckFontInfo(doc, xref)
2748: (4)               if fontinfo is None:  # not recorded yet: create it
2749: (8)                   if fontdict is None:
2750: (12)                      name, ext, stype, asc, dsc = _get_font_properties(doc, xref)
2751: (12)                      fontdict = {
2752: (16)                          "name": name,
2753: (16)                          "type": stype,
2754: (16)                          "ext": ext,
2755: (16)                          "ascender": asc,
2756: (16)                          "descender": dsc,
2757: (12)                      }
2758: (8)                   else:
2759: (12)                      name = fontdict["name"]
2760: (12)                      ext = fontdict["ext"]
2761: (12)                      stype = fontdict["type"]
2762: (12)                      ordering = fontdict["ordering"]
2763: (12)                      simple = fontdict["simple"]
2764: (8)                   if ext == "":
2765: (12)                      raise ValueError("xref is not a font")
2766: (8)                   if stype in ("Type1", "MMType1", "TrueType"):
2767: (12)                      simple = True
2768: (8)                   else:
2769: (12)                      simple = False
2770: (8)                   if name in ("Fangti", "Ming"):
2771: (12)                      ordering = 0
2772: (8)                   elif name in ("Heiti", "Song"):
2773: (12)                      ordering = 1
2774: (8)                   elif name in ("Gothic", "Mincho"):
2775: (12)                      ordering = 2
2776: (8)                   elif name in ("Dotum", "Batang"):
2777: (12)                      ordering = 3
2778: (8)                   else:
2779: (12)                      ordering = -1
2780: (8)                   fontdict["simple"] = simple
2781: (8)                   if name == "ZapfDingbats":
```

```
2782: (12)                      glyphs = pymupdf.zapf_glyphs
2783: (8)                   elif name == "Symbol":
2784: (12)                      glyphs = pymupdf.symbol_glyphs
2785: (8)                   else:
2786: (12)                      glyphs = None
2787: (8)                   fontdict["glyphs"] = glyphs
2788: (8)                   fontdict["ordering"] = ordering
2789: (8)                   fontinfo = [xref, fontdict]
2790: (8)                   doc.FontInfos.append(fontinfo)
2791: (4)               else:
2792: (8)                   fontdict = fontinfo[1]
2793: (8)                   glyphs = fontdict["glyphs"]
2794: (8)                   simple = fontdict["simple"]
2795: (8)                   ordering = fontdict["ordering"]
2796: (4)               if glyphs is None:
2797: (8)                   oldlimit = 0
2798: (4)               else:
2799: (8)                   oldlimit = len(glyphs)
2800: (4)               mylimit = max(256, limit)
2801: (4)               if mylimit <= oldlimit:
2802: (8)                   return glyphs
2803: (4)               if ordering < 0:  # not a CJK font
2804: (8)                   glyphs = doc._get_char_widths(
2805: (12)                      xref, fontdict["name"], fontdict["ext"], fontdict["ordering"],
mylimit, idx
2806: (8)                   )
2807: (4)               else:  # CJK fonts use char codes and width = 1
2808: (8)                   glyphs = None
2809: (4)               fontdict["glyphs"] = glyphs
2810: (4)               fontinfo[1] = fontdict
2811: (4)               pymupdf.UpdateFontInfo(doc, fontinfo)
2812: (4)               return glyphs
2813: (0)           class Shape:
2814: (4)               """Create a new shape."""
2815: (4)               @staticmethod
2816: (4)               def horizontal_angle(C, P):
2817: (8)                   """Return the angle to the horizontal for the connection from C to P.
2818: (8)                   This uses the arcus sine function and resolves its inherent ambiguity
by
2819: (8)                   looking up in which quadrant vector S = P - C is located.
2820: (8)                   """
2821: (8)                   S = pymupdf.Point(P - C).unit  # unit vector 'C' -> 'P'
2822: (8)                   alfa = math.asin(abs(S.y))  # absolute angle from horizontal
2823: (8)                   if S.x < 0:  # make arcsin result unique
2824: (12)                      if S.y <= 0:  # bottom-left
2825: (16)                          alfa = -(math.pi - alfa)
2826: (12)                      else:  # top-left
2827: (16)                          alfa = math.pi - alfa
2828: (8)                   else:
2829: (12)                      if S.y >= 0:  # top-right
2830: (16)                          pass
2831: (12)                      else:  # bottom-right
2832: (16)                          alfa = -alfa
2833: (8)                   return alfa
2834: (4)               def __init__(self, page: pymupdf.Page):
2835: (8)                   pymupdf.CheckParent(page)
2836: (8)                   self.page = page
2837: (8)                   self.doc = page.parent
2838: (8)                   if not self.doc.is_pdf:
2839: (12)                      raise ValueError("is no PDF")
2840: (8)                   self.height = page.mediabox_size.y
2841: (8)                   self.width = page.mediabox_size.x
2842: (8)                   self.x = page.cropbox_position.x
2843: (8)                   self.y = page.cropbox_position.y
2844: (8)                   self.pctm = page.transformation_matrix  # page transf. matrix
2845: (8)                   self.ipctm = ~self.pctm  # inverted transf. matrix
2846: (8)                   self.draw_cont = ""
2847: (8)                   self.text_cont = ""
2848: (8)                   self.totalcont = ""
```

```
2849: (8)                    self.last_point = None
2850: (8)                    self.rect = None
2851: (4)                def updateRect(self, x):
2852: (8)                    if self.rect is None:
2853: (12)                       if len(x) == 2:
2854: (16)                           self.rect = pymupdf.Rect(x, x)
2855: (12)                       else:
2856: (16)                           self.rect = pymupdf.Rect(x)
2857: (8)                    else:
2858: (12)                       if len(x) == 2:
2859: (16)                           x = pymupdf.Point(x)
2860: (16)                           self.rect.x0 = min(self.rect.x0, x.x)
2861: (16)                           self.rect.y0 = min(self.rect.y0, x.y)
2862: (16)                           self.rect.x1 = max(self.rect.x1, x.x)
2863: (16)                           self.rect.y1 = max(self.rect.y1, x.y)
2864: (12)                       else:
2865: (16)                           x = pymupdf.Rect(x)
2866: (16)                           self.rect.x0 = min(self.rect.x0, x.x0)
2867: (16)                           self.rect.y0 = min(self.rect.y0, x.y0)
2868: (16)                           self.rect.x1 = max(self.rect.x1, x.x1)
2869: (16)                           self.rect.y1 = max(self.rect.y1, x.y1)
2870: (4)                def draw_line(self, p1: point_like, p2: point_like) -> pymupdf.Point:
2871: (8)                    """Draw a line between two points."""
2872: (8)                    p1 = pymupdf.Point(p1)
2873: (8)                    p2 = pymupdf.Point(p2)
2874: (8)                    if not (self.last_point == p1):
2875: (12)                       self.draw_cont += _format_g(pymupdf.JM_TUPLE(p1 * self.ipctm)) + "
m\n"
2876: (12)                       self.last_point = p1
2877: (12)                       self.updateRect(p1)
2878: (8)                    self.draw_cont += _format_g(pymupdf.JM_TUPLE(p2 * self.ipctm)) + "
l\n"
2879: (8)                    self.updateRect(p2)
2880: (8)                    self.last_point = p2
2881: (8)                    return self.last_point
2882: (4)                def draw_polyline(self, points: list) -> pymupdf.Point:
2883: (8)                    """Draw several connected line segments."""
2884: (8)                    for i, p in enumerate(points):
2885: (12)                       if i == 0:
2886: (16)                           if not (self.last_point == pymupdf.Point(p)):
2887: (20)                               self.draw_cont +=
_format_g(pymupdf.JM_TUPLE(pymupdf.Point(p) * self.ipctm)) + " m\n"
2888: (20)                               self.last_point = pymupdf.Point(p)
2889: (12)                       else:
2890: (16)                           self.draw_cont += _format_g(pymupdf.JM_TUPLE(pymupdf.Point(p)
* self.ipctm)) + " l\n"
2891: (12)                       self.updateRect(p)
2892: (8)                    self.last_point = pymupdf.Point(points[-1])
2893: (8)                    return self.last_point
2894: (4)                def draw_bezier(
2895: (8)                    self,
2896: (8)                    p1: point_like,
2897: (8)                    p2: point_like,
2898: (8)                    p3: point_like,
2899: (8)                    p4: point_like,
2900: (4)                ) -> pymupdf.Point:
2901: (8)                    """Draw a standard cubic Bezier curve."""
2902: (8)                    p1 = pymupdf.Point(p1)
2903: (8)                    p2 = pymupdf.Point(p2)
2904: (8)                    p3 = pymupdf.Point(p3)
2905: (8)                    p4 = pymupdf.Point(p4)
2906: (8)                    if not (self.last_point == p1):
2907: (12)                       self.draw_cont += _format_g(pymupdf.JM_TUPLE(p1 * self.ipctm)) + "
m\n"
2908: (8)                    args = pymupdf.JM_TUPLE(list(p2 * self.ipctm) + list(p3 * self.ipctm)
+ list(p4 * self.ipctm))
2909: (8)                    self.draw_cont += _format_g(args) + " c\n"
2910: (8)                    self.updateRect(p1)
2911: (8)                    self.updateRect(p2)
```

```
2912: (8)                    self.updateRect(p3)
2913: (8)                    self.updateRect(p4)
2914: (8)                    self.last_point = p4
2915: (8)                    return self.last_point
2916: (4)                def draw_oval(self, tetra: typing.Union[quad_like, rect_like]) ->
pymupdf.Point:
2917: (8)                    """Draw an ellipse inside a tetrapod."""
2918: (8)                    if len(tetra) != 4:
2919: (12)                       raise ValueError("invalid arg length")
2920: (8)                    if hasattr(tetra[0], "__float__"):
2921: (12)                       q = pymupdf.Rect(tetra).quad
2922: (8)                    else:
2923: (12)                       q = pymupdf.Quad(tetra)
2924: (8)                    mt = q.ul + (q.ur - q.ul) * 0.5
2925: (8)                    mr = q.ur + (q.lr - q.ur) * 0.5
2926: (8)                    mb = q.ll + (q.lr - q.ll) * 0.5
2927: (8)                    ml = q.ul + (q.ll - q.ul) * 0.5
2928: (8)                    if not (self.last_point == ml):
2929: (12)                       self.draw_cont += _format_g(pymupdf.JM_TUPLE(ml * self.ipctm)) + "
m\n"
2930: (12)                       self.last_point = ml
2931: (8)                    self.draw_curve(ml, q.ll, mb)
2932: (8)                    self.draw_curve(mb, q.lr, mr)
2933: (8)                    self.draw_curve(mr, q.ur, mt)
2934: (8)                    self.draw_curve(mt, q.ul, ml)
2935: (8)                    self.updateRect(q.rect)
2936: (8)                    self.last_point = ml
2937: (8)                    return self.last_point
2938: (4)                def draw_circle(self, center: point_like, radius: float) -> pymupdf.Point:
2939: (8)                    """Draw a circle given its center and radius."""
2940: (8)                    if not radius > pymupdf.EPSILON:
2941: (12)                       raise ValueError("radius must be positive")
2942: (8)                    center = pymupdf.Point(center)
2943: (8)                    p1 = center - (radius, 0)
2944: (8)                    return self.draw_sector(center, p1, 360, fullSector=False)
2945: (4)                def draw_curve(
2946: (8)                    self,
2947: (8)                    p1: point_like,
2948: (8)                    p2: point_like,
2949: (8)                    p3: point_like,
2950: (4)                ) -> pymupdf.Point:
2951: (8)                    """Draw a curve between points using one control point."""
2952: (8)                    kappa = 0.55228474983
2953: (8)                    p1 = pymupdf.Point(p1)
2954: (8)                    p2 = pymupdf.Point(p2)
2955: (8)                    p3 = pymupdf.Point(p3)
2956: (8)                    k1 = p1 + (p2 - p1) * kappa
2957: (8)                    k2 = p3 + (p2 - p3) * kappa
2958: (8)                    return self.draw_bezier(p1, k1, k2, p3)
2959: (4)                def draw_sector(
2960: (8)                    self,
2961: (8)                    center: point_like,
2962: (8)                    point: point_like,
2963: (8)                    beta: float,
2964: (8)                    fullSector: bool = True,
2965: (4)                ) -> pymupdf.Point:
2966: (8)                    """Draw a circle sector."""
2967: (8)                    center = pymupdf.Point(center)
2968: (8)                    point = pymupdf.Point(point)
2969: (8)                    l3 = lambda a, b: _format_g((a, b)) + " m\n"
2970: (8)                    l4 = lambda a, b, c, d, e, f: _format_g((a, b, c, d, e, f)) + " c\n"
2971: (8)                    l5 = lambda a, b: _format_g((a, b)) + " l\n"
2972: (8)                    betar = math.radians(-beta)
2973: (8)                    w360 = math.radians(math.copysign(360, betar)) * (-1)
2974: (8)                    w90 = math.radians(math.copysign(90, betar))
2975: (8)                    w45 = w90 / 2
2976: (8)                    while abs(betar) > 2 * math.pi:
2977: (12)                       betar += w360  # bring angle below 360 degrees
2978: (8)                    if not (self.last_point == point):
```

```
2979: (12)                      self.draw_cont += l3(*pymupdf.JM_TUPLE(point * self.ipctm))
2980: (12)                      self.last_point = point
2981: (8)                   Q = pymupdf.Point(0, 0)  # just make sure it exists
2982: (8)                   C = center
2983: (8)                   P = point
2984: (8)                   S = P - C  # vector 'center' -> 'point'
2985: (8)                   rad = abs(S)  # circle radius
2986: (8)                   if not rad > pymupdf.EPSILON:
2987: (12)                      raise ValueError("radius must be positive")
2988: (8)                   alfa = self.horizontal_angle(center, point)
2989: (8)                   while abs(betar) > abs(w90):  # draw 90 degree arcs
2990: (12)                      q1 = C.x + math.cos(alfa + w90) * rad
2991: (12)                      q2 = C.y + math.sin(alfa + w90) * rad
2992: (12)                      Q = pymupdf.Point(q1, q2)  # the arc's end point
2993: (12)                      r1 = C.x + math.cos(alfa + w45) * rad / math.cos(w45)
2994: (12)                      r2 = C.y + math.sin(alfa + w45) * rad / math.cos(w45)
2995: (12)                      R = pymupdf.Point(r1, r2)  # crossing point of tangents
2996: (12)                      kappah = (1 - math.cos(w45)) * 4 / 3 / abs(R - Q)
2997: (12)                      kappa = kappah * abs(P - Q)
2998: (12)                      cp1 = P + (R - P) * kappa  # control point 1
2999: (12)                      cp2 = Q + (R - Q) * kappa  # control point 2
3000: (12)                      self.draw_cont += l4(*pymupdf.JM_TUPLE(
3001: (16)                          list(cp1 * self.ipctm) + list(cp2 * self.ipctm) + list(Q *
self.ipctm)
3002: (12)                      ))
3003: (12)                      betar -= w90  # reduce param angle by 90 deg
3004: (12)                      alfa += w90  # advance start angle by 90 deg
3005: (12)                      P = Q  # advance to arc end point
3006: (8)                   if abs(betar) > 1e-3:  # significant degrees left?
3007: (12)                      beta2 = betar / 2
3008: (12)                      q1 = C.x + math.cos(alfa + betar) * rad
3009: (12)                      q2 = C.y + math.sin(alfa + betar) * rad
3010: (12)                      Q = pymupdf.Point(q1, q2)  # the arc's end point
3011: (12)                      r1 = C.x + math.cos(alfa + beta2) * rad / math.cos(beta2)
3012: (12)                      r2 = C.y + math.sin(alfa + beta2) * rad / math.cos(beta2)
3013: (12)                      R = pymupdf.Point(r1, r2)  # crossing point of tangents
3014: (12)                      kappah = (1 - math.cos(beta2)) * 4 / 3 / abs(R - Q)  # kappa
height
3015: (12)                      kappa = kappah * abs(P - Q) / (1 - math.cos(betar))
3016: (12)                      cp1 = P + (R - P) * kappa  # control point 1
3017: (12)                      cp2 = Q + (R - Q) * kappa  # control point 2
3018: (12)                      self.draw_cont += l4(*pymupdf.JM_TUPLE(
3019: (16)                          list(cp1 * self.ipctm) + list(cp2 * self.ipctm) + list(Q *
self.ipctm)
3020: (12)                      ))
3021: (8)                   if fullSector:
3022: (12)                      self.draw_cont += l3(*pymupdf.JM_TUPLE(point * self.ipctm))
3023: (12)                      self.draw_cont += l5(*pymupdf.JM_TUPLE(center * self.ipctm))
3024: (12)                      self.draw_cont += l5(*pymupdf.JM_TUPLE(Q * self.ipctm))
3025: (8)                   self.last_point = Q
3026: (8)                   return self.last_point
3027: (4)               def draw_rect(self, rect: rect_like, *, radius=None) -> pymupdf.Point:
3028: (8)                   """Draw a rectangle.
3029: (8)                   Args:
3030: (12)                      radius: if not None, the rectangle will have rounded corners.
3031: (16)                          This is the radius of the curvature, given as percentage of
3032: (16)                          the rectangle width or height. Valid are values 0 < v <= 0.5.
3033: (16)                          For a sequence of two values, the corners will have different
3034: (16)                          radii. Otherwise, the percentage will be computed from the
3035: (16)                          shorter side. A value of (0.5, 0.5) will draw an ellipse.
3036: (8)                   """
3037: (8)                   r = pymupdf.Rect(rect)
3038: (8)                   if radius is None:  # standard rectangle
3039: (12)                      self.draw_cont += _format_g(pymupdf.JM_TUPLE(
3040: (16)                          list(r.bl * self.ipctm) + [r.width, r.height]
3041: (12)                      )) + " re\n"
3042: (12)                      self.updateRect(r)
3043: (12)                      self.last_point = r.tl
3044: (12)                      return self.last_point
```

```
3045: (8)                    if hasattr(radius, "__float__"):
3046: (12)                       if radius <= 0 or radius > 0.5:
3047: (16)                           raise ValueError(f"bad radius value {radius}.")
3048: (12)                       d = min(r.width, r.height) * radius
3049: (12)                       px = (d, 0)
3050: (12)                       py = (0, d)
3051: (8)                    elif hasattr(radius, "__len__") and len(radius) == 2:
3052: (12)                       rx, ry = radius
3053: (12)                       px = (rx * r.width, 0)
3054: (12)                       py = (0, ry * r.height)
3055: (12)                       if min(rx, ry) <= 0 or max(rx, ry) > 0.5:
3056: (16)                           raise ValueError(f"bad radius value {radius}.")
3057: (8)                    else:
3058: (12)                       raise ValueError(f"bad radius value {radius}.")
3059: (8)                    lp = self.draw_line(r.tl + py, r.bl - py)
3060: (8)                    lp = self.draw_curve(lp, r.bl, r.bl + px)
3061: (8)                    lp = self.draw_line(lp, r.br - px)
3062: (8)                    lp = self.draw_curve(lp, r.br, r.br - py)
3063: (8)                    lp = self.draw_line(lp, r.tr + py)
3064: (8)                    lp = self.draw_curve(lp, r.tr, r.tr - px)
3065: (8)                    lp = self.draw_line(lp, r.tl + px)
3066: (8)                    self.last_point = self.draw_curve(lp, r.tl, r.tl + py)
3067: (8)                    self.updateRect(r)
3068: (8)                    return self.last_point
3069: (4)                def draw_quad(self, quad: quad_like) -> pymupdf.Point:
3070: (8)                    """Draw a Quad."""
3071: (8)                    q = pymupdf.Quad(quad)
3072: (8)                    return self.draw_polyline([q.ul, q.ll, q.lr, q.ur, q.ul])
3073: (4)                def draw_zigzag(
3074: (8)                    self,
3075: (8)                    p1: point_like,
3076: (8)                    p2: point_like,
3077: (8)                    breadth: float = 2,
3078: (4)                ) -> pymupdf.Point:
3079: (8)                    """Draw a zig-zagged line from p1 to p2."""
3080: (8)                    p1 = pymupdf.Point(p1)
3081: (8)                    p2 = pymupdf.Point(p2)
3082: (8)                    S = p2 - p1  # vector start - end
3083: (8)                    rad = abs(S)  # distance of points
3084: (8)                    cnt = 4 * int(round(rad / (4 * breadth), 0))  # always take full
phases
3085: (8)                    if cnt < 4:
3086: (12)                       raise ValueError("points too close")
3087: (8)                    mb = rad / cnt  # revised breadth
3088: (8)                    matrix = pymupdf.Matrix(pymupdf.util_hor_matrix(p1, p2))  # normalize
line to x-axis
3089: (8)                    i_mat = ~matrix  # get original position
3090: (8)                    points = []  # stores edges
3091: (8)                    for i in range(1, cnt):
3092: (12)                       if i % 4 == 1:  # point "above" connection
3093: (16)                           p = pymupdf.Point(i, -1) * mb
3094: (12)                       elif i % 4 == 3:  # point "below" connection
3095: (16)                           p = pymupdf.Point(i, 1) * mb
3096: (12)                       else:  # ignore others
3097: (16)                           continue
3098: (12)                       points.append(p * i_mat)
3099: (8)                    self.draw_polyline([p1] + points + [p2])  # add start and end points
3100: (8)                    return p2
3101: (4)                def draw_squiggle(
3102: (8)                    self,
3103: (8)                    p1: point_like,
3104: (8)                    p2: point_like,
3105: (8)                    breadth=2,
3106: (4)                ) -> pymupdf.Point:
3107: (8)                    """Draw a squiggly line from p1 to p2."""
3108: (8)                    p1 = pymupdf.Point(p1)
3109: (8)                    p2 = pymupdf.Point(p2)
3110: (8)                    S = p2 - p1  # vector start - end
3111: (8)                    rad = abs(S)  # distance of points
```

```
3112: (8)                    cnt = 4 * int(round(rad / (4 * breadth), 0))  # always take full
phases
3113: (8)                    if cnt < 4:
3114: (12)                       raise ValueError("points too close")
3115: (8)                    mb = rad / cnt  # revised breadth
3116: (8)                    matrix = pymupdf.Matrix(pymupdf.util_hor_matrix(p1, p2))  # normalize
line to x-axis
3117: (8)                    i_mat = ~matrix  # get original position
3118: (8)                    k = 2.4142135623765633  # y of draw_curve helper point
3119: (8)                    points = []  # stores edges
3120: (8)                    for i in range(1, cnt):
3121: (12)                       if i % 4 == 1:  # point "above" connection
3122: (16)                           p = pymupdf.Point(i, -k) * mb
3123: (12)                       elif i % 4 == 3:  # point "below" connection
3124: (16)                           p = pymupdf.Point(i, k) * mb
3125: (12)                       else:  # else on connection line
3126: (16)                           p = pymupdf.Point(i, 0) * mb
3127: (12)                       points.append(p * i_mat)
3128: (8)                    points = [p1] + points + [p2]
3129: (8)                    cnt = len(points)
3130: (8)                    i = 0
3131: (8)                    while i + 2 < cnt:
3132: (12)                       self.draw_curve(points[i], points[i + 1], points[i + 2])
3133: (12)                       i += 2
3134: (8)                    return p2
3135: (4)                def insert_text(
3136: (8)                    self,
3137: (8)                    point: point_like,
3138: (8)                    buffer: typing.Union[str, list],
3139: (8)                    fontsize: float = 11,
3140: (8)                    lineheight: OptFloat = None,
3141: (8)                    fontname: str = "helv",
3142: (8)                    fontfile: OptStr = None,
3143: (8)                    set_simple: bool = 0,
3144: (8)                    encoding: int = 0,
3145: (8)                    color: OptSeq = None,
3146: (8)                    fill: OptSeq = None,
3147: (8)                    render_mode: int = 0,
3148: (8)                    border_width: float = 0.05,
3149: (8)                    rotate: int = 0,
3150: (8)                    morph: OptSeq = None,
3151: (8)                    stroke_opacity: float = 1,
3152: (8)                    fill_opacity: float = 1,
3153: (8)                    oc: int = 0,
3154: (4)                ) -> int:
3155: (8)                    if not bool(buffer):
3156: (12)                       return 0
3157: (8)                    if type(buffer) not in (list, tuple):
3158: (12)                       text = buffer.splitlines()
3159: (8)                    else:
3160: (12)                       text = buffer
3161: (8)                    if not len(text) > 0:
3162: (12)                       return 0
3163: (8)                    point = pymupdf.Point(point)
3164: (8)                    try:
3165: (12)                       maxcode = max([ord(c) for c in " ".join(text)])
3166: (8)                    except Exception:
3167: (12)                       pymupdf.exception_info()
3168: (12)                       return 0
3169: (8)                    fname = fontname
3170: (8)                    if fname.startswith("/"):
3171: (12)                       fname = fname[1:]
3172: (8)                    xref = self.page.insert_font(
3173: (12)                       fontname=fname, fontfile=fontfile, encoding=encoding,
set_simple=set_simple
3174: (8)                    )
3175: (8)                    fontinfo = pymupdf.CheckFontInfo(self.doc, xref)
3176: (8)                    fontdict = fontinfo[1]
3177: (8)                    ordering = fontdict["ordering"]
```

```
(8)                 simple = fontdict["simple"]
(8)                 bfname = fontdict["name"]
(8)                 ascender = fontdict["ascender"]
(8)                 descender = fontdict["descender"]
(8)                 if lineheight:
(12)                    lheight = fontsize * lineheight
(8)                 elif ascender - descender <= 1:
(12)                    lheight = fontsize * 1.2
(8)                 else:
(12)                    lheight = fontsize * (ascender - descender)
(8)                 if maxcode > 255:
(12)                    glyphs = self.doc.get_char_widths(xref, maxcode + 1)
(8)                 else:
(12)                    glyphs = fontdict["glyphs"]
(8)                 tab = []
(8)                 for t in text:
(12)                    if simple and bfname not in ("Symbol", "ZapfDingbats"):
(16)                        g = None
(12)                    else:
(16)                        g = glyphs
(12)                    tab.append(pymupdf.getTJstr(t, g, simple, ordering))
(8)                 text = tab
(8)                 color_str = pymupdf.ColorCode(color, "c")
(8)                 fill_str = pymupdf.ColorCode(fill, "f")
(8)                 if not fill and render_mode == 0:  # ensure fill color when 0 Tr
(12)                    fill = color
(12)                    fill_str = pymupdf.ColorCode(color, "f")
(8)                 morphing = pymupdf.CheckMorph(morph)
(8)                 rot = rotate
(8)                 if rot % 90 != 0:
(12)                    raise ValueError("bad rotate value")
(8)                 while rot < 0:
(12)                    rot += 360
(8)                 rot = rot % 360  # text rotate = 0, 90, 270, 180
(8)                 templ1 = lambda a, b, c, d, e, f, g: f"\nq\n{a}{b}BT\n{c}1 0 0 1
{_format_g((d, e))} Tm\n/{f} {_format_g(g)} Tf "
(8)                 templ2 = lambda a: f"TJ\n0 -{_format_g(a)} TD\n"
(8)                 cmp90 = "0 1 -1 0 0 0 cm\n"  # rotates 90 deg counter-clockwise
(8)                 cmm90 = "0 -1 1 0 0 0 cm\n"  # rotates 90 deg clockwise
(8)                 cm180 = "-1 0 0 -1 0 0 cm\n"  # rotates by 180 deg.
(8)                 height = self.height
(8)                 width = self.width
(8)                 if morphing:
(12)                    m1 = pymupdf.Matrix(1, 0, 0, 1, morph[0].x + self.x, height -
morph[0].y - self.y)
(12)                    mat = ~m1 * morph[1] * m1
(12)                    cm = _format_g(pymupdf.JM_TUPLE(mat)) + " cm\n"
(8)                 else:
(12)                    cm = ""
(8)                 top = height - point.y - self.y  # start of 1st char
(8)                 left = point.x + self.x  # start of 1. char
(8)                 space = top  # space available
(8)                 if rot == 90:
(12)                    left = height - point.y - self.y
(12)                    top = -point.x - self.x
(12)                    cm += cmp90
(12)                    space = width - abs(top)
(8)                 elif rot == 270:
(12)                    left = -height + point.y + self.y
(12)                    top = point.x + self.x
(12)                    cm += cmm90
(12)                    space = abs(top)
(8)                 elif rot == 180:
(12)                    left = -point.x - self.x
(12)                    top = -height + point.y + self.y
(12)                    cm += cm180
(12)                    space = abs(point.y + self.y)
(8)                 optcont = self.page._get_optional_content(oc)
(8)                 if optcont is not None:
```

```
3245: (12)                      bdc = "/OC /%s BDC\n" % optcont
3246: (12)                      emc = "EMC\n"
3247: (8)                   else:
3248: (12)                      bdc = emc = ""
3249: (8)                   alpha = self.page._set_opacity(CA=stroke_opacity, ca=fill_opacity)
3250: (8)                   if alpha is None:
3251: (12)                      alpha = ""
3252: (8)                   else:
3253: (12)                      alpha = "/%s gs\n" % alpha
3254: (8)                   nres = templ1(bdc, alpha, cm, left, top, fname, fontsize)
3255: (8)                   if render_mode > 0:
3256: (12)                      nres += "%i Tr " % render_mode
3257: (12)                      nres += _format_g(border_width * fontsize) + " w "
3258: (8)                   if color is not None:
3259: (12)                      nres += color_str
3260: (8)                   if fill is not None:
3261: (12)                      nres += fill_str
3262: (8)                   nres += text[0]
3263: (8)                   nlines = 1  # set output line counter
3264: (8)                   if len(text) > 1:
3265: (12)                      nres += templ2(lheight)  # line 1
3266: (8)                   else:
3267: (12)                      nres += 'TJ'
3268: (8)                   for i in range(1, len(text)):
3269: (12)                      if space < lheight:
3270: (16)                          break  # no space left on page
3271: (12)                      if i > 1:
3272: (16)                          nres += "\nT* "
3273: (12)                      nres += text[i] + 'TJ'
3274: (12)                      space -= lheight
3275: (12)                      nlines += 1
3276: (8)                   nres += "\nET\n%sQ\n" % emc
3277: (8)                   self.text_cont += nres
3278: (8)                   return nlines
3279: (4)               def insert_textbox(
3280: (8)                   self,
3281: (8)                   rect: rect_like,
3282: (8)                   buffer: typing.Union[str, list],
3283: (8)                   fontname: OptStr = "helv",
3284: (8)                   fontfile: OptStr = None,
3285: (8)                   fontsize: float = 11,
3286: (8)                   lineheight: OptFloat = None,
3287: (8)                   set_simple: bool = 0,
3288: (8)                   encoding: int = 0,
3289: (8)                   color: OptSeq = None,
3290: (8)                   fill: OptSeq = None,
3291: (8)                   expandtabs: int = 1,
3292: (8)                   border_width: float = 0.05,
3293: (8)                   align: int = 0,
3294: (8)                   render_mode: int = 0,
3295: (8)                   rotate: int = 0,
3296: (8)                   morph: OptSeq = None,
3297: (8)                   stroke_opacity: float = 1,
3298: (8)                   fill_opacity: float = 1,
3299: (8)                   oc: int = 0,
3300: (4)               ) -> float:
3301: (8)                   """Insert text into a given rectangle.
3302: (8)                   Args:
3303: (12)                      rect -- the textbox to fill
3304: (12)                      buffer -- text to be inserted
3305: (12)                      fontname -- a Base-14 font, font name or '/name'
3306: (12)                      fontfile -- name of a font file
3307: (12)                      fontsize -- font size
3308: (12)                      lineheight -- overwrite the font property
3309: (12)                      color -- RGB stroke color triple
3310: (12)                      fill -- RGB fill color triple
3311: (12)                      render_mode -- text rendering control
3312: (12)                      border_width -- thickness of glyph borders as percentage of
fontsize
```

```
3313: (12)                      expandtabs -- handles tabulators with string function
3314: (12)                      align -- left, center, right, justified
3315: (12)                      rotate -- 0, 90, 180, or 270 degrees
3316: (12)                      morph -- morph box with a matrix and a fixpoint
3317: (8)                   Returns:
3318: (12)                      unused or deficit rectangle area (float)
3319: (8)                   """
3320: (8)                   rect = pymupdf.Rect(rect)
3321: (8)                   if rect.is_empty or rect.is_infinite:
3322: (12)                      raise ValueError("text box must be finite and not empty")
3323: (8)                   color_str = pymupdf.ColorCode(color, "c")
3324: (8)                   fill_str = pymupdf.ColorCode(fill, "f")
3325: (8)                   if fill is None and render_mode == 0:  # ensure fill color for 0 Tr
3326: (12)                      fill = color
3327: (12)                      fill_str = pymupdf.ColorCode(color, "f")
3328: (8)                   optcont = self.page._get_optional_content(oc)
3329: (8)                   if optcont is not None:
3330: (12)                      bdc = "/OC /%s BDC\n" % optcont
3331: (12)                      emc = "EMC\n"
3332: (8)                   else:
3333: (12)                      bdc = emc = ""
3334: (8)                   alpha = self.page._set_opacity(CA=stroke_opacity, ca=fill_opacity)
3335: (8)                   if alpha is None:
3336: (12)                      alpha = ""
3337: (8)                   else:
3338: (12)                      alpha = "/%s gs\n" % alpha
3339: (8)                   if rotate % 90 != 0:
3340: (12)                      raise ValueError("rotate must be multiple of 90")
3341: (8)                   rot = rotate
3342: (8)                   while rot < 0:
3343: (12)                      rot += 360
3344: (8)                   rot = rot % 360
3345: (8)                   if not bool(buffer):
3346: (12)                      return rect.height if rot in (0, 180) else rect.width
3347: (8)                   cmp90 = "0 1 -1 0 0 0 cm\n"  # rotates counter-clockwise
3348: (8)                   cmm90 = "0 -1 1 0 0 0 cm\n"  # rotates clockwise
3349: (8)                   cm180 = "-1 0 0 -1 0 0 cm\n"  # rotates by 180 deg.
3350: (8)                   height = self.height
3351: (8)                   fname = fontname
3352: (8)                   if fname.startswith("/"):
3353: (12)                      fname = fname[1:]
3354: (8)                   xref = self.page.insert_font(
3355: (12)                      fontname=fname, fontfile=fontfile, encoding=encoding,
set_simple=set_simple
3356: (8)                   )
3357: (8)                   fontinfo = pymupdf.CheckFontInfo(self.doc, xref)
3358: (8)                   fontdict = fontinfo[1]
3359: (8)                   ordering = fontdict["ordering"]
3360: (8)                   simple = fontdict["simple"]
3361: (8)                   glyphs = fontdict["glyphs"]
3362: (8)                   bfname = fontdict["name"]
3363: (8)                   ascender = fontdict["ascender"]
3364: (8)                   descender = fontdict["descender"]
3365: (8)                   if lineheight:
3366: (12)                      lheight_factor = lineheight
3367: (8)                   elif ascender - descender <= 1:
3368: (12)                      lheight_factor = 1.2
3369: (8)                   else:
3370: (12)                      lheight_factor = ascender - descender
3371: (8)                   lheight = fontsize * lheight_factor
3372: (8)                   if type(buffer) in (list, tuple):
3373: (12)                      t0 = "\n".join(buffer)
3374: (8)                   else:
3375: (12)                      t0 = buffer
3376: (8)                   maxcode = max([ord(c) for c in t0])
3377: (8)                   if simple and maxcode > 255:
3378: (12)                      t0 = "".join([c if ord(c) < 256 else "?" for c in t0])
3379: (8)                   t0 = t0.splitlines()
3380: (8)                   glyphs = self.doc.get_char_widths(xref, maxcode + 1)
```

```
3381: (8)                    if simple and bfname not in ("Symbol", "ZapfDingbats"):
3382: (12)                       tj_glyphs = None
3383: (8)                    else:
3384: (12)                       tj_glyphs = glyphs
3385: (8)                    def pixlen(x):
3386: (12)                       """Calculate pixel length of x."""
3387: (12)                       if ordering < 0:
3388: (16)                           return sum([glyphs[ord(c)][1] for c in x]) * fontsize
3389: (12)                       else:
3390: (16)                           return len(x) * fontsize
3391: (8)                    if ordering < 0:
3392: (12)                       blen = glyphs[32][1] * fontsize  # pixel size of space character
3393: (8)                    else:
3394: (12)                       blen = fontsize
3395: (8)                    text = ""  # output buffer
3396: (8)                    if pymupdf.CheckMorph(morph):
3397: (12)                       m1 = pymupdf.Matrix(
3398: (16)                           1, 0, 0, 1, morph[0].x + self.x, self.height - morph[0].y -
self.y
3399: (12)                       )
3400: (12)                       mat = ~m1 * morph[1] * m1
3401: (12)                       cm = _format_g(pymupdf.JM_TUPLE(mat)) + " cm\n"
3402: (8)                    else:
3403: (12)                       cm = ""
3404: (8)                    progr = 1  # direction of line progress
3405: (8)                    c_pnt = pymupdf.Point(0, fontsize * ascender)  # used for line
progress
3406: (8)                    if rot == 0:  # normal orientation
3407: (12)                       point = rect.tl + c_pnt  # line 1 is 'lheight' below top
3408: (12)                       maxwidth = rect.width  # pixels available in one line
3409: (12)                       maxheight = rect.height  # available text height
3410: (8)                    elif rot == 90:  # rotate counter clockwise
3411: (12)                       c_pnt = pymupdf.Point(fontsize * ascender, 0)  # progress in x-
direction
3412: (12)                       point = rect.bl + c_pnt  # line 1 'lheight' away from left
3413: (12)                       maxwidth = rect.height  # pixels available in one line
3414: (12)                       maxheight = rect.width  # available text height
3415: (12)                       cm += cmp90
3416: (8)                    elif rot == 180:  # text upside down
3417: (12)                       c_pnt = -pymupdf.Point(0, fontsize * ascender)
3418: (12)                       point = rect.br + c_pnt  # line 1 'lheight' above bottom
3419: (12)                       maxwidth = rect.width  # pixels available in one line
3420: (12)                       progr = -1  # subtract lheight for next line
3421: (12)                       maxheight =rect.height  # available text height
3422: (12)                       cm += cm180
3423: (8)                    else:  # rotate clockwise (270 or -90)
3424: (12)                       c_pnt = -pymupdf.Point(fontsize * ascender, 0)
3425: (12)                       point = rect.tr + c_pnt  # line 1 'lheight' left of right
3426: (12)                       maxwidth = rect.height  # pixels available in one line
3427: (12)                       progr = -1  # subtract lheight for next line
3428: (12)                       maxheight = rect.width  # available text height
3429: (12)                       cm += cmm90
3430: (8)                    just_tab = []  # 'justify' indicators per line
3431: (8)                    for i, line in enumerate(t0):
3432: (12)                       line_t = line.expandtabs(expandtabs).split(" ")  # split into
words
3433: (12)                       num_words = len(line_t)
3434: (12)                       lbuff = ""  # init line buffer
3435: (12)                       rest = maxwidth  # available line pixels
3436: (12)                       for j in range(num_words):
3437: (16)                           word = line_t[j]
3438: (16)                           pl_w = pixlen(word)  # pixel len of word
3439: (16)                           if rest >= pl_w:  # does it fit on the line?
3440: (20)                               lbuff += word + " "  # yes, append word
3441: (20)                               rest -= pl_w + blen  # update available line space
3442: (20)                               continue  # next word
3443: (16)                           if lbuff:
3444: (20)                               lbuff = lbuff.rstrip() + "\n"  # line full, append line
break
```

```
3445: (20)                                text += lbuff  # append to total text
3446: (20)                                just_tab.append(True)  # can align-justify
3447: (16)                            lbuff = ""  # re-init line buffer
3448: (16)                            rest = maxwidth  # re-init avail. space
3449: (16)                            if pl_w <= maxwidth:  # word shorter than 1 line?
3450: (20)                                lbuff = word + " "  # start the line with it
3451: (20)                                rest = maxwidth - pl_w - blen  # update free space
3452: (20)                                continue
3453: (16)                            if len(just_tab) > 0:
3454: (20)                                just_tab[-1] = False  # cannot align-justify
3455: (16)                            for c in word:
3456: (20)                                if pixlen(lbuff) <= maxwidth - pixlen(c):
3457: (24)                                    lbuff += c
3458: (20)                                else:  # line full
3459: (24)                                    lbuff += "\n"  # close line
3460: (24)                                    text += lbuff  # append to text
3461: (24)                                    just_tab.append(False)  # cannot align-justify
3462: (24)                                    lbuff = c  # start new line with this char
3463: (16)                            lbuff += " "  # finish long word
3464: (16)                            rest = maxwidth - pixlen(lbuff)  # long word stored
3465: (12)                        if lbuff:  # unprocessed line content?
3466: (16)                            text += lbuff.rstrip()  # append to text
3467: (16)                            just_tab.append(False)  # cannot align-justify
3468: (12)                        if i < len(t0) - 1:  # not the last line?
3469: (16)                            text += "\n"  # insert line break
3470: (8)                    if text.endswith("\n"):
3471: (12)                        text = text[:-1]
3472: (8)                    lb_count = text.count("\n") + 1  # number of lines written
3473: (8)                    text_height = lheight * lb_count - descender * fontsize
3474: (8)                    more = text_height - maxheight  # difference to height limit
3475: (8)                    if more > pymupdf.EPSILON:  # landed too much outside rect
3476: (12)                       return (-1) * more  # return deficit, don't output
3477: (8)                    more = abs(more)
3478: (8)                    if more < pymupdf.EPSILON:
3479: (12)                       more = 0  # don't bother with epsilons
3480: (8)                    nres = "\nq\n%s%sBT\n" % (bdc, alpha) + cm  # initialize output buffer
3481: (8)                    templ = lambda a, b, c, d: f"1 0 0 1 {_format_g((a, b))} Tm /{c}
{_format_g(d)} Tf "
3482: (8)                    text_t = text.splitlines()  # split text in lines again
3483: (8)                    just_tab[-1] = False  # never justify last line
3484: (8)                    for i, t in enumerate(text_t):
3485: (12)                       spacing = 0
3486: (12)                       pl = maxwidth - pixlen(t)  # length of empty line part
3487: (12)                       pnt = point + c_pnt * (i * lheight_factor)  # text start of line
3488: (12)                       if align == 1:  # center: right shift by half width
3489: (16)                           if rot in (0, 180):
3490: (20)                               pnt = pnt + pymupdf.Point(pl / 2, 0) * progr
3491: (16)                           else:
3492: (20)                               pnt = pnt - pymupdf.Point(0, pl / 2) * progr
3493: (12)                       elif align == 2:  # right: right shift by full width
3494: (16)                           if rot in (0, 180):
3495: (20)                               pnt = pnt + pymupdf.Point(pl, 0) * progr
3496: (16)                           else:
3497: (20)                               pnt = pnt - pymupdf.Point(0, pl) * progr
3498: (12)                       elif align == 3:  # justify
3499: (16)                           spaces = t.count(" ")  # number of spaces in line
3500: (16)                           if spaces > 0 and just_tab[i]:  # if any, and we may justify
3501: (20)                               spacing = pl / spaces  # make every space this much larger
3502: (16)                           else:
3503: (20)                               spacing = 0  # keep normal space length
3504: (12)                       top = height - pnt.y - self.y
3505: (12)                       left = pnt.x + self.x
3506: (12)                       if rot == 90:
3507: (16)                           left = height - pnt.y - self.y
3508: (16)                           top = -pnt.x - self.x
3509: (12)                       elif rot == 270:
3510: (16)                           left = -height + pnt.y + self.y
3511: (16)                           top = pnt.x + self.x
3512: (12)                       elif rot == 180:
```

```
3513: (16)                          left = -pnt.x - self.x
3514: (16)                          top = -height + pnt.y + self.y
3515: (12)                      nres += templ(left, top, fname, fontsize)
3516: (12)                      if render_mode > 0:
3517: (16)                          nres += "%i Tr " % render_mode
3518: (16)                          nres += _format_g(border_width * fontsize) + " w "
3519: (12)                      if align == 3:
3520: (16)                          nres += _format_g(spacing) + " Tw "
3521: (12)                      if color is not None:
3522: (16)                          nres += color_str
3523: (12)                      if fill is not None:
3524: (16)                          nres += fill_str
3525: (12)                      nres += "%sTJ\n" % pymupdf.getTJstr(t, tj_glyphs, simple,
ordering)
3526: (8)                   nres += "ET\n%sQ\n" % emc
3527: (8)                   self.text_cont += nres
3528: (8)                   self.updateRect(rect)
3529: (8)                   return more
3530: (4)               def finish(
3531: (8)                   self,
3532: (8)                   width: float = 1,
3533: (8)                   color: OptSeq = (0,),
3534: (8)                   fill: OptSeq = None,
3535: (8)                   lineCap: int = 0,
3536: (8)                   lineJoin: int = 0,
3537: (8)                   dashes: OptStr = None,
3538: (8)                   even_odd: bool = False,
3539: (8)                   morph: OptSeq = None,
3540: (8)                   closePath: bool = True,
3541: (8)                   fill_opacity: float = 1,
3542: (8)                   stroke_opacity: float = 1,
3543: (8)                   oc: int = 0,
3544: (4)               ) -> None:
3545: (8)                   """Finish the current drawing segment.
3546: (8)                   Notes:
3547: (12)                      Apply colors, opacity, dashes, line style and width, or
3548: (12)                      morphing. Also whether to close the path
3549: (12)                      by connecting last to first point.
3550: (8)                   """
3551: (8)                   if self.draw_cont == "":  # treat empty contents as no-op
3552: (12)                      return
3553: (8)                   if width == 0:  # border color makes no sense then
3554: (12)                      color = None
3555: (8)                   elif color is None:  # vice versa
3556: (12)                      width = 0
3557: (8)                   color_str = pymupdf.ColorCode(color, "c")  # ensure proper color
string
3558: (8)                   fill_str = pymupdf.ColorCode(fill, "f")  # ensure proper fill string
3559: (8)                   optcont = self.page._get_optional_content(oc)
3560: (8)                   if optcont is not None:
3561: (12)                      self.draw_cont = "/OC /%s BDC\n" % optcont + self.draw_cont
3562: (12)                      emc = "EMC\n"
3563: (8)                   else:
3564: (12)                      emc = ""
3565: (8)                   alpha = self.page._set_opacity(CA=stroke_opacity, ca=fill_opacity)
3566: (8)                   if alpha is not None:
3567: (12)                      self.draw_cont = "/%s gs\n" % alpha + self.draw_cont
3568: (8)                   if width != 1 and width != 0:
3569: (12)                      self.draw_cont += _format_g(width) + " w\n"
3570: (8)                   if lineCap != 0:
3571: (12)                      self.draw_cont = "%i J\n" % lineCap + self.draw_cont
3572: (8)                   if lineJoin != 0:
3573: (12)                      self.draw_cont = "%i j\n" % lineJoin + self.draw_cont
3574: (8)                   if dashes not in (None, "", "[] 0"):
3575: (12)                      self.draw_cont = "%s d\n" % dashes + self.draw_cont
3576: (8)                   if closePath:
3577: (12)                      self.draw_cont += "h\n"
3578: (12)                      self.last_point = None
3579: (8)                   if color is not None:
```

```
3580: (12)                      self.draw_cont += color_str
3581: (8)                   if fill is not None:
3582: (12)                      self.draw_cont += fill_str
3583: (12)                      if color is not None:
3584: (16)                          if not even_odd:
3585: (20)                              self.draw_cont += "B\n"
3586: (16)                          else:
3587: (20)                              self.draw_cont += "B*\n"
3588: (12)                      else:
3589: (16)                          if not even_odd:
3590: (20)                              self.draw_cont += "f\n"
3591: (16)                          else:
3592: (20)                              self.draw_cont += "f*\n"
3593: (8)                   else:
3594: (12)                      self.draw_cont += "S\n"
3595: (8)                   self.draw_cont += emc
3596: (8)                   if pymupdf.CheckMorph(morph):
3597: (12)                      m1 = pymupdf.Matrix(
3598: (16)                          1, 0, 0, 1, morph[0].x + self.x, self.height - morph[0].y -
self.y
3599: (12)                      )
3600: (12)                      mat = ~m1 * morph[1] * m1
3601: (12)                      self.draw_cont = _format_g(pymupdf.JM_TUPLE(mat)) + " cm\n" +
self.draw_cont
3602: (8)                   self.totalcont += "\nq\n" + self.draw_cont + "Q\n"
3603: (8)                   self.draw_cont = ""
3604: (8)                   self.last_point = None
3605: (8)                   return
3606: (4)               def commit(self, overlay: bool = True) -> None:
3607: (8)                   """Update the page's /Contents object with Shape data.
3608: (8)                   The argument controls whether data appear in foreground (default)
3609: (8)                   or background.
3610: (8)                   """
3611: (8)                   pymupdf.CheckParent(self.page)  # doc may have died meanwhile
3612: (8)                   self.totalcont += self.text_cont
3613: (8)                   self.totalcont = self.totalcont.encode()
3614: (8)                   if self.totalcont:
3615: (12)                      if overlay:
3616: (16)                          self.page.wrap_contents()  # ensure a balanced graphics state
3617: (12)                      xref = pymupdf.TOOLS._insert_contents(self.page, b" ", overlay)
3618: (12)                      self.doc.update_stream(xref, self.totalcont)
3619: (8)                   self.last_point = None  # clean up ...
3620: (8)                   self.rect = None  #
3621: (8)                   self.draw_cont = ""  # for potential ...
3622: (8)                   self.text_cont = ""  # ...
3623: (8)                   self.totalcont = ""  # re-use
3624: (0)           def apply_redactions(
3625: (4)               page: pymupdf.Page, images: int = 2, graphics: int = 1, text: int = 0
3626: (0)           ) -> bool:
3627: (4)               """Apply the redaction annotations of the page.
3628: (4)               Args:
3629: (8)                   page: the PDF page.
3630: (8)                   images:
3631: (14)                        0 - ignore images
3632: (14)                        1 - remove all overlapping images
3633: (14)                        2 - blank out overlapping image parts
3634: (14)                        3 - remove image unless invisible
3635: (8)                   graphics:
3636: (14)                        0 - ignore graphics
3637: (14)                        1 - remove graphics if contained in rectangle
3638: (14)                        2 - remove all overlapping graphics
3639: (8)                   text:
3640: (14)                        0 - remove text
3641: (14)                        1 - ignore text
3642: (4)               """
3643: (4)               def center_rect(annot_rect, new_text, font, fsize):
3644: (8)                   """Calculate minimal sub-rectangle for the overlay text.
3645: (8)                   Notes:
3646: (12)                      Because 'insert_textbox' supports no vertical text centering,
```

```
3647: (12)                      we calculate an approximate number of lines here and return a
3648: (12)                      sub-rect with smaller height, which should still be sufficient.
3649: (8)                   Args:
3650: (12)                      annot_rect: the annotation rectangle
3651: (12)                      new_text: the text to insert.
3652: (12)                      font: the fontname. Must be one of the CJK or Base-14 set, else
3653: (16)                          the rectangle is returned unchanged.
3654: (12)                      fsize: the fontsize
3655: (8)                   Returns:
3656: (12)                      A rectangle to use instead of the annot rectangle.
3657: (8)                   """
3658: (8)                   exception_types = (ValueError, mupdf.FzErrorBase)
3659: (8)                   if pymupdf.mupdf_version_tuple < (1, 24):
3660: (12)                      exception_types = ValueError
3661: (8)                   if not new_text or annot_rect.width <= pymupdf.EPSILON:
3662: (12)                      return annot_rect
3663: (8)                   try:
3664: (12)                      text_width = pymupdf.get_text_length(new_text, font, fsize)
3665: (8)                   except exception_types:  # unsupported font
3666: (12)                      if g_exceptions_verbose:
3667: (16)                          pymupdf.exception_info()
3668: (12)                      return annot_rect
3669: (8)                   line_height = fsize * 1.2
3670: (8)                   limit = annot_rect.width
3671: (8)                   h = math.ceil(text_width / limit) * line_height  # estimate rect
height
3672: (8)                   if h >= annot_rect.height:
3673: (12)                      return annot_rect
3674: (8)                   r = annot_rect
3675: (8)                   y = (annot_rect.tl.y + annot_rect.bl.y - h) * 0.5
3676: (8)                   r.y0 = y
3677: (8)                   return r
3678: (4)               pymupdf.CheckParent(page)
3679: (4)               doc = page.parent
3680: (4)               if doc.is_encrypted or doc.is_closed:
3681: (8)                   raise ValueError("document closed or encrypted")
3682: (4)               if not doc.is_pdf:
3683: (8)                   raise ValueError("is no PDF")
3684: (4)               redact_annots = []  # storage of annot values
3685: (4)               for annot in page.annots(
3686: (8)                   types=(pymupdf.PDF_ANNOT_REDACT,)  # pylint: disable=no-member
3687: (4)               ):
3688: (8)                   redact_annots.append(annot._get_redact_values())  # save annot values
3689: (4)               if redact_annots == []:  # any redactions on this page?
3690: (8)                   return False  # no redactions
3691: (4)               rc = page._apply_redactions(text, images, graphics)  # call MuPDF
3692: (4)               if not rc:  # should not happen really
3693: (8)                   raise ValueError("Error applying redactions.")
3694: (4)               shape = page.new_shape()
3695: (4)               for redact in redact_annots:
3696: (8)                   annot_rect = redact["rect"]
3697: (8)                   fill = redact["fill"]
3698: (8)                   if fill:
3699: (12)                      shape.draw_rect(annot_rect)  # colorize the rect background
3700: (12)                      shape.finish(fill=fill, color=fill)
3701: (8)                   if "text" in redact.keys():  # if we also have text
3702: (12)                      new_text = redact["text"]
3703: (12)                      align = redact.get("align", 0)
3704: (12)                      fname = redact["fontname"]
3705: (12)                      fsize = redact["fontsize"]
3706: (12)                      color = redact["text_color"]
3707: (12)                      trect = center_rect(annot_rect, new_text, fname, fsize)
3708: (12)                      rc = -1
3709: (12)                      while rc < 0 and fsize >= 4:  # while not enough room
3710: (16)                          rc = shape.insert_textbox(
3711: (20)                              trect,
3712: (20)                              new_text,
3713: (20)                              fontname=fname,
3714: (20)                              fontsize=fsize,
```

```
(20)                                color=color,
(20)                                align=align,
(16)                            )
(16)                            fsize -= 0.5  # reduce font if unsuccessful
(4)                 shape.commit()  # append new contents object
(4)                 return True
(0)             def scrub(
(4)                 doc: pymupdf.Document,
(4)                 attached_files: bool = True,
(4)                 clean_pages: bool = True,
(4)                 embedded_files: bool = True,
(4)                 hidden_text: bool = True,
(4)                 javascript: bool = True,
(4)                 metadata: bool = True,
(4)                 redactions: bool = True,
(4)                 redact_images: int = 0,
(4)                 remove_links: bool = True,
(4)                 reset_fields: bool = True,
(4)                 reset_responses: bool = True,
(4)                 thumbnails: bool = True,
(4)                 xml_metadata: bool = True,
(0)             ) -> None:
(4)                 def remove_hidden(cont_lines):
(8)                     """Remove hidden text from a PDF page.
(8)                     Args:
(12)                        cont_lines: list of lines with /Contents content. Should have
status
(16)                            from after page.cleanContents().
(8)                     Returns:
(12)                        List of /Contents lines from which hidden text has been removed.
(8)                     Notes:
(12)                        The input must have been created after the page's /Contents
object(s)
(12)                        have been cleaned with page.cleanContents(). This ensures a
standard
(12)                        formatting: one command per line, single spaces between operators.
(12)                        This allows for drastic simplification of this code.
(8)                     """
(8)                     out_lines = []  # will return this
(8)                     in_text = False  # indicate if within BT/ET object
(8)                     suppress = False  # indicate text suppression active
(8)                     make_return = False
(8)                     for line in cont_lines:
(12)                        if line == b"BT":  # start of text object
(16)                            in_text = True  # switch on
(16)                            out_lines.append(line)  # output it
(16)                            continue
(12)                        if line == b"ET":  # end of text object
(16)                            in_text = False  # switch off
(16)                            out_lines.append(line)  # output it
(16)                            continue
(12)                        if line == b"3 Tr":  # text suppression operator
(16)                            suppress = True  # switch on
(16)                            make_return = True
(16)                            continue
(12)                        if line[-2:] == b"Tr" and line[0] != b"3":
(16)                            suppress = False  # text rendering changed
(16)                            out_lines.append(line)
(16)                            continue
(12)                        if line == b"Q":  # unstack command also switches off
(16)                            suppress = False
(16)                            out_lines.append(line)
(16)                            continue
(12)                        if suppress and in_text:  # suppress hidden lines
(16)                            continue
(12)                        out_lines.append(line)
(8)                     if make_return:
(12)                        return out_lines
(8)                     else:
```

```
3781: (12)                        return None
3782: (4)                if not doc.is_pdf:  # only works for PDF
3783: (8)                    raise ValueError("is no PDF")
3784: (4)                if doc.is_encrypted or doc.is_closed:
3785: (8)                    raise ValueError("closed or encrypted doc")
3786: (4)                if clean_pages is False:
3787: (8)                    hidden_text = False
3788: (8)                    redactions = False
3789: (4)                if metadata:
3790: (8)                    doc.set_metadata({})  # remove standard metadata
3791: (4)                for page in doc:
3792: (8)                    if reset_fields:
3793: (12)                       for widget in page.widgets():
3794: (16)                           widget.reset()
3795: (8)                    if remove_links:
3796: (12)                       links = page.get_links()  # list of all links on page
3797: (12)                       for link in links:  # remove all links
3798: (16)                           page.delete_link(link)
3799: (8)                    found_redacts = False
3800: (8)                    for annot in page.annots():
3801: (12)                       if annot.type[0] == mupdf.PDF_ANNOT_FILE_ATTACHMENT and
attached_files:
3802: (16)                           annot.update_file(buffer=b" ")  # set file content to empty
3803: (12)                       if reset_responses:
3804: (16)                           annot.delete_responses()
3805: (12)                       if annot.type[0] == pymupdf.PDF_ANNOT_REDACT:  # pylint:
disable=no-member
3806: (16)                           found_redacts = True
3807: (8)                    if redactions and found_redacts:
3808: (12)                       page.apply_redactions(images=redact_images)
3809: (8)                    if not (clean_pages or hidden_text):
3810: (12)                       continue  # done with the page
3811: (8)                    page.clean_contents()
3812: (8)                    if not page.get_contents():
3813: (12)                       continue
3814: (8)                    if hidden_text:
3815: (12)                       xref = page.get_contents()[0]  # only one b/o cleaning!
3816: (12)                       cont = doc.xref_stream(xref)
3817: (12)                       cont_lines = remove_hidden(cont.splitlines())  # remove hidden
text
3818: (12)                       if cont_lines:  # something was actually removed
3819: (16)                           cont = b"\n".join(cont_lines)
3820: (16)                           doc.update_stream(xref, cont)  # rewrite the page /Contents
3821: (8)                    if thumbnails:  # remove page thumbnails?
3822: (12)                       if doc.xref_get_key(page.xref, "Thumb")[0] != "null":
3823: (16)                           doc.xref_set_key(page.xref, "Thumb", "null")
3824: (4)                if embedded_files:
3825: (8)                    for name in doc.embfile_names():
3826: (12)                       doc.embfile_del(name)
3827: (4)                if xml_metadata:
3828: (8)                    doc.del_xml_metadata()
3829: (4)                if not (xml_metadata or javascript):
3830: (8)                    xref_limit = 0
3831: (4)                else:
3832: (8)                    xref_limit = doc.xref_length()
3833: (4)                for xref in range(1, xref_limit):
3834: (8)                    if not doc.xref_object(xref):
3835: (12)                       msg = "bad xref %i - clean PDF before scrubbing" % xref
3836: (12)                       raise ValueError(msg)
3837: (8)                    if javascript and doc.xref_get_key(xref, "S")[1] == "/JavaScript":
3838: (12)                       obj = "<</S/JavaScript/JS()>>"  # replace with a null JavaScript
3839: (12)                       doc.update_object(xref, obj)  # update this object
3840: (12)                       continue  # no further handling
3841: (8)                    if not xml_metadata:
3842: (12)                       continue
3843: (8)                    if doc.xref_get_key(xref, "Type")[1] == "/Metadata":
3844: (12)                       doc.update_object(xref, "<<>>")
3845: (12)                       doc.update_stream(xref, b"deleted", new=True)
3846: (12)                       continue
```

```
3847: (8)                   if doc.xref_get_key(xref, "Metadata")[0] != "null":
3848: (12)                      doc.xref_set_key(xref, "Metadata", "null")
3849: (0)           def _show_fz_text( text):
3850: (4)               num_spans = 0
3851: (4)               num_chars = 0
3852: (4)               span = text.m_internal.head
3853: (4)               while 1:
3854: (8)                   if not span:
3855: (12)                      break
3856: (8)                   num_spans += 1
3857: (8)                   num_chars += span.len
3858: (8)                   span = span.next
3859: (4)               return f'num_spans={num_spans} num_chars={num_chars}'
3860: (0)           def fill_textbox(
3861: (4)               writer: pymupdf.TextWriter,
3862: (4)               rect: rect_like,
3863: (4)               text: typing.Union[str, list],
3864: (4)               pos: point_like = None,
3865: (4)               font: typing.Optional[pymupdf.Font] = None,
3866: (4)               fontsize: float = 11,
3867: (4)               lineheight: OptFloat = None,
3868: (4)               align: int = 0,
3869: (4)               warn: bool = None,
3870: (4)               right_to_left: bool = False,
3871: (4)               small_caps: bool = False,
3872: (0)           ) -> tuple:
3873: (4)               """Fill a rectangle with text.
3874: (4)               Args:
3875: (8)                   writer: pymupdf.TextWriter object (= "self")
3876: (8)                   rect: rect-like to receive the text.
3877: (8)                   text: string or list/tuple of strings.
3878: (8)                   pos: point-like start position of first word.
3879: (8)                   font: pymupdf.Font object (default pymupdf.Font('helv')).
3880: (8)                   fontsize: the fontsize.
3881: (8)                   lineheight: overwrite the font property
3882: (8)                   align: (int) 0 = left, 1 = center, 2 = right, 3 = justify
3883: (8)                   warn: (bool) text overflow action: none, warn, or exception
3884: (8)                   right_to_left: (bool) indicate right-to-left language.
3885: (4)               """
3886: (4)               rect = pymupdf.Rect(rect)
3887: (4)               if rect.is_empty:
3888: (8)                   raise ValueError("fill rect must not empty.")
3889: (4)               if type(font) is not pymupdf.Font:
3890: (8)                   font = pymupdf.Font("helv")
3891: (4)               def textlen(x):
3892: (8)                   """Return length of a string."""
3893: (8)                   return font.text_length(
3894: (12)                      x, fontsize=fontsize, small_caps=small_caps
3895: (8)                   )  # abbreviation
3896: (4)               def char_lengths(x):
3897: (8)                   """Return list of single character lengths for a string."""
3898: (8)                   return font.char_lengths(x, fontsize=fontsize, small_caps=small_caps)
3899: (4)               def append_this(pos, text):
3900: (8)                   ret = writer.append(
3901: (16)                          pos, text, font=font, fontsize=fontsize, small_caps=small_caps
3902: (16)                          )
3903: (8)                   return ret
3904: (4)               tolerance = fontsize * 0.2  # extra distance to left border
3905: (4)               space_len = textlen(" ")
3906: (4)               std_width = rect.width - tolerance
3907: (4)               std_start = rect.x0 + tolerance
3908: (4)               def norm_words(width, words):
3909: (8)                   """Cut any word in pieces no longer than 'width'."""
3910: (8)                   nwords = []
3911: (8)                   word_lengths = []
3912: (8)                   for w in words:
3913: (12)                      wl_lst = char_lengths(w)
3914: (12)                      wl = sum(wl_lst)
3915: (12)                      if wl <= width:  # nothing to do - copy over
```

```
(16)                        nwords.append(w)
(16)                        word_lengths.append(wl)
(16)                        continue
(12)                    n = len(wl_lst)
(12)                    while n > 0:
(16)                        wl = sum(wl_lst[:n])
(16)                        if wl <= width:
(20)                            nwords.append(w[:n])
(20)                            word_lengths.append(wl)
(20)                            w = w[n:]
(20)                            wl_lst = wl_lst[n:]
(20)                            n = len(wl_lst)
(16)                        else:
(20)                            n -= 1
(8)                 return nwords, word_lengths
(4)             def output_justify(start, line):
(8)                 """Justified output of a line."""
(8)                 words = [w for w in line.split(" ") if w != ""]
(8)                 nwords = len(words)
(8)                 if nwords == 0:
(12)                    return
(8)                 if nwords == 1:  # single word cannot be justified
(12)                    append_this(start, words[0])
(12)                    return
(8)                 tl = sum([textlen(w) for w in words])  # total word lengths
(8)                 gaps = nwords - 1  # number of word gaps
(8)                 gapl = (std_width - tl) / gaps  # width of each gap
(8)                 for w in words:
(12)                    _, lp = append_this(start, w)  # output one word
(12)                    start.x = lp.x + gapl  # next start at word end plus gap
(8)                 return
(4)             asc = font.ascender
(4)             dsc = font.descender
(4)             if not lineheight:
(8)                 if asc - dsc <= 1:
(12)                    lheight = 1.2
(8)                 else:
(12)                    lheight = asc - dsc
(4)             else:
(8)                 lheight = lineheight
(4)             LINEHEIGHT = fontsize * lheight  # effective line height
(4)             width = std_width  # available horizontal space
(4)             if pos is not None:
(8)                 pos = pymupdf.Point(pos)
(4)             else:  # default is just below rect top-left
(8)                 pos = rect.tl + (tolerance, fontsize * asc)
(4)             if pos not in rect:
(8)                 raise ValueError("Text must start in rectangle.")
(4)             if align == pymupdf.TEXT_ALIGN_CENTER:
(8)                 factor = 0.5
(4)             elif align == pymupdf.TEXT_ALIGN_RIGHT:
(8)                 factor = 1.0
(4)             else:
(8)                 factor = 0
(4)             if type(text) is str:
(8)                 textlines = text.splitlines()
(4)             else:
(8)                 textlines = []
(8)                 for line in text:
(12)                    textlines.extend(line.splitlines())
(4)             max_lines = int((rect.y1 - pos.y) / LINEHEIGHT) + 1
(4)             new_lines = []  # the final list of textbox lines
(4)             no_justify = []  # no justify for these line numbers
(4)             for i, line in enumerate(textlines):
(8)                 if line in ("", " "):
(12)                    new_lines.append((line, space_len))
(12)                    width = rect.width - tolerance
(12)                    no_justify.append((len(new_lines) - 1))
(12)                    continue
```

```
3985: (8)                   if i == 0:
3986: (12)                      width = rect.x1 - pos.x
3987: (8)                   else:
3988: (12)                      width = rect.width - tolerance
3989: (8)                   if right_to_left:  # reverses Arabic / Hebrew text front to back
3990: (12)                      line = writer.clean_rtl(line)
3991: (8)                   tl = textlen(line)
3992: (8)                   if tl <= width:  # line short enough
3993: (12)                      new_lines.append((line, tl))
3994: (12)                      no_justify.append((len(new_lines) - 1))
3995: (12)                      continue
3996: (8)                   words = line.split(" ")  # the words in the line
3997: (8)                   words, word_lengths = norm_words(std_width, words)
3998: (8)                   n = len(words)
3999: (8)                   while True:
4000: (12)                      line0 = " ".join(words[:n])
4001: (12)                      wl = sum(word_lengths[:n]) + space_len * (len(word_lengths[:n]) -
1)
4002: (12)                      if wl <= width:
4003: (16)                          new_lines.append((line0, wl))
4004: (16)                          words = words[n:]
4005: (16)                          word_lengths = word_lengths[n:]
4006: (16)                          n = len(words)
4007: (16)                          line0 = None
4008: (12)                      else:
4009: (16)                          n -= 1
4010: (12)                      if len(words) == 0:
4011: (16)                          break
4012: (4)               nlines = len(new_lines)
4013: (4)               if nlines > max_lines:
4014: (8)                   msg = "Only fitting %i of %i lines." % (max_lines, nlines)
4015: (8)                   if warn is True:
4016: (12)                      pymupdf.message("Warning: " + msg)
4017: (8)                   elif warn is False:
4018: (12)                      raise ValueError(msg)
4019: (4)               start = pymupdf.Point()
4020: (4)               no_justify += [len(new_lines) - 1]  # no justifying of last line
4021: (4)               for i in range(max_lines):
4022: (8)                   try:
4023: (12)                      line, tl = new_lines.pop(0)
4024: (8)                   except IndexError:
4025: (12)                      if g_exceptions_verbose >= 2:   pymupdf.exception_info()
4026: (12)                      break
4027: (8)                   if right_to_left:  # Arabic, Hebrew
4028: (12)                      line = "".join(reversed(line))
4029: (8)                   if i == 0:  # may have different start for first line
4030: (12)                      start = pos
4031: (8)                   if align == pymupdf.TEXT_ALIGN_JUSTIFY and i not in no_justify and tl
< std_width:
4032: (12)                      output_justify(start, line)
4033: (12)                      start.x = std_start
4034: (12)                      start.y += LINEHEIGHT
4035: (12)                      continue
4036: (8)                   if i > 0 or pos.x == std_start:  # left, center, right alignments
4037: (12)                      start.x += (width - tl) * factor
4038: (8)                   append_this(start, line)
4039: (8)                   start.x = std_start
4040: (8)                   start.y += LINEHEIGHT
4041: (4)               return new_lines  # return non-written lines
4042: (0)           def get_oc(doc: pymupdf.Document, xref: int) -> int:
4043: (4)               """Return optional content object xref for an image or form xobject.
4044: (4)               Args:
4045: (8)                   xref: (int) xref number of an image or form xobject.
4046: (4)               """
4047: (4)               if doc.is_closed or doc.is_encrypted:
4048: (8)                   raise ValueError("document close or encrypted")
4049: (4)               t, name = doc.xref_get_key(xref, "Subtype")
4050: (4)               if t != "name" or name not in ("/Image", "/Form"):
4051: (8)                   raise ValueError("bad object type at xref %i" % xref)
```

```
4052: (4)               t, oc = doc.xref_get_key(xref, "OC")
4053: (4)               if t != "xref":
4054: (8)                   return 0
4055: (4)               rc = int(oc.replace("0 R", ""))
4056: (4)               return rc
4057: (0)           def set_oc(doc: pymupdf.Document, xref: int, oc: int) -> None:
4058: (4)               """Attach optional content object to image or form xobject.
4059: (4)               Args:
4060: (8)                   xref: (int) xref number of an image or form xobject
4061: (8)                   oc: (int) xref number of an OCG or OCMD
4062: (4)               """
4063: (4)               if doc.is_closed or doc.is_encrypted:
4064: (8)                   raise ValueError("document close or encrypted")
4065: (4)               t, name = doc.xref_get_key(xref, "Subtype")
4066: (4)               if t != "name" or name not in ("/Image", "/Form"):
4067: (8)                   raise ValueError("bad object type at xref %i" % xref)
4068: (4)               if oc > 0:
4069: (8)                   t, name = doc.xref_get_key(oc, "Type")
4070: (8)                   if t != "name" or name not in ("/OCG", "/OCMD"):
4071: (12)                      raise ValueError("bad object type at xref %i" % oc)
4072: (4)               if oc == 0 and "OC" in doc.xref_get_keys(xref):
4073: (8)                   doc.xref_set_key(xref, "OC", "null")
4074: (8)                   return None
4075: (4)               doc.xref_set_key(xref, "OC", "%i 0 R" % oc)
4076: (4)               return None
4077: (0)           def set_ocmd(
4078: (4)               doc: pymupdf.Document,
4079: (4)               xref: int = 0,
4080: (4)               ocgs: typing.Union[list, None] = None,
4081: (4)               policy: OptStr = None,
4082: (4)               ve: typing.Union[list, None] = None,
4083: (0)           ) -> int:
4084: (4)               """Create or update an OCMD object in a PDF document.
4085: (4)               Args:
4086: (8)                   xref: (int) 0 for creating a new object, otherwise update existing
one.
4087: (8)                   ocgs: (list) OCG xref numbers, which shall be subject to 'policy'.
4088: (8)                   policy: one of 'AllOn', 'AllOff', 'AnyOn', 'AnyOff' (any casing).
4089: (8)                   ve: (list) visibility expression. Use instead of 'ocgs' with 'policy'.
4090: (4)               Returns:
4091: (8)                   Xref of the created or updated OCMD.
4092: (4)               """
4093: (4)               all_ocgs = set(doc.get_ocgs().keys())
4094: (4)               def ve_maker(ve):
4095: (8)                   if type(ve) not in (list, tuple) or len(ve) < 2:
4096: (12)                      raise ValueError("bad 've' format: %s" % ve)
4097: (8)                   if ve[0].lower() not in ("and", "or", "not"):
4098: (12)                      raise ValueError("bad operand: %s" % ve[0])
4099: (8)                   if ve[0].lower() == "not" and len(ve) != 2:
4100: (12)                      raise ValueError("bad 've' format: %s" % ve)
4101: (8)                   item = "[/%s" % ve[0].title()
4102: (8)                   for x in ve[1:]:
4103: (12)                      if type(x) is int:
4104: (16)                          if x not in all_ocgs:
4105: (20)                              raise ValueError("bad OCG %i" % x)
4106: (16)                          item += " %i 0 R" % x
4107: (12)                      else:
4108: (16)                          item += " %s" % ve_maker(x)
4109: (8)                   item += "]"
4110: (8)                   return item
4111: (4)               text = "<<</Type/OCMD"
4112: (4)               if ocgs and type(ocgs) in (list, tuple):  # some OCGs are provided
4113: (8)                   s = set(ocgs).difference(all_ocgs)  # contains illegal xrefs
4114: (8)                   if s != set():
4115: (12)                      msg = "bad OCGs: %s" % s
4116: (12)                      raise ValueError(msg)
4117: (8)                   text += "/OCGs[" + " ".join(map(lambda x: "%i 0 R" % x, ocgs)) + "]"
4118: (4)               if policy:
4119: (8)                   policy = str(policy).lower()
```

```
4120: (8)                   pols = {
4121: (12)                      "anyon": "AnyOn",
4122: (12)                      "allon": "AllOn",
4123: (12)                      "anyoff": "AnyOff",
4124: (12)                      "alloff": "AllOff",
4125: (8)                   }
4126: (8)                   if policy not in ("anyon", "allon", "anyoff", "alloff"):
4127: (12)                      raise ValueError("bad policy: %s" % policy)
4128: (8)                   text += "/P/%s" % pols[policy]
4129: (4)               if ve:
4130: (8)                   text += "/VE%s" % ve_maker(ve)
4131: (4)               text += ">>"
4132: (4)               if xref == 0:
4133: (8)                   xref = doc.get_new_xref()
4134: (4)               elif "/Type/OCMD" not in doc.xref_object(xref, compressed=True):
4135: (8)                   raise ValueError("bad xref or not an OCMD")
4136: (4)               doc.update_object(xref, text)
4137: (4)               return xref
4138: (0)           def get_ocmd(doc: pymupdf.Document, xref: int) -> dict:
4139: (4)               """Return the definition of an OCMD (optional content membership
dictionary).
4140: (4)               Recognizes PDF dict keys /OCGs (PDF array of OCGs), /P (policy string) and
4141: (4)               /VE (visibility expression, PDF array). Via string manipulation, this
4142: (4)               info is converted to a Python dictionary with keys "xref", "ocgs",
"policy"
4143: (4)               and "ve" - ready to recycle as input for 'set_ocmd()'.
4144: (4)               """
4145: (4)               if xref not in range(doc.xref_length()):
4146: (8)                   raise ValueError("bad xref")
4147: (4)               text = doc.xref_object(xref, compressed=True)
4148: (4)               if "/Type/OCMD" not in text:
4149: (8)                   raise ValueError("bad object type")
4150: (4)               textlen = len(text)
4151: (4)               p0 = text.find("/OCGs[")  # look for /OCGs key
4152: (4)               p1 = text.find("]", p0)
4153: (4)               if p0 < 0 or p1 < 0:  # no OCGs found
4154: (8)                   ocgs = None
4155: (4)               else:
4156: (8)                   ocgs = text[p0 + 6 : p1].replace("0 R", " ").split()
4157: (8)                   ocgs = list(map(int, ocgs))
4158: (4)               p0 = text.find("/P/")  # look for /P policy key
4159: (4)               if p0 < 0:
4160: (8)                   policy = None
4161: (4)               else:
4162: (8)                   p1 = text.find("ff", p0)
4163: (8)                   if p1 < 0:
4164: (12)                      p1 = text.find("on", p0)
4165: (8)                   if p1 < 0:  # some irregular syntax
4166: (12)                      raise ValueError("bad object at xref")
4167: (8)                   else:
4168: (12)                      policy = text[p0 + 3 : p1 + 2]
4169: (4)               p0 = text.find("/VE[")  # look for /VE visibility expression key
4170: (4)               if p0 < 0:  # no visibility expression found
4171: (8)                   ve = None
4172: (4)               else:
4173: (8)                   lp = rp = 0  # find end of /VE by finding last ']'.
4174: (8)                   p1 = p0
4175: (8)                   while lp < 1 or lp != rp:
4176: (12)                      p1 += 1
4177: (12)                      if not p1 < textlen:  # some irregular syntax
4178: (16)                          raise ValueError("bad object at xref")
4179: (12)                      if text[p1] == "[":
4180: (16)                          lp += 1
4181: (12)                      if text[p1] == "]":
4182: (16)                          rp += 1
4183: (8)                   ve = text[p0 + 3 : p1 + 1]  # the PDF /VE array
4184: (8)                   ve = (
4185: (12)                      ve.replace("/And", '"and",')
4186: (12)                      .replace("/Not", '"not",')
```

```
(12)                    .replace("/Or", '"or",')
(8)                 )
(8)                 ve = ve.replace(" 0 R]", "]").replace(" 0 R", ",").replace("][", "],
(8)                 import json
(8)                 try:
(12)                    ve = json.loads(ve)
(8)                 except Exception:
(12)                    pymupdf.exception_info()
(12)                    pymupdf.message(f"bad /VE key: {ve!r}")
(12)                    raise
(4)             return {"xref": xref, "ocgs": ocgs, "policy": policy, "ve": ve}
(0)         """
(0)         Handle page labels for PDF documents.
(0)         Reading
(0)         -------
(0)         * compute the label of a page
(0)         * find page number(s) having the given label.
(0)         Writing
(0)         -------
(0)         Supports setting (defining) page labels for PDF documents.
(0)         A big Thank You goes to WILLIAM CHAPMAN who contributed the idea and
(0)         significant parts of the following code during late December 2020
(0)         through early January 2021.
(0)         """
(0)         def rule_dict(item):
(4)             """Make a Python dict from a PDF page label rule.
(4)             Args:
(8)                 item -- a tuple (pno, rule) with the start page number and the rule
(16)                        string like <<</S/D...>>>.
(4)             Returns:
(8)                 A dict like
(8)                 {'startpage': int, 'prefix': str, 'style': str, 'firstpagenum': int}.
(4)             """
(4)             pno, rule = item
(4)             rule = rule[2:-2].split("/")[1:]  # strip "<<" and ">>"
(4)             d = {"startpage": pno, "prefix": "", "firstpagenum": 1}
(4)             skip = False
(4)             for i, item in enumerate(rule): # pylint: disable=redefined-argument-from-
local
(8)                 if skip:  # this item has already been processed
(12)                    skip = False  # deactivate skipping again
(12)                    continue
(8)                 if item == "S":  # style specification
(12)                    d["style"] = rule[i + 1]  # next item has the style
(12)                    skip = True  # do not process next item again
(12)                    continue
(8)                 if item.startswith("P"):  # prefix specification: extract the string
(12)                    x = item[1:].replace("(", "").replace(")", "")
(12)                    d["prefix"] = x
(12)                    continue
(8)                 if item.startswith("St"):  # start page number specification
(12)                    x = int(item[2:])
(12)                    d["firstpagenum"] = x
(4)             return d
(0)         def get_label_pno(pgNo, labels):
(4)             """Return the label for this page number.
(4)             Args:
(8)                 pgNo: page number, 0-based.
(8)                 labels: result of doc._get_page_labels().
(4)             Returns:
(8)                 The label (str) of the page number. Errors return an empty string.
(4)             """
(4)             item = [x for x in labels if x[0] <= pgNo][-1]
(4)             rule = rule_dict(item)
(4)             prefix = rule.get("prefix", "")
(4)             style = rule.get("style", "")
(4)             delta = -1 if style in ("a", "A") else 0
(4)             pagenumber = pgNo - rule["startpage"] + rule["firstpagenum"] + delta
```

```
(4)             return construct_label(style, prefix, pagenumber)
(0)         def get_label(page):
(4)             """Return the label for this PDF page.
(4)             Args:
(8)                 page: page object.
(4)             Returns:
(8)                 The label (str) of the page. Errors return an empty string.
(4)             """
(4)             labels = page.parent._get_page_labels()
(4)             if not labels:
(8)                 return ""
(4)             labels.sort()
(4)             return get_label_pno(page.number, labels)
(0)         def get_page_numbers(doc, label, only_one=False):
(4)             """Return a list of page numbers with the given label.
(4)             Args:
(8)                 doc: PDF document object (resp. 'self').
(8)                 label: (str) label.
(8)                 only_one: (bool) stop searching after first hit.
(4)             Returns:
(8)                 List of page numbers having this label.
(4)             """
(4)             numbers = []
(4)             if not label:
(8)                 return numbers
(4)             labels = doc._get_page_labels()
(4)             if labels == []:
(8)                 return numbers
(4)             for i in range(doc.page_count):
(8)                 plabel = get_label_pno(i, labels)
(8)                 if plabel == label:
(12)                    numbers.append(i)
(12)                    if only_one:
(16)                        break
(4)             return numbers
(0)         def construct_label(style, prefix, pno) -> str:
(4)             """Construct a label based on style, prefix and page number."""
(4)             n_str = ""
(4)             if style == "D":
(8)                 n_str = str(pno)
(4)             elif style == "r":
(8)                 n_str = integerToRoman(pno).lower()
(4)             elif style == "R":
(8)                 n_str = integerToRoman(pno).upper()
(4)             elif style == "a":
(8)                 n_str = integerToLetter(pno).lower()
(4)             elif style == "A":
(8)                 n_str = integerToLetter(pno).upper()
(4)             result = prefix + n_str
(4)             return result
(0)         def integerToLetter(i) -> str:
(4)             """Returns letter sequence string for integer i."""
(4)             import string
(4)             ls = string.ascii_uppercase
(4)             n, a = 1, i
(4)             while pow(26, n) <= a:
(8)                 a -= int(math.pow(26, n))
(8)                 n += 1
(4)             str_t = ""
(4)             for j in reversed(range(n)):
(8)                 f, g = divmod(a, int(math.pow(26, j)))
(8)                 str_t += ls[f]
(8)                 a = g
(4)             return str_t
(0)         def integerToRoman(num: int) -> str:
(4)             """Return roman numeral for an integer."""
(4)             roman = (
(8)                 (1000, "M"),
(8)                 (900, "CM"),
```

```
4323: (8)                   (500, "D"),
4324: (8)                   (400, "CD"),
4325: (8)                   (100, "C"),
4326: (8)                   (90, "XC"),
4327: (8)                   (50, "L"),
4328: (8)                   (40, "XL"),
4329: (8)                   (10, "X"),
4330: (8)                   (9, "IX"),
4331: (8)                   (5, "V"),
4332: (8)                   (4, "IV"),
4333: (8)                   (1, "I"),
4334: (4)               )
4335: (4)               def roman_num(num):
4336: (8)                   for r, ltr in roman:
4337: (12)                      x, _ = divmod(num, r)
4338: (12)                      yield ltr * x
4339: (12)                      num -= r * x
4340: (12)                      if num <= 0:
4341: (16)                          break
4342: (4)               return "".join([a for a in roman_num(num)])
4343: (0)           def get_page_labels(doc):
4344: (4)               """Return page label definitions in PDF document.
4345: (4)               Args:
4346: (8)                   doc: PDF document (resp. 'self').
4347: (4)               Returns:
4348: (8)                   A list of dictionaries with the following format:
4349: (8)                   {'startpage': int, 'prefix': str, 'style': str, 'firstpagenum': int}.
4350: (4)               """
4351: (4)               return [rule_dict(item) for item in doc._get_page_labels()]
4352: (0)           def set_page_labels(doc, labels):
4353: (4)               """Add / replace page label definitions in PDF document.
4354: (4)               Args:
4355: (8)                   doc: PDF document (resp. 'self').
4356: (8)                   labels: list of label dictionaries like:
4357: (8)                   {'startpage': int, 'prefix': str, 'style': str, 'firstpagenum': int},
4358: (8)                   as returned by get_page_labels().
4359: (4)               """
4360: (4)               def create_label_str(label):
4361: (8)                   """Convert Python label dict to corresponding PDF rule string.
4362: (8)                   Args:
4363: (12)                      label: (dict) build rule for the label.
4364: (8)                   Returns:
4365: (12)                      PDF label rule string wrapped in "<<", ">>".
4366: (8)                   """
4367: (8)                   s = "%i<<" % label["startpage"]
4368: (8)                   if label.get("prefix", "") != "":
4369: (12)                      s += "/P(%s)" % label["prefix"]
4370: (8)                   if label.get("style", "") != "":
4371: (12)                      s += "/S/%s" % label["style"]
4372: (8)                   if label.get("firstpagenum", 1) > 1:
4373: (12)                      s += "/St %i" % label["firstpagenum"]
4374: (8)                   s += ">>"
4375: (8)                   return s
4376: (4)               def create_nums(labels):
4377: (8)                   """Return concatenated string of all labels rules.
4378: (8)                   Args:
4379: (12)                      labels: (list) dictionaries as created by function 'rule_dict'.
4380: (8)                   Returns:
4381: (12)                      PDF compatible string for page label definitions, ready to be
4382: (12)                      enclosed in PDF array 'Nums[...]'.
4383: (8)                   """
4384: (8)                   labels.sort(key=lambda x: x["startpage"])
4385: (8)                   s = "".join([create_label_str(label) for label in labels])
4386: (8)                   return s
4387: (4)               doc._set_page_labels(create_nums(labels))
4388: (0)           def has_links(doc: pymupdf.Document) -> bool:
4389: (4)               """Check whether there are links on any page."""
4390: (4)               if doc.is_closed:
4391: (8)                   raise ValueError("document closed")
```

```
4392: (4)               if not doc.is_pdf:
4393: (8)                   raise ValueError("is no PDF")
4394: (4)               for i in range(doc.page_count):
4395: (8)                   for item in doc.page_annot_xrefs(i):
4396: (12)                      if item[1] == pymupdf.PDF_ANNOT_LINK:  # pylint: disable=no-member
4397: (16)                          return True
4398: (4)               return False
4399: (0)           def has_annots(doc: pymupdf.Document) -> bool:
4400: (4)               """Check whether there are annotations on any page."""
4401: (4)               if doc.is_closed:
4402: (8)                   raise ValueError("document closed")
4403: (4)               if not doc.is_pdf:
4404: (8)                   raise ValueError("is no PDF")
4405: (4)               for i in range(doc.page_count):
4406: (8)                   for item in doc.page_annot_xrefs(i):
4407: (12)                      if not (item[1] == pymupdf.PDF_ANNOT_LINK or item[1] ==
pymupdf.PDF_ANNOT_WIDGET):
4408: (16)                          return True
4409: (4)               return False
4410: (0)           def recover_bbox_quad(line_dir: tuple, span: dict, bbox: tuple) ->
pymupdf.Quad:
4411: (4)               """Compute the quad located inside the bbox.
4412: (4)               The bbox may be any of the resp. tuples occurring inside the given span.
4413: (4)               Args:
4414: (8)                   line_dir: (tuple) 'line["dir"]' of the owning line or None.
4415: (8)                   span: (dict) the span. May be from get_texttrace() method.
4416: (8)                   bbox: (tuple) the bbox of the span or any of its characters.
4417: (4)               Returns:
4418: (8)                   The quad which is wrapped by the bbox.
4419: (4)               """
4420: (4)               if line_dir is None:
4421: (8)                   line_dir = span["dir"]
4422: (4)               cos, sin = line_dir
4423: (4)               bbox = pymupdf.Rect(bbox)  # make it a rect
4424: (4)               if pymupdf.TOOLS.set_small_glyph_heights():  # ==> just fontsize as height
4425: (8)                   d = 1
4426: (4)               else:
4427: (8)                   d = span["ascender"] - span["descender"]
4428: (4)               height = d * span["size"]  # the quad's rectangle height
4429: (4)               hs = height * sin
4430: (4)               hc = height * cos
4431: (4)               if hc >= 0 and hs <= 0:  # quadrant 1
4432: (8)                   ul = bbox.bl - (0, hc)
4433: (8)                   ur = bbox.tr + (hs, 0)
4434: (8)                   ll = bbox.bl - (hs, 0)
4435: (8)                   lr = bbox.tr + (0, hc)
4436: (4)               elif hc <= 0 and hs <= 0:  # quadrant 2
4437: (8)                   ul = bbox.br + (hs, 0)
4438: (8)                   ur = bbox.tl - (0, hc)
4439: (8)                   ll = bbox.br + (0, hc)
4440: (8)                   lr = bbox.tl - (hs, 0)
4441: (4)               elif hc <= 0 and hs >= 0:  # quadrant 3
4442: (8)                   ul = bbox.tr - (0, hc)
4443: (8)                   ur = bbox.bl + (hs, 0)
4444: (8)                   ll = bbox.tr - (hs, 0)
4445: (8)                   lr = bbox.bl + (0, hc)
4446: (4)               else:  # quadrant 4
4447: (8)                   ul = bbox.tl + (hs, 0)
4448: (8)                   ur = bbox.br - (0, hc)
4449: (8)                   ll = bbox.tl + (0, hc)
4450: (8)                   lr = bbox.br - (hs, 0)
4451: (4)               return pymupdf.Quad(ul, ur, ll, lr)
4452: (0)           def recover_quad(line_dir: tuple, span: dict) -> pymupdf.Quad:
4453: (4)               """Recover the quadrilateral of a text span.
4454: (4)               Args:
4455: (8)                   line_dir: (tuple) 'line["dir"]' of the owning line.
4456: (8)                   span: the span.
4457: (4)               Returns:
4458: (8)                   The quadrilateral enveloping the span's text.
```

```
4459: (4)                """
4460: (4)                if type(line_dir) is not tuple or len(line_dir) != 2:
4461: (8)                    raise ValueError("bad line dir argument")
4462: (4)                if type(span) is not dict:
4463: (8)                    raise ValueError("bad span argument")
4464: (4)                return recover_bbox_quad(line_dir, span, span["bbox"])
4465: (0)            def recover_line_quad(line: dict, spans: list = None) -> pymupdf.Quad:
4466: (4)                """Calculate the line quad for 'dict' / 'rawdict' text extractions.
4467: (4)                The lower quad points are those of the first, resp. last span quad.
4468: (4)                The upper points are determined by the maximum span quad height.
4469: (4)                From this, compute a rect with bottom-left in (0, 0), convert this to a
4470: (4)                quad and rotate and shift back to cover the text of the spans.
4471: (4)                Args:
4472: (8)                    spans: (list, optional) sub-list of spans to consider.
4473: (4)                Returns:
4474: (8)                    pymupdf.Quad covering selected spans.
4475: (4)                """
4476: (4)                if spans is None:  # no sub-selection
4477: (8)                    spans = line["spans"]  # all spans
4478: (4)                if len(spans) == 0:
4479: (8)                    raise ValueError("bad span list")
4480: (4)                line_dir = line["dir"]  # text direction
4481: (4)                cos, sin = line_dir
4482: (4)                q0 = recover_quad(line_dir, spans[0])  # quad of first span
4483: (4)                if len(spans) > 1:  # get quad of last span
4484: (8)                    q1 = recover_quad(line_dir, spans[-1])
4485: (4)                else:
4486: (8)                    q1 = q0  # last = first
4487: (4)                line_ll = q0.ll  # lower-left of line quad
4488: (4)                line_lr = q1.lr  # lower-right of line quad
4489: (4)                mat0 = pymupdf.planish_line(line_ll, line_lr)
4490: (4)                x_lr = line_lr * mat0
4491: (4)                small = pymupdf.TOOLS.set_small_glyph_heights()  # small glyph heights?
4492: (4)                h = max(
4493: (8)                    [s["size"] * (1 if small else (s["ascender"] - s["descender"])) for s
in spans]
4494: (4)                )
4495: (4)                line_rect = pymupdf.Rect(0, -h, x_lr.x, 0)  # line rectangle
4496: (4)                line_quad = line_rect.quad  # make it a quad and:
4497: (4)                line_quad *= ~mat0
4498: (4)                return line_quad
4499: (0)            def recover_span_quad(line_dir: tuple, span: dict, chars: list = None) ->
pymupdf.Quad:
4500: (4)                """Calculate the span quad for 'dict' / 'rawdict' text extractions.
4501: (4)                Notes:
4502: (8)                    There are two execution paths:
4503: (8)                    1. For the full span quad, the result of 'recover_quad' is returned.
4504: (8)                    2. For the quad of a sub-list of characters, the char quads are
4505: (11)                      computed and joined. This is only supported for the "rawdict"
4506: (11)                      extraction option.
4507: (4)                Args:
4508: (8)                    line_dir: (tuple) 'line["dir"]' of the owning line.
4509: (8)                    span: (dict) the span.
4510: (8)                    chars: (list, optional) sub-list of characters to consider.
4511: (4)                Returns:
4512: (8)                    pymupdf.Quad covering selected characters.
4513: (4)                """
4514: (4)                if line_dir is None:  # must be a span from get_texttrace()
4515: (8)                    line_dir = span["dir"]
4516: (4)                if chars is None:  # no sub-selection
4517: (8)                    return recover_quad(line_dir, span)
4518: (4)                if "chars" not in span.keys():
4519: (8)                    raise ValueError("need 'rawdict' option to sub-select chars")
4520: (4)                q0 = recover_char_quad(line_dir, span, chars[0])  # quad of first char
4521: (4)                if len(chars) > 1:  # get quad of last char
4522: (8)                    q1 = recover_char_quad(line_dir, span, chars[-1])
4523: (4)                else:
4524: (8)                    q1 = q0  # last = first
4525: (4)                span_ll = q0.ll  # lower-left of span quad
```

```
4526: (4)               span_lr = q1.lr  # lower-right of span quad
4527: (4)               mat0 = pymupdf.planish_line(span_ll, span_lr)
4528: (4)               x_lr = span_lr * mat0
4529: (4)               small = pymupdf.TOOLS.set_small_glyph_heights()  # small glyph heights?
4530: (4)               h = span["size"] * (1 if small else (span["ascender"] -
span["descender"]))
4531: (4)               span_rect = pymupdf.Rect(0, -h, x_lr.x, 0)  # line rectangle
4532: (4)               span_quad = span_rect.quad  # make it a quad and:
4533: (4)               span_quad *= ~mat0  # rotate back and shift back
4534: (4)               return span_quad
4535: (0)           def recover_char_quad(line_dir: tuple, span: dict, char: dict) ->
pymupdf.Quad:
4536: (4)               """Recover the quadrilateral of a text character.
4537: (4)               This requires the "rawdict" option of text extraction.
4538: (4)               Args:
4539: (8)                   line_dir: (tuple) 'line["dir"]' of the span's line.
4540: (8)                   span: (dict) the span dict.
4541: (8)                   char: (dict) the character dict.
4542: (4)               Returns:
4543: (8)                   The quadrilateral enveloping the character.
4544: (4)               """
4545: (4)               if line_dir is None:
4546: (8)                   line_dir = span["dir"]
4547: (4)               if type(line_dir) is not tuple or len(line_dir) != 2:
4548: (8)                   raise ValueError("bad line dir argument")
4549: (4)               if type(span) is not dict:
4550: (8)                   raise ValueError("bad span argument")
4551: (4)               if type(char) is dict:
4552: (8)                   bbox = pymupdf.Rect(char["bbox"])
4553: (4)               elif type(char) is tuple:
4554: (8)                   bbox = pymupdf.Rect(char[3])
4555: (4)               else:
4556: (8)                   raise ValueError("bad span argument")
4557: (4)               return recover_bbox_quad(line_dir, span, bbox)
4558: (0)           def subset_fonts(doc: pymupdf.Document, verbose: bool = False, fallback: bool
= False) -> None:
4559: (4)               """Build font subsets of a PDF. Requires package 'fontTools'.
4560: (4)               Eligible fonts are potentially replaced by smaller versions. Page text is
4561: (4)               NOT rewritten and thus should retain properties like being hidden or
4562: (4)               controlled by optional content.
4563: (4)               This method by default uses MuPDF's own internal feature to create subset
4564: (4)               fonts. As this is a new function, errors may still occur. In this case,
4565: (4)               please fall back to using the previous version by using "fallback=True".
4566: (4)               """
4567: (4)               if fallback is False:  # by default use MuPDF function
4568: (8)                   pdf = mupdf.pdf_document_from_fz_document(doc)
4569: (8)                   mupdf.pdf_subset_fonts2(pdf, list(range(doc.page_count)))
4570: (8)                   return
4571: (4)               font_buffers = {}
4572: (4)               def get_old_widths(xref):
4573: (8)                   """Retrieve old font '/W' and '/DW' values."""
4574: (8)                   df = doc.xref_get_key(xref, "DescendantFonts")
4575: (8)                   if df[0] != "array":  # only handle xref specifications
4576: (12)                      return None, None
4577: (8)                   df_xref = int(df[1][1:-1].replace("0 R", ""))
4578: (8)                   widths = doc.xref_get_key(df_xref, "W")
4579: (8)                   if widths[0] != "array":  # no widths key found
4580: (12)                      widths = None
4581: (8)                   else:
4582: (12)                      widths = widths[1]
4583: (8)                   dwidths = doc.xref_get_key(df_xref, "DW")
4584: (8)                   if dwidths[0] != "int":
4585: (12)                      dwidths = None
4586: (8)                   else:
4587: (12)                      dwidths = dwidths[1]
4588: (8)                   return widths, dwidths
4589: (4)               def set_old_widths(xref, widths, dwidths):
4590: (8)                   """Restore the old '/W' and '/DW' in subsetted font.
4591: (8)                   If either parameter is None or evaluates to False, the corresponding
```

```
4592: (8)                   dictionary key will be set to null.
4593: (8)                   """
4594: (8)                   df = doc.xref_get_key(xref, "DescendantFonts")
4595: (8)                   if df[0] != "array":  # only handle xref specs
4596: (12)                      return None
4597: (8)                   df_xref = int(df[1][1:-1].replace("0 R", ""))
4598: (8)                   if (type(widths) is not str or not widths) and
doc.xref_get_key(df_xref, "W")[
4599: (12)                      0
4600: (8)                   ] != "null":
4601: (12)                      doc.xref_set_key(df_xref, "W", "null")
4602: (8)                   else:
4603: (12)                      doc.xref_set_key(df_xref, "W", widths)
4604: (8)                   if (type(dwidths) is not str or not dwidths) and doc.xref_get_key(
4605: (12)                      df_xref, "DW"
4606: (8)                   )[0] != "null":
4607: (12)                      doc.xref_set_key(df_xref, "DW", "null")
4608: (8)                   else:
4609: (12)                      doc.xref_set_key(df_xref, "DW", dwidths)
4610: (8)                   return None
4611: (4)               def set_subset_fontname(new_xref):
4612: (8)                   """Generate a name prefix to tag a font as subset.
4613: (8)                   We use a random generator to select 6 upper case ASCII characters.
4614: (8)                   The prefixed name must be put in the font xref as the "/BaseFont"
value
4615: (8)                   and in the FontDescriptor object as the '/FontName' value.
4616: (8)                   """
4617: (8)                   import random
4618: (8)                   import string
4619: (8)                   prefix = "".join(random.choices(tuple(string.ascii_uppercase), k=6)) +
"+"
4620: (8)                   font_str = doc.xref_object(new_xref, compressed=True)
4621: (8)                   font_str = font_str.replace("/BaseFont/", "/BaseFont/" + prefix)
4622: (8)                   df = doc.xref_get_key(new_xref, "DescendantFonts")
4623: (8)                   if df[0] == "array":
4624: (12)                      df_xref = int(df[1][1:-1].replace("0 R", ""))
4625: (12)                      fd = doc.xref_get_key(df_xref, "FontDescriptor")
4626: (12)                      if fd[0] == "xref":
4627: (16)                          fd_xref = int(fd[1].replace("0 R", ""))
4628: (16)                          fd_str = doc.xref_object(fd_xref, compressed=True)
4629: (16)                          fd_str = fd_str.replace("/FontName/", "/FontName/" + prefix)
4630: (16)                          doc.update_object(fd_xref, fd_str)
4631: (8)                   doc.update_object(new_xref, font_str)
4632: (4)               def build_subset(buffer, unc_set, gid_set):
4633: (8)                   """Build font subset using fontTools.
4634: (8)                   Args:
4635: (12)                      buffer: (bytes) the font given as a binary buffer.
4636: (12)                      unc_set: (set) required glyph ids.
4637: (8)                   Returns:
4638: (12)                      Either None if subsetting is unsuccessful or the subset font
buffer.
4639: (8)                   """
4640: (8)                   try:
4641: (12)                      import fontTools.subset as fts
4642: (8)                   except ImportError:
4643: (12)                      if g_exceptions_verbose:    pymupdf.exception_info()
4644: (12)                      pymupdf.message("This method requires fontTools to be installed.")
4645: (12)                      raise
4646: (8)                   import tempfile
4647: (8)                   with tempfile.TemporaryDirectory() as tmp_dir:
4648: (12)                      oldfont_path = f"{tmp_dir}/oldfont.ttf"
4649: (12)                      newfont_path = f"{tmp_dir}/newfont.ttf"
4650: (12)                      uncfile_path = f"{tmp_dir}/uncfile.txt"
4651: (12)                      args = [
4652: (16)                          oldfont_path,
4653: (16)                          "--retain-gids",
4654: (16)                          f"--output-file={newfont_path}",
4655: (16)                          "--layout-features='*'",
4656: (16)                          "--passthrough-tables",
```

```
4657: (16)                          "--ignore-missing-glyphs",
4658: (16)                          "--ignore-missing-unicodes",
4659: (16)                          "--symbol-cmap",
4660: (12)                      ]
4661: (12)                      with open(f"{tmp_dir}/uncfile.txt", "w", encoding='utf8') as
unc_file:
4662: (16)                          if 0xFFFD in unc_set:  # error unicode exists -> use glyphs
4663: (20)                              args.append(f"--gids-file={uncfile_path}")
4664: (20)                              gid_set.add(189)
4665: (20)                              unc_list = list(gid_set)
4666: (20)                              for unc in unc_list:
4667: (24)                                  unc_file.write("%i\n" % unc)
4668: (16)                          else:
4669: (20)                              args.append(f"--unicodes-file={uncfile_path}")
4670: (20)                              unc_set.add(255)
4671: (20)                              unc_list = list(unc_set)
4672: (20)                              for unc in unc_list:
4673: (24)                                  unc_file.write("%04x\n" % unc)
4674: (12)                      with open(oldfont_path, "wb") as fontfile:
4675: (16)                          fontfile.write(buffer)
4676: (12)                      try:
4677: (16)                          os.remove(newfont_path)  # remove old file
4678: (12)                      except Exception:
4679: (16)                          pass
4680: (12)                      try:  # invoke fontTools subsetter
4681: (16)                          fts.main(args)
4682: (16)                          font = pymupdf.Font(fontfile=newfont_path)
4683: (16)                          new_buffer = font.buffer  # subset font binary
4684: (16)                          if font.glyph_count == 0:  # intercept empty font
4685: (20)                              new_buffer = None
4686: (12)                      except Exception:
4687: (16)                          pymupdf.exception_info()
4688: (16)                          new_buffer = None
4689: (8)                   return new_buffer
4690: (4)               def repl_fontnames(doc):
4691: (8)                   """Populate 'font_buffers'.
4692: (8)                   For each font candidate, store its xref and the list of names
4693: (8)                   by which PDF text may refer to it (there may be multiple).
4694: (8)                   """
4695: (8)                   def norm_name(name):
4696: (12)                      """Recreate font name that contains PDF hex codes.
4697: (12)                      E.g. #20 -> space, chr(32)
4698: (12)                      """
4699: (12)                      while "#" in name:
4700: (16)                          p = name.find("#")
4701: (16)                          c = int(name[p + 1 : p + 3], 16)
4702: (16)                          name = name.replace(name[p : p + 3], chr(c))
4703: (12)                      return name
4704: (8)                   def get_fontnames(doc, item):
4705: (12)                      """Return a list of fontnames for an item of page.get_fonts().
4706: (12)                      There may be multiple names e.g. for Type0 fonts.
4707: (12)                      """
4708: (12)                      fontname = item[3]
4709: (12)                      names = [fontname]
4710: (12)                      fontname = doc.xref_get_key(item[0], "BaseFont")[1][1:]
4711: (12)                      fontname = norm_name(fontname)
4712: (12)                      if fontname not in names:
4713: (16)                          names.append(fontname)
4714: (12)                      descendents = doc.xref_get_key(item[0], "DescendantFonts")
4715: (12)                      if descendents[0] != "array":
4716: (16)                          return names
4717: (12)                      descendents = descendents[1][1:-1]
4718: (12)                      if descendents.endswith(" 0 R"):
4719: (16)                          xref = int(descendents[:-4])
4720: (16)                          descendents = doc.xref_object(xref, compressed=True)
4721: (12)                      p1 = descendents.find("/BaseFont")
4722: (12)                      if p1 >= 0:
4723: (16)                          p2 = descendents.find("/", p1 + 1)
4724: (16)                          p1 = min(descendents.find("/", p2 + 1), descendents.find(">>",
```

```
p2 + 1))
4725: (16)                          fontname = descendents[p2 + 1 : p1]
4726: (16)                          fontname = norm_name(fontname)
4727: (16)                          if fontname not in names:
4728: (20)                              names.append(fontname)
4729: (12)                      return names
4730: (8)                   for i in range(doc.page_count):
4731: (12)                      for f in doc.get_page_fonts(i, full=True):
4732: (16)                          font_xref = f[0]  # font xref
4733: (16)                          font_ext = f[1]  # font file extension
4734: (16)                          basename = f[3]  # font basename
4735: (16)                          if font_ext not in (  # skip if not supported by fontTools
4736: (20)                              "otf",
4737: (20)                              "ttf",
4738: (20)                              "woff",
4739: (20)                              "woff2",
4740: (16)                          ):
4741: (20)                              continue
4742: (16)                          if len(basename) > 6 and basename[6] == "+":
4743: (20)                              continue
4744: (16)                          extr = doc.extract_font(font_xref)
4745: (16)                          fontbuffer = extr[-1]
4746: (16)                          names = get_fontnames(doc, f)
4747: (16)                          name_set, xref_set, subsets = font_buffers.get(
4748: (20)                              fontbuffer, (set(), set(), (set(), set()))
4749: (16)                          )
4750: (16)                          xref_set.add(font_xref)
4751: (16)                          for name in names:
4752: (20)                              name_set.add(name)
4753: (16)                          font = pymupdf.Font(fontbuffer=fontbuffer)
4754: (16)                          name_set.add(font.name)
4755: (16)                          del font
4756: (16)                          font_buffers[fontbuffer] = (name_set, xref_set, subsets)
4757: (4)               def find_buffer_by_name(name):
4758: (8)                   for buffer, (name_set, _, _) in font_buffers.items():
4759: (12)                      if name in name_set:
4760: (16)                          return buffer
4761: (8)                   return None
4762: (4)               repl_fontnames(doc)  # populate font information
4763: (4)               if not font_buffers:  # nothing found to do
4764: (8)                   if verbose:
4765: (12)                      pymupdf.message(f'No fonts to subset.')
4766: (8)                   return 0
4767: (4)               old_fontsize = 0
4768: (4)               new_fontsize = 0
4769: (4)               for fontbuffer in font_buffers.keys():
4770: (8)                   old_fontsize += len(fontbuffer)
4771: (4)               for page in doc:
4772: (8)                   for span in page.get_texttrace():
4773: (12)                      if type(span) is not dict:  # skip useless information
4774: (16)                          continue
4775: (12)                      fontname = span["font"][:33]  # fontname for the span
4776: (12)                      buffer = find_buffer_by_name(fontname)
4777: (12)                      if buffer is None:
4778: (16)                          continue
4779: (12)                      name_set, xref_set, (set_ucs, set_gid) = font_buffers[buffer]
4780: (12)                      for c in span["chars"]:
4781: (16)                          set_ucs.add(c[0])  # unicode
4782: (16)                          set_gid.add(c[1])  # glyph id
4783: (12)                      font_buffers[buffer] = (name_set, xref_set, (set_ucs, set_gid))
4784: (4)               for old_buffer, (name_set, xref_set, subsets) in font_buffers.items():
4785: (8)                   new_buffer = build_subset(old_buffer, subsets[0], subsets[1])
4786: (8)                   fontname = list(name_set)[0]
4787: (8)                   if new_buffer is None or len(new_buffer) >= len(old_buffer):
4788: (12)                      if verbose:
4789: (16)                          pymupdf.message(f'Cannot subset {fontname!r}.')
4790: (12)                      continue
4791: (8)                   if verbose:
4792: (12)                      pymupdf.message(f"Built subset of font {fontname!r}.")
```

```
4793: (8)                   val = doc._insert_font(fontbuffer=new_buffer)  # store subset font in
PDF
4794: (8)                   new_xref = val[0]  # get its xref
4795: (8)                   set_subset_fontname(new_xref)  # tag fontname as subset font
4796: (8)                   font_str = doc.xref_object(  # get its object definition
4797: (12)                      new_xref,
4798: (12)                      compressed=True,
4799: (8)                   )
4800: (8)                   for font_xref in xref_set:
4801: (12)                      width_table, def_width = get_old_widths(font_xref)
4802: (12)                      doc.update_object(font_xref, font_str)
4803: (12)                      if width_table or def_width:
4804: (16)                          set_old_widths(font_xref, width_table, def_width)
4805: (8)                   new_fontsize += len(new_buffer)
4806: (4)               return old_fontsize - new_fontsize
4807: (0)           def xref_copy(doc: pymupdf.Document, source: int, target: int, *, keep: list =
None) -> None:
4808: (4)               """Copy a PDF dictionary object to another one given their xref numbers.
4809: (4)               Args:
4810: (8)                   doc: PDF document object
4811: (8)                   source: source xref number
4812: (8)                   target: target xref number, the xref must already exist
4813: (8)                   keep: an optional list of 1st level keys in target that should not be
4814: (14)                        removed before copying.
4815: (4)               Notes:
4816: (8)                   This works similar to the copy() method of dictionaries in Python. The
4817: (8)                   source may be a stream object.
4818: (4)               """
4819: (4)               if doc.xref_is_stream(source):
4820: (8)                   stream = doc.xref_stream_raw(source)
4821: (8)                   doc.update_stream(
4822: (12)                      target,
4823: (12)                      stream,
4824: (12)                      compress=False,  # keeps source compression
4825: (12)                      new=True,  # in case target is no stream
4826: (8)                   )
4827: (4)               if keep is None:
4828: (8)                   keep = []
4829: (4)               for key in doc.xref_get_keys(target):
4830: (8)                   if key in keep:
4831: (12)                      continue
4832: (8)                   doc.xref_set_key(target, key, "null")
4833: (4)               for key in doc.xref_get_keys(source):
4834: (8)                   item = doc.xref_get_key(source, key)
4835: (8)                   doc.xref_set_key(target, key, item[1])
```

File 6 - _apply_pages.py:

```
1: (0)              import multiprocessing
2: (0)              import os
3: (0)              import time
4: (0)              import pymupdf
5: (0)              class _worker_State:
6: (4)                  pass
7: (0)              _worker_state = _worker_State()
8: (0)              def _worker_init(
9: (8)                      path,
10: (8)                     initfn,
11: (8)                     initfn_args,
12: (8)                     initfn_kwargs,
13: (8)                     pagefn,
14: (8)                     pagefn_args,
15: (8)                     pagefn_kwargs,
16: (8)                     stats,
17: (8)                     ):
18: (4)                 _worker_state.path = path
19: (4)                 _worker_state.pagefn = pagefn
```

```
 20: (4)                 _worker_state.pagefn_args = pagefn_args
 21: (4)                 _worker_state.pagefn_kwargs = pagefn_kwargs
 22: (4)                 _worker_state.stats = stats
 23: (4)                 _worker_state.document = None
 24: (4)                 if initfn:
 25: (8)                     initfn(*initfn_args, **initfn_kwargs)
 26: (0)             def _stats_write(t, label):
 27: (4)                 t = time.time() - t
 28: (4)                 if t >= 0.1:
 29: (8)                     pymupdf.log(f'{os.getpid()=}: {t:2f}s: {label}.')
 30: (0)             def _worker_fn(page_number):
 31: (4)                 if not _worker_state.document:
 32: (8)                     if _worker_state.stats:
 33: (12)                        t = time.time()
 34: (8)                     _worker_state.document = pymupdf.Document(_worker_state.path)   #
pylint: disable=attribute-defined-outside-init
 35: (8)                     if _worker_state.stats:
 36: (12)                        _stats_write(t, 'pymupdf.Document()')
 37: (4)                 if _worker_state.stats:
 38: (8)                     t = time.time()
 39: (4)                 page = _worker_state.document[page_number]
 40: (4)                 if _worker_state.stats:
 41: (8)                     _stats_write(t, '_worker_state.document[page_number]')
 42: (4)                 if _worker_state.stats:
 43: (8)                     t = time.time()
 44: (4)                 ret = _worker_state.pagefn(
 45: (12)                        page,
 46: (12)                        *_worker_state.pagefn_args,
 47: (12)                        **_worker_state.pagefn_kwargs,
 48: (12)                        )
 49: (4)                 if _worker_state.stats:
 50: (8)                     _stats_write(t, '_worker_state.pagefn()')
 51: (4)                 return ret
 52: (0)             def _multiprocessing(
 53: (8)                     path,
 54: (8)                     pages,
 55: (8)                     pagefn,
 56: (8)                     pagefn_args,
 57: (8)                     pagefn_kwargs,
 58: (8)                     initfn,
 59: (8)                     initfn_args,
 60: (8)                     initfn_kwargs,
 61: (8)                     concurrency,
 62: (8)                     stats,
 63: (8)                     ):
 64: (4)                 with multiprocessing.Pool(
 65: (12)                        concurrency,
 66: (12)                        _worker_init,
 67: (12)                        (
 68: (16)                            path,
 69: (16)                            initfn, initfn_args, initfn_kwargs,
 70: (16)                            pagefn, pagefn_args, pagefn_kwargs,
 71: (16)                            stats,
 72: (12)                        ),
 73: (12)                        ) as pool:
 74: (8)                     result = pool.map_async(_worker_fn, pages)
 75: (8)                     return result.get()
 76: (0)             def _fork(
 77: (8)                     path,
 78: (8)                     pages,
 79: (8)                     pagefn,
 80: (8)                     pagefn_args,
 81: (8)                     pagefn_kwargs,
 82: (8)                     initfn,
 83: (8)                     initfn_args,
 84: (8)                     initfn_kwargs,
 85: (8)                     concurrency,
 86: (8)                     stats,
 87: (8)                     ):
```

```
 88: (4)                verbose = 0
 89: (4)                if concurrency is None:
 90: (8)                    concurrency = multiprocessing.cpu_count()
 91: (4)                queue_down = multiprocessing.Queue()
 92: (4)                queue_up = multiprocessing.Queue()
 93: (4)                def childfn():
 94: (8)                    document = None
 95: (8)                    if verbose:
 96: (12)                       pymupdf.log(f'{os.getpid()=}: {initfn=} {initfn_args=}')
 97: (8)                    _worker_init(
 98: (16)                           path,
 99: (16)                           initfn,
100: (16)                           initfn_args,
101: (16)                           initfn_kwargs,
102: (16)                           pagefn,
103: (16)                           pagefn_args,
104: (16)                           pagefn_kwargs,
105: (16)                           stats,
106: (16)                           )
107: (8)                    while 1:
108: (12)                       if verbose:
109: (16)                           pymupdf.log(f'{os.getpid()=}: calling get().')
110: (12)                       page_num = queue_down.get()
111: (12)                       if verbose:
112: (16)                           pymupdf.log(f'{os.getpid()=}: {page_num=}.')
113: (12)                       if page_num is None:
114: (16)                           break
115: (12)                       try:
116: (16)                           if not document:
117: (20)                               if stats:
118: (24)                                   t = time.time()
119: (20)                               document = pymupdf.Document(path)
120: (20)                               if stats:
121: (24)                                   _stats_write(t, 'pymupdf.Document(path)')
122: (16)                           if stats:
123: (20)                               t = time.time()
124: (16)                           page = document[page_num]
125: (16)                           if stats:
126: (20)                               _stats_write(t, 'document[page_num]')
127: (16)                           if verbose:
128: (20)                               pymupdf.log(f'{os.getpid()=}: {_worker_state=}')
129: (16)                           if stats:
130: (20)                               t = time.time()
131: (16)                           ret = pagefn(
132: (24)                                   page,
133: (24)                                   *_worker_state.pagefn_args,
134: (24)                                   **_worker_state.pagefn_kwargs,
135: (24)                                   )
136: (16)                           if stats:
137: (20)                               _stats_write(t, f'{page_num=} pagefn()')
138: (12)                       except Exception as e:
139: (16)                           if verbose: pymupdf.log(f'{os.getpid()=}: exception {e=}')
140: (16)                           ret = e
141: (12)                       if verbose:
142: (16)                           pymupdf.log(f'{os.getpid()=}: sending {page_num=} {ret=}')
143: (12)                       queue_up.put( (page_num, ret) )
144: (4)                error = None
145: (4)                pids = list()
146: (4)                try:
147: (8)                    if stats:
148: (12)                       t = time.time()
149: (8)                    for i in range(concurrency):
150: (12)                       p = os.fork()   # pylint: disable=no-member
151: (12)                       if p == 0:
152: (16)                           try:
153: (20)                               try:
154: (24)                                   childfn()
155: (20)                               except Exception as e:
156: (24)                                   pymupdf.log(f'{os.getpid()=}: childfn() => {e=}')
```

```
157: (24)                                   raise
158: (16)                               finally:
159: (20)                                   if verbose:
160: (24)                                       pymupdf.log(f'{os.getpid()=}: calling os._exit(0)')
161: (20)                                   os._exit(0)
162: (12)                           pids.append(p)
163: (8)                        if stats:
164: (12)                           _stats_write(t, 'create child processes')
165: (8)                        if stats:
166: (12)                           t = time.time()
167: (8)                        if verbose:
168: (12)                           pymupdf.log(f'Sending page numbers.')
169: (8)                        for page_num in range(len(pages)):
170: (12)                           queue_down.put(page_num)
171: (8)                        if stats:
172: (12)                           _stats_write(t, 'Send page numbers')
173: (8)                        ret = [None] * len(pages)
174: (8)                        for i in range(len(pages)):
175: (12)                           page_num, text = queue_up.get()
176: (12)                           if verbose:
177: (16)                               pymupdf.log(f'{page_num=} {len(text)=}')
178: (12)                           assert ret[page_num] is None
179: (12)                           if isinstance(text, Exception):
180: (16)                               if not error:
181: (20)                                   error = text
182: (16)                               break
183: (12)                           ret[page_num] = text
184: (8)                        for i in range(concurrency):
185: (12)                           queue_down.put(None)
186: (8)                        if verbose: pymupdf.log(f'Closing queues.')
187: (8)                        queue_down.close()
188: (8)                        if error:
189: (12)                           raise error
190: (8)                        if verbose:
191: (12)                           pymupdf.log(f'After concurrent, returning {len(ret)=}')
192: (8)                        return ret
193: (4)                    finally:
194: (8)                        if stats:
195: (12)                           t = time.time()
196: (8)                        for pid in pids:
197: (12)                           if verbose:
198: (16)                               pymupdf.log(f'waiting for {pid=}.')
199: (12)                           e = os.waitpid(pid, 0)
200: (12)                           if verbose:
201: (16)                               pymupdf.log(f'{pid=} => {e=}')
202: (8)                        if stats:
203: (12)                           _stats_write(t, 'Join all child proceses')
```

File 7 - _build.py:

```
1: (0)              mupdf_location='https://mupdf.com/downloads/archive/mupdf-1.24.9-
source.tar.gz'
```

File 8 - __init__.py:

```
1: (0)              '''
2: (0)              PyMuPDF implemented on top of MuPDF Python bindings.
3: (0)              License:
4: (4)                  SPDX-License-Identifier: GPL-3.0-only
5: (0)              '''
6: (0)              import atexit
7: (0)              import binascii
8: (0)              import collections
9: (0)              import inspect
10: (0)             import io
```

```
 11: (0)             import math
 12: (0)             import os
 13: (0)             import pathlib
 14: (0)             import glob
 15: (0)             import re
 16: (0)             import string
 17: (0)             import sys
 18: (0)             import tarfile
 19: (0)             import time
 20: (0)             import typing
 21: (0)             import warnings
 22: (0)             import weakref
 23: (0)             import zipfile
 24: (0)             from . import extra
 25: (0)             def _set_stream(name, default):
 26: (4)                 '''
 27: (4)                 Returns a stream to use based on environmental variable `name`.
 28: (4)                 '''
 29: (4)                 t = os.environ.get(name)
 30: (4)                 if t is None:
 31: (8)                     return default
 32: (4)                 elif t.startswith('fd:'):
 33: (8)                     return open(int(t[3:]), mode='w', closefd=False)
 34: (4)                 elif t.startswith('path:'):
 35: (8)                     return open(t[5:], 'w')
 36: (4)                 elif t.startswith('path+:'):
 37: (8)                     return open(t[6:], 'a')
 38: (4)                 else:
 39: (8)                     raise Exception(f'Unrecognised stream specification for {name!r}
 should match `fd:<int>`, `path:<string>` or `path+:<string>`: {t!r}')
 40: (0)             _g_out_log = _set_stream('PYMUPDF_LOG', sys.stdout)
 41: (0)             _g_out_message = _set_stream('PYMUPDF_MESSAGE', sys.stdout)
 42: (0)             _g_log_items = list()
 43: (0)             _g_log_items_active = False
 44: (0)             def _log_items():
 45: (4)                 return _g_log_items
 46: (0)             def _log_items_active(active):
 47: (4)                 global _g_log_items_active
 48: (4)                 _g_log_items_active = active
 49: (0)             def _log_items_clear():
 50: (4)                 del _g_log_items[:]
 51: (0)             def log( text='', caller=1):
 52: (4)                 '''
 53: (4)                 For development/debugging diagnostics.
 54: (4)                 '''
 55: (4)                 try:
 56: (8)                     stack = inspect.stack(context=0)
 57: (4)                 except StopIteration:
 58: (8)                     pass
 59: (4)                 else:
 60: (8)                     frame_record = stack[caller]
 61: (8)                     try:
 62: (12)                        filename = os.path.relpath(frame_record.filename)
 63: (8)                     except Exception:   # Can fail on windows.
 64: (12)                        filename = frame_record.filename
 65: (8)                     line = frame_record.lineno
 66: (8)                     function = frame_record.function
 67: (8)                     text = f'{filename}:{line}:{function}(): {text}'
 68: (4)                 if _g_log_items_active:
 69: (8)                     _g_log_items.append(text)
 70: (4)                 print(text, file=_g_out_log, flush=1)
 71: (0)             def message(text=''):
 72: (4)                 '''
 73: (4)                 For user messages.
 74: (4)                 '''
 75: (4)                 print(text, file=_g_out_message)
 76: (4)                 _g_out_message.flush()
 77: (0)             def exception_info():
 78: (4)                 import traceback
```

```
 79: (4)                 log(f'exception_info:')
 80: (4)                 log(traceback.format_exc())
 81: (0)             INVALID_NAME_CHARS = set(string.whitespace + "()<>[]{}/%" + chr(0))
 82: (0)             def get_env_bool( name, default):
 83: (4)                 '''
 84: (4)                 Returns `True`, `False` or `default` depending on whether $<name> is '1',
 85: (4)                 '0' or unset. Otherwise assert-fails.
 86: (4)                 '''
 87: (4)                 v = os.environ.get( name)
 88: (4)                 if v is None:
 89: (8)                     ret = default
 90: (4)                 elif v == '1':
 91: (8)                     ret = True
 92: (4)                 elif v == '0':
 93: (8)                     ret = False
 94: (4)                 else:
 95: (8)                     assert 0, f'Unrecognised value for {name}: {v!r}'
 96: (4)                 if ret != default:
 97: (8)                     log(f'Using non-default setting from {name}: {v!r}')
 98: (4)                 return ret
 99: (0)             def get_env_int( name, default):
100: (4)                 '''
101: (4)                 Returns `True`, `False` or `default` depending on whether $<name> is '1',
102: (4)                 '0' or unset. Otherwise assert-fails.
103: (4)                 '''
104: (4)                 v = os.environ.get( name)
105: (4)                 if v is None:
106: (8)                     ret = default
107: (4)                 else:
108: (8)                     ret = int(v)
109: (4)                 if ret != default:
110: (8)                     log(f'Using non-default setting from {name}: {v}')
111: (4)                 return ret
112: (0)             g_exceptions_verbose = get_env_int( 'PYMUPDF_EXCEPTIONS_VERBOSE', 1)
113: (0)             g_use_extra = get_env_bool( 'PYMUPDF_USE_EXTRA', True)
114: (0)             class _Globals:
115: (4)                 def __init__(self):
116: (8)                     self.no_device_caching = 0
117: (8)                     self.small_glyph_heights = 0
118: (8)                     self.subset_fontnames = 0
119: (8)                     self.skip_quad_corrections = 0
120: (0)             _globals = _Globals()
121: (0)             mupdf_cppyy = os.environ.get( 'MUPDF_CPPYY')
122: (0)             if mupdf_cppyy is not None:
123: (4)                 log( f'{__file__}: $MUPDF_CPPYY={mupdf_cppyy!r} so attempting to import
mupdf_cppyy.')
124: (4)                 log( f'{__file__}: $PYTHONPATH={os.environ["PYTHONPATH"]}')
125: (4)                 if mupdf_cppyy == '':
126: (8)                     import mupdf_cppyy
127: (4)                 else:
128: (8)                     import importlib
129: (8)                     mupdf_cppyy = importlib.machinery.SourceFileLoader(
130: (16)                            'mupdf_cppyy',
131: (16)                            mupdf_cppyy
132: (16)                            ).load_module()
133: (4)                 mupdf = mupdf_cppyy.cppyy.gbl.mupdf
134: (0)             else:
135: (4)                 try:
136: (8)                     from . import mupdf
137: (4)                 except Exception:
138: (8)                     import mupdf
139: (4)                 mupdf.reinit_singlethreaded()
140: (0)             def _int_rc(text):
141: (4)                 '''
142: (4)                 Converts string to int, ignoring trailing 'rc...'.
143: (4)                 '''
144: (4)                 rc = text.find('rc')
145: (4)                 if rc >= 0:
146: (8)                     text = text[:rc]
```

```
147: (4)                return int(text)
148: (0)            pymupdf_version = "1.24.10"
149: (0)            mupdf_version = mupdf.FZ_VERSION
150: (0)            pymupdf_date = "2024-09-02 00:00:01"
151: (0)            pymupdf_version_tuple = tuple( [_int_rc(i) for i in
pymupdf_version.split('.')])
152: (0)            mupdf_version_tuple = tuple( [_int_rc(i) for i in mupdf_version.split('.')])
153: (0)            assert mupdf_version_tuple == (mupdf.FZ_VERSION_MAJOR, mupdf.FZ_VERSION_MINOR,
mupdf.FZ_VERSION_PATCH), \
154: (8)                    f'Inconsistent MuPDF version numbers: {mupdf_version_tuple=} !=
{(mupdf.FZ_VERSION_MAJOR, mupdf.FZ_VERSION_MINOR, mupdf.FZ_VERSION_PATCH)=}'
155: (0)            pymupdf_date2 = pymupdf_date.replace('-', '').replace(' ', '').replace(':',
'')
156: (0)            version = (pymupdf_version, mupdf_version, pymupdf_date2)
157: (0)            VersionFitz = mupdf_version
158: (0)            VersionBind = pymupdf_version
159: (0)            VersionDate = pymupdf_date
160: (0)            def _format_g(value, *, fmt='%g'):
161: (4)                '''
162: (4)                Returns `value` formatted with mupdf.fz_format_double() if available,
163: (4)                otherwise with Python's `%`.
164: (4)                If `value` is a list or tuple, we return a space-separated string of
165: (4)                formatted values.
166: (4)                '''
167: (4)                if isinstance(value, (list, tuple)):
168: (8)                    ret = ''
169: (8)                    for v in value:
170: (12)                       if ret:
171: (16)                           ret += ' '
172: (12)                       ret += _format_g(v, fmt=fmt)
173: (8)                    return ret
174: (4)                else:
175: (8)                    if mupdf_version_tuple >= (1, 24, 2):
176: (12)                       return mupdf.fz_format_double(fmt, value)
177: (8)                    else:
178: (12)                       return fmt % value
179: (0)            format_g = _format_g
180: (0)            OptBytes = typing.Optional[typing.ByteString]
181: (0)            OptDict = typing.Optional[dict]
182: (0)            OptFloat = typing.Optional[float]
183: (0)            OptInt = typing.Union[int, None]
184: (0)            OptSeq = typing.Optional[typing.Sequence]
185: (0)            OptStr = typing.Optional[str]
186: (0)            Page = 'Page_forward_decl'
187: (0)            Point = 'Point_forward_decl'
188: (0)            TESSDATA_PREFIX = os.environ.get("TESSDATA_PREFIX")
189: (0)            matrix_like = 'matrix_like'
190: (0)            point_like = 'point_like'
191: (0)            quad_like = 'quad_like'
192: (0)            rect_like = 'rect_like'
193: (0)            def _as_fz_document(document):
194: (4)                '''
195: (4)                Returns document as a mupdf.FzDocument, upcasting as required. Raises
196: (4)                'document closed' exception if closed.
197: (4)                '''
198: (4)                if isinstance(document, Document):
199: (8)                    if document.is_closed:
200: (12)                       raise ValueError('document closed')
201: (8)                    document = document.this
202: (4)                if isinstance(document, mupdf.FzDocument):
203: (8)                    return document
204: (4)                elif isinstance(document, mupdf.PdfDocument):
205: (8)                    return document.super()
206: (4)                elif document is None:
207: (8)                    assert 0, f'document is None'
208: (4)                else:
209: (8)                    assert 0, f'Unrecognised {type(document)=}'
210: (0)            def _as_pdf_document(document, required=True):
211: (4)                '''
```

```
212: (4)                Returns `document` downcast to a mupdf.PdfDocument. If downcast fails
(i.e.
213: (4)                `document` is not actually a `PdfDocument`) then we assert-fail if
`required`
214: (4)                is true (the default) else return a `mupdf.PdfDocument` with `.m_internal`
215: (4)                false.
216: (4)                '''
217: (4)                if isinstance(document, Document):
218: (8)                    if document.is_closed:
219: (12)                       raise ValueError('document closed')
220: (8)                    document = document.this
221: (4)                if isinstance(document, mupdf.PdfDocument):
222: (8)                    return document
223: (4)                elif isinstance(document, mupdf.FzDocument):
224: (8)                    ret = mupdf.PdfDocument(document)
225: (8)                    if required:
226: (12)                       assert ret.m_internal
227: (8)                    return ret
228: (4)                elif document is None:
229: (8)                    assert 0, f'document is None'
230: (4)                else:
231: (8)                    assert 0, f'Unrecognised {type(document)=}'
232: (0)            def _as_fz_page(page):
233: (4)                '''
234: (4)                Returns page as a mupdf.FzPage, upcasting as required.
235: (4)                '''
236: (4)                if isinstance(page, Page):
237: (8)                    page = page.this
238: (4)                if isinstance(page, mupdf.PdfPage):
239: (8)                    return page.super()
240: (4)                elif isinstance(page, mupdf.FzPage):
241: (8)                    return page
242: (4)                elif page is None:
243: (8)                    assert 0, f'page is None'
244: (4)                else:
245: (8)                    assert 0, f'Unrecognised {type(page)=}'
246: (0)            def _as_pdf_page(page, required=True):
247: (4)                '''
248: (4)                Returns `page` downcast to a mupdf.PdfPage. If downcast fails (i.e. `page`
249: (4)                is not actually a `PdfPage`) then we assert-fail if `required` is true
(the
250: (4)                default) else return a `mupdf.PdfPage` with `.m_internal` false.
251: (4)                '''
252: (4)                if isinstance(page, Page):
253: (8)                    page = page.this
254: (4)                if isinstance(page, mupdf.PdfPage):
255: (8)                    return page
256: (4)                elif isinstance(page, mupdf.FzPage):
257: (8)                    ret = mupdf.pdf_page_from_fz_page(page)
258: (8)                    if required:
259: (12)                       assert ret.m_internal
260: (8)                    return ret
261: (4)                elif page is None:
262: (8)                    assert 0, f'page is None'
263: (4)                else:
264: (8)                    assert 0, f'Unrecognised {type(page)=}'
265: (0)            JM_MEMORY = 0
266: (0)            class Annot:
267: (4)                def __init__(self, annot):
268: (8)                    assert isinstance( annot, mupdf.PdfAnnot)
269: (8)                    self.this = annot
270: (4)                def __repr__(self):
271: (8)                    parent = getattr(self, 'parent', '<>')
272: (8)                    return "'%s' annotation on %s" % (self.type[1], str(parent))
273: (4)                def __str__(self):
274: (8)                    return self.__repr__()
275: (4)                def _erase(self):
276: (8)                    if getattr(self, "thisown", False):
277: (12)                       self.thisown = False
```

```
278: (4)                def _get_redact_values(self):
279: (8)                    annot = self.this
280: (8)                    if mupdf.pdf_annot_type(annot) != mupdf.PDF_ANNOT_REDACT:
281: (12)                       return
282: (8)                    values = dict()
283: (8)                    try:
284: (12)                       obj = mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot), "RO")
285: (12)                       if obj.m_internal:
286: (16)                           message_warning("Ignoring redaction key '/RO'.")
287: (16)                           xref = mupdf.pdf_to_num(obj)
288: (16)                           values[dictkey_xref] = xref
289: (12)                       obj = mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot),
"OverlayText")
290: (12)                       if obj.m_internal:
291: (16)                           text = mupdf.pdf_to_text_string(obj)
292: (16)                           values[dictkey_text] = JM_UnicodeFromStr(text)
293: (12)                       else:
294: (16)                           values[dictkey_text] = ''
295: (12)                       obj = mupdf.pdf_dict_get(mupdf.pdf_annot_obj(annot),
PDF_NAME('Q'))
296: (12)                       align = 0
297: (12)                       if obj.m_internal:
298: (16)                           align = mupdf.pdf_to_int(obj)
299: (12)                       values[dictkey_align] = align
300: (8)                    except Exception:
301: (12)                       if g_exceptions_verbose:    exception_info()
302: (12)                       return
303: (8)                    val = values
304: (8)                    if not val:
305: (12)                       return val
306: (8)                    val["rect"] = self.rect
307: (8)                    text_color, fontname, fontsize = TOOLS._parse_da(self)
308: (8)                    val["text_color"] = text_color
309: (8)                    val["fontname"] = fontname
310: (8)                    val["fontsize"] = fontsize
311: (8)                    fill = self.colors["fill"]
312: (8)                    val["fill"] = fill
313: (8)                    return val
314: (4)                def _getAP(self):
315: (8)                    if g_use_extra:
316: (12)                       assert isinstance( self.this, mupdf.PdfAnnot)
317: (12)                       ret = extra.Annot_getAP(self.this)
318: (12)                       assert isinstance( ret, bytes)
319: (12)                       return ret
320: (8)                    else:
321: (12)                       r = None
322: (12)                       res = None
323: (12)                       annot = self.this
324: (12)                       assert isinstance( annot, mupdf.PdfAnnot)
325: (12)                       annot_obj = mupdf.pdf_annot_obj( annot)
326: (12)                       ap = mupdf.pdf_dict_getl( annot_obj, PDF_NAME('AP'),
PDF_NAME('N'))
327: (12)                       if mupdf.pdf_is_stream( ap):
328: (16)                           res = mupdf.pdf_load_stream( ap)
329: (12)                       if res and res.m_internal:
330: (16)                           r = JM_BinFromBuffer(res)
331: (12)                       return r
332: (4)                def _setAP(self, buffer_, rect=0):
333: (8)                    try:
334: (12)                       annot = self.this
335: (12)                       annot_obj = mupdf.pdf_annot_obj( annot)
336: (12)                       page = mupdf.pdf_annot_page( annot)
337: (12)                       apobj = mupdf.pdf_dict_getl( annot_obj, PDF_NAME('AP'),
PDF_NAME('N'))
338: (12)                       if not apobj.m_internal:
339: (16)                           raise RuntimeError( MSG_BAD_APN)
340: (12)                       if not mupdf.pdf_is_stream( apobj):
341: (16)                           raise RuntimeError( MSG_BAD_APN)
342: (12)                       res = JM_BufferFromBytes( buffer_)
```

```
343: (12)                        if not res.m_internal:
344: (16)                            raise ValueError( MSG_BAD_BUFFER)
345: (12)                        JM_update_stream( page.doc(), apobj, res, 1)
346: (12)                        if rect:
347: (16)                            bbox = mupdf.pdf_dict_get_rect( annot_obj, PDF_NAME('Rect'))
348: (16)                            mupdf.pdf_dict_put_rect( apobj, PDF_NAME('BBox'), bbox)
349: (8)                    except Exception:
350: (12)                        if g_exceptions_verbose:    exception_info()
351: (4)                def _update_appearance(self, opacity=-1, blend_mode=None, fill_color=None,
rotate=-1):
352: (8)                    annot = self.this
353: (8)                    assert annot.m_internal
354: (8)                    annot_obj = mupdf.pdf_annot_obj( annot)
355: (8)                    page = mupdf.pdf_annot_page( annot)
356: (8)                    pdf = page.doc()
357: (8)                    type_ = mupdf.pdf_annot_type( annot)
358: (8)                    nfcol, fcol = JM_color_FromSequence(fill_color)
359: (8)                    try:
360: (12)                        if nfcol == 0 or type_ not in (
361: (20)                                mupdf.PDF_ANNOT_SQUARE,
362: (20)                                mupdf.PDF_ANNOT_CIRCLE,
363: (20)                                mupdf.PDF_ANNOT_LINE,
364: (20)                                mupdf.PDF_ANNOT_POLY_LINE,
365: (20)                                mupdf.PDF_ANNOT_POLYGON
366: (20)                                ):
367: (16)                            mupdf.pdf_dict_del( annot_obj, PDF_NAME('IC'))
368: (12)                        elif nfcol > 0:
369: (16)                            mupdf.pdf_set_annot_interior_color( annot, fcol[:nfcol])
370: (12)                        insert_rot = 1 if rotate >= 0 else 0
371: (12)                        if type_ not in (
372: (20)                                mupdf.PDF_ANNOT_CARET,
373: (20)                                mupdf.PDF_ANNOT_CIRCLE,
374: (20)                                mupdf.PDF_ANNOT_FREE_TEXT,
375: (20)                                mupdf.PDF_ANNOT_FILE_ATTACHMENT,
376: (20)                                mupdf.PDF_ANNOT_INK,
377: (20)                                mupdf.PDF_ANNOT_LINE,
378: (20)                                mupdf.PDF_ANNOT_POLY_LINE,
379: (20)                                mupdf.PDF_ANNOT_POLYGON,
380: (20)                                mupdf.PDF_ANNOT_SQUARE,
381: (20)                                mupdf.PDF_ANNOT_STAMP,
382: (20)                                mupdf.PDF_ANNOT_TEXT,
383: (20)                                ):
384: (16)                            insert_rot = 0
385: (12)                        if insert_rot:
386: (16)                            mupdf.pdf_dict_put_int( annot_obj, PDF_NAME('Rotate'), rotate)
387: (12)                        mupdf.pdf_dirty_annot( annot)
388: (12)                        mupdf.pdf_update_annot( annot) # let MuPDF update
389: (12)                        pdf.resynth_required = 0
390: (12)                        if type_ == mupdf.PDF_ANNOT_FREE_TEXT:
391: (16)                            if nfcol > 0:
392: (20)                                mupdf.pdf_set_annot_color( annot, fcol[:nfcol])
393: (12)                        elif nfcol > 0:
394: (16)                            col = mupdf.pdf_new_array( page.doc(), nfcol)
395: (16)                            for i in range( nfcol):
396: (20)                                mupdf.pdf_array_push_real( col, fcol[i])
397: (16)                            mupdf.pdf_dict_put( annot_obj, PDF_NAME('IC'), col)
398: (8)                    except Exception as e:
399: (12)                        if g_exceptions_verbose:    exception_info()
400: (12)                        message( f'cannot update annot: {e}')
401: (12)                        raise
402: (8)                    if (opacity < 0 or opacity >= 1) and not blend_mode:    # no opacity,
no blend_mode
403: (12)                        return True
404: (8)                    try:    # create or update /ExtGState
405: (12)                        ap = mupdf.pdf_dict_getl(
406: (20)                                mupdf.pdf_annot_obj(annot),
407: (20)                                PDF_NAME('AP'),
408: (20)                                PDF_NAME('N')
409: (20)                                )
```

```
410: (12)                        if not ap.m_internal:   # should never happen
411: (16)                            raise RuntimeError( MSG_BAD_APN)
412: (12)                        resources = mupdf.pdf_dict_get( ap, PDF_NAME('Resources'))
413: (12)                        if not resources.m_internal:    # no Resources yet: make one
414: (16)                            resources = mupdf.pdf_dict_put_dict( ap,
PDF_NAME('Resources'), 2)
415: (12)                        alp0 = mupdf.pdf_new_dict( page.doc(), 3)
416: (12)                        if opacity >= 0 and opacity < 1:
417: (16)                            mupdf.pdf_dict_put_real( alp0, PDF_NAME('CA'), opacity)
418: (16)                            mupdf.pdf_dict_put_real( alp0, PDF_NAME('ca'), opacity)
419: (16)                            mupdf.pdf_dict_put_real( annot_obj, PDF_NAME('CA'), opacity)
420: (12)                        if blend_mode:
421: (16)                            mupdf.pdf_dict_put_name( alp0, PDF_NAME('BM'), blend_mode)
422: (16)                            mupdf.pdf_dict_put_name( annot_obj, PDF_NAME('BM'),
blend_mode)
423: (12)                        extg = mupdf.pdf_dict_get( resources, PDF_NAME('ExtGState'))
424: (12)                        if not extg.m_internal: # no ExtGState yet: make one
425: (16)                            extg = mupdf.pdf_dict_put_dict( resources,
PDF_NAME('ExtGState'), 2)
426: (12)                        mupdf.pdf_dict_put( extg, PDF_NAME('H'), alp0)
427: (8)                    except Exception as e:
428: (12)                        if g_exceptions_verbose:    exception_info()
429: (12)                        message( f'cannot set opacity or blend mode\n: {e}')
430: (12)                        raise
431: (8)                    return True
432: (4)                @property
433: (4)                def apn_bbox(self):
434: (8)                    """annotation appearance bbox"""
435: (8)                    CheckParent(self)
436: (8)                    annot = self.this
437: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
438: (8)                    ap = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AP'), PDF_NAME('N'))
439: (8)                    if not ap.m_internal:
440: (12)                       val = JM_py_from_rect(mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE))
441: (8)                    else:
442: (12)                       rect = mupdf.pdf_dict_get_rect(ap, PDF_NAME('BBox'))
443: (12)                       val = JM_py_from_rect(rect)
444: (8)                    val = Rect(val) * self.get_parent().transformation_matrix
445: (8)                    val *= self.get_parent().derotation_matrix
446: (8)                    return val
447: (4)                @property
448: (4)                def apn_matrix(self):
449: (8)                    """annotation appearance matrix"""
450: (8)                    try:
451: (12)                       CheckParent(self)
452: (12)                       annot = self.this
453: (12)                       assert isinstance(annot, mupdf.PdfAnnot)
454: (12)                       ap = mupdf.pdf_dict_getl(
455: (20)                               mupdf.pdf_annot_obj(annot),
456: (20)                               mupdf.PDF_ENUM_NAME_AP,
457: (20)                               mupdf.PDF_ENUM_NAME_N
458: (20)                               )
459: (12)                       if not ap.m_internal:
460: (16)                           return JM_py_from_matrix(mupdf.FzMatrix())
461: (12)                       mat = mupdf.pdf_dict_get_matrix(ap, mupdf.PDF_ENUM_NAME_Matrix)
462: (12)                       val = JM_py_from_matrix(mat)
463: (12)                       val = Matrix(val)
464: (12)                       return val
465: (8)                    except Exception:
466: (12)                       if g_exceptions_verbose:    exception_info()
467: (12)                       raise
468: (4)                @property
469: (4)                def blendmode(self):
470: (8)                    """annotation BlendMode"""
471: (8)                    CheckParent(self)
472: (8)                    annot = self.this
473: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
474: (8)                    obj = mupdf.pdf_dict_get(annot_obj, PDF_NAME('BM'))
475: (8)                    blend_mode = None
```

```
476: (8)                    if obj.m_internal:
477: (12)                       blend_mode = JM_UnicodeFromStr(mupdf.pdf_to_name(obj))
478: (12)                       return blend_mode
479: (8)                    obj = mupdf.pdf_dict_getl(
480: (16)                           annot_obj,
481: (16)                           PDF_NAME('AP'),
482: (16)                           PDF_NAME('N'),
483: (16)                           PDF_NAME('Resources'),
484: (16)                           PDF_NAME('ExtGState'),
485: (16)                           )
486: (8)                    if mupdf.pdf_is_dict(obj):
487: (12)                       n = mupdf.pdf_dict_len(obj)
488: (12)                       for i in range(n):
489: (16)                           obj1 = mupdf.pdf_dict_get_val(obj, i)
490: (16)                           if mupdf.pdf_is_dict(obj1):
491: (20)                               m = mupdf.pdf_dict_len(obj1)
492: (20)                               for j in range(m):
493: (24)                                   obj2 = mupdf.pdf_dict_get_key(obj1, j)
494: (24)                                   if mupdf.pdf_objcmp(obj2, PDF_NAME('BM')) == 0:
495: (28)                                       blend_mode =
JM_UnicodeFromStr(mupdf.pdf_to_name(mupdf.pdf_dict_get_val(obj1, j)))
496: (28)                                       return blend_mode
497: (8)                    return blend_mode
498: (4)                @property
499: (4)                def border(self):
500: (8)                    """Border information."""
501: (8)                    CheckParent(self)
502: (8)                    atype = self.type[0]
503: (8)                    if atype not in (
504: (16)                           mupdf.PDF_ANNOT_CIRCLE,
505: (16)                           mupdf.PDF_ANNOT_FREE_TEXT,
506: (16)                           mupdf.PDF_ANNOT_INK,
507: (16)                           mupdf.PDF_ANNOT_LINE,
508: (16)                           mupdf.PDF_ANNOT_POLY_LINE,
509: (16)                           mupdf.PDF_ANNOT_POLYGON,
510: (16)                           mupdf.PDF_ANNOT_SQUARE,
511: (16)                           ):
512: (12)                       return dict()
513: (8)                    ao = mupdf.pdf_annot_obj(self.this)
514: (8)                    ret = JM_annot_border(ao)
515: (8)                    return ret
516: (4)                def clean_contents(self, sanitize=1):
517: (8)                    """Clean appearance contents stream."""
518: (8)                    CheckParent(self)
519: (8)                    annot = self.this
520: (8)                    pdf = mupdf.pdf_get_bound_document(mupdf.pdf_annot_obj(annot))
521: (8)                    filter_ = _make_PdfFilterOptions(recurse=1, instance_forms=0, ascii=0,
sanitize=sanitize)
522: (8)                    mupdf.pdf_filter_annot_contents(pdf, annot, filter_)
523: (4)                @property
524: (4)                def colors(self):
525: (8)                    """Color definitions."""
526: (8)                    try:
527: (12)                       CheckParent(self)
528: (12)                       annot = self.this
529: (12)                       assert isinstance(annot, mupdf.PdfAnnot)
530: (12)                       return JM_annot_colors(mupdf.pdf_annot_obj(annot))
531: (8)                    except Exception:
532: (12)                       if g_exceptions_verbose:    exception_info()
533: (12)                       raise
534: (4)                def delete_responses(self):
535: (8)                    """Delete 'Popup' and responding annotations."""
536: (8)                    CheckParent(self)
537: (8)                    annot = self.this
538: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
539: (8)                    page = mupdf.pdf_annot_page(annot)
540: (8)                    while 1:
541: (12)                       irt_annot = JM_find_annot_irt(annot)
542: (12)                       if not irt_annot.m_internal:
```

```
(16)                            break
(12)                        mupdf.pdf_delete_annot(page, irt_annot)
(8)                     mupdf.pdf_dict_del(annot_obj, PDF_NAME('Popup'))
(8)                     annots = mupdf.pdf_dict_get(page.obj(), PDF_NAME('Annots'))
(8)                     n = mupdf.pdf_array_len(annots)
(8)                     found = 0
(8)                     for i in range(n-1, -1, -1):
(12)                        o = mupdf.pdf_array_get(annots, i)
(12)                        p = mupdf.pdf_dict_get(o, PDF_NAME('Parent'))
(12)                        if not o.m_internal:
(16)                            continue
(12)                        if not mupdf.pdf_objcmp(p, annot_obj):
(16)                            mupdf.pdf_array_delete(annots, i)
(16)                            found = 1
(8)                     if found:
(12)                        mupdf.pdf_dict_put(page.obj(), PDF_NAME('Annots'), annots)
(4)                 @property
(4)                 def file_info(self):
(8)                     """Attached file information."""
(8)                     CheckParent(self)
(8)                     res = dict()
(8)                     length = -1
(8)                     size = -1
(8)                     desc = None
(8)                     annot = self.this
(8)                     annot_obj = mupdf.pdf_annot_obj(annot)
(8)                     type_ = mupdf.pdf_annot_type(annot)
(8)                     if type_ != mupdf.PDF_ANNOT_FILE_ATTACHMENT:
(12)                        raise TypeError( MSG_BAD_ANNOT_TYPE)
(8)                     stream = mupdf.pdf_dict_getl(
(16)                            annot_obj,
(16)                            PDF_NAME('FS'),
(16)                            PDF_NAME('EF'),
(16)                            PDF_NAME('F'),
(16)                            )
(8)                     if not stream.m_internal:
(12)                        RAISEPY( "bad PDF: file entry not found", JM_Exc_FileDataError)
(8)                     fs = mupdf.pdf_dict_get(annot_obj, PDF_NAME('FS'))
(8)                     o = mupdf.pdf_dict_get(fs, PDF_NAME('UF'))
(8)                     if o.m_internal:
(12)                        filename = mupdf.pdf_to_text_string(o)
(8)                     else:
(12)                        o = mupdf.pdf_dict_get(fs, PDF_NAME('F'))
(12)                        if o.m_internal:
(16)                            filename = mupdf.pdf_to_text_string(o)
(8)                     o = mupdf.pdf_dict_get(fs, PDF_NAME('Desc'))
(8)                     if o.m_internal:
(12)                        desc = mupdf.pdf_to_text_string(o)
(8)                     o = mupdf.pdf_dict_get(stream, PDF_NAME('Length'))
(8)                     if o.m_internal:
(12)                        length = mupdf.pdf_to_int(o)
(8)                     o = mupdf.pdf_dict_getl(stream, PDF_NAME('Params'), PDF_NAME('Size'))
(8)                     if o.m_internal:
(12)                        size = mupdf.pdf_to_int(o)
(8)                     res[ dictkey_filename] = JM_EscapeStrFromStr(filename)
(8)                     res[ dictkey_desc] = JM_UnicodeFromStr(desc)
(8)                     res[ dictkey_length] = length
(8)                     res[ dictkey_size] = size
(8)                     return res
(4)                 @property
(4)                 def flags(self):
(8)                     """Flags field."""
(8)                     CheckParent(self)
(8)                     annot = self.this
(8)                     return mupdf.pdf_annot_flags(annot)
(4)                 def get_file(self):
(8)                     """Retrieve attached file content."""
(8)                     CheckParent(self)
(8)                     annot = self.this
```

```
612: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
613: (8)                    type = mupdf.pdf_annot_type(annot)
614: (8)                    if type != mupdf.PDF_ANNOT_FILE_ATTACHMENT:
615: (12)                       raise TypeError( MSG_BAD_ANNOT_TYPE)
616: (8)                    stream = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('FS'),
PDF_NAME('EF'), PDF_NAME('F'))
617: (8)                    if not stream.m_internal:
618: (12)                       RAISEPY( "bad PDF: file entry not found", JM_Exc_FileDataError)
619: (8)                    buf = mupdf.pdf_load_stream(stream)
620: (8)                    res = JM_BinFromBuffer(buf)
621: (8)                    return res
622: (4)                def get_oc(self):
623: (8)                    """Get annotation optional content reference."""
624: (8)                    CheckParent(self)
625: (8)                    oc = 0
626: (8)                    annot = self.this
627: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
628: (8)                    obj = mupdf.pdf_dict_get(annot_obj, PDF_NAME('OC'))
629: (8)                    if obj.m_internal:
630: (12)                       oc = mupdf.pdf_to_num(obj)
631: (8)                    return oc
632: (4)                def get_parent(self):
633: (8)                    try:
634: (12)                       ret = getattr( self, 'parent')
635: (8)                    except AttributeError:
636: (12)                       page = mupdf.pdf_annot_page(self.this)
637: (12)                       assert isinstance( page, mupdf.PdfPage)
638: (12)                       document = Document( page.doc()) if page.m_internal else None
639: (12)                       ret = Page(page, document)
640: (12)                       self.parent = ret
641: (8)                    return ret
642: (4)                def get_pixmap(self, matrix=None, dpi=None, colorspace=None, alpha=0):
643: (8)                    """annotation Pixmap"""
644: (8)                    CheckParent(self)
645: (8)                    cspaces = {"gray": csGRAY, "rgb": csRGB, "cmyk": csCMYK}
646: (8)                    if type(colorspace) is str:
647: (12)                       colorspace = cspaces.get(colorspace.lower(), None)
648: (8)                    if dpi:
649: (12)                       matrix = Matrix(dpi / 72, dpi / 72)
650: (8)                    ctm = JM_matrix_from_py(matrix)
651: (8)                    cs = colorspace
652: (8)                    if not cs:
653: (12)                       cs = mupdf.fz_device_rgb()
654: (8)                    pix = mupdf.pdf_new_pixmap_from_annot(self.this, ctm, cs,
mupdf.FzSeparations(0), alpha)
655: (8)                    ret = Pixmap(pix)
656: (8)                    if dpi:
657: (12)                       ret.set_dpi(dpi, dpi)
658: (8)                    return ret
659: (4)                def get_sound(self):
660: (8)                    """Retrieve sound stream."""
661: (8)                    CheckParent(self)
662: (8)                    annot = self.this
663: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
664: (8)                    type = mupdf.pdf_annot_type(annot)
665: (8)                    sound = mupdf.pdf_dict_get(annot_obj, PDF_NAME('Sound'))
666: (8)                    if type != mupdf.PDF_ANNOT_SOUND or not sound.m_internal:
667: (12)                       raise TypeError( MSG_BAD_ANNOT_TYPE)
668: (8)                    if mupdf.pdf_dict_get(sound, PDF_NAME('F')).m_internal:
669: (12)                       RAISEPY( "unsupported sound stream", JM_Exc_FileDataError)
670: (8)                    res = dict()
671: (8)                    obj = mupdf.pdf_dict_get(sound, PDF_NAME('R'))
672: (8)                    if obj.m_internal:
673: (12)                       res['rate'] = mupdf.pdf_to_real(obj)
674: (8)                    obj = mupdf.pdf_dict_get(sound, PDF_NAME('C'))
675: (8)                    if obj.m_internal:
676: (12)                       res['channels'] = mupdf.pdf_to_int(obj)
677: (8)                    obj = mupdf.pdf_dict_get(sound, PDF_NAME('B'))
678: (8)                    if obj.m_internal:
```

```
679: (12)                        res['bps'] = mupdf.pdf_to_int(obj)
680: (8)                     obj = mupdf.pdf_dict_get(sound, PDF_NAME('E'))
681: (8)                     if obj.m_internal:
682: (12)                        res['encoding'] = mupdf.pdf_to_name(obj)
683: (8)                     obj = mupdf.pdf_dict_gets(sound, "CO")
684: (8)                     if obj.m_internal:
685: (12)                        res['compression'] = mupdf.pdf_to_name(obj)
686: (8)                     buf = mupdf.pdf_load_stream(sound)
687: (8)                     stream = JM_BinFromBuffer(buf)
688: (8)                     res['stream'] = stream
689: (8)                     return res
690: (4)                 def get_textpage(self, clip=None, flags=0):
691: (8)                     """Make annotation TextPage."""
692: (8)                     CheckParent(self)
693: (8)                     options = mupdf.FzStextOptions()
694: (8)                     options.flags = flags
695: (8)                     annot = self.this
696: (8)                     stextpage = mupdf.FzStextPage(annot, options)
697: (8)                     ret = TextPage(stextpage)
698: (8)                     p = self.get_parent()
699: (8)                     if isinstance(p, weakref.ProxyType):
700: (12)                        ret.parent = p
701: (8)                     else:
702: (12)                        ret.parent = weakref.proxy(p)
703: (8)                     return ret
704: (4)                 @property
705: (4)                 def has_popup(self):
706: (8)                     """Check if annotation has a Popup."""
707: (8)                     CheckParent(self)
708: (8)                     annot = self.this
709: (8)                     obj = mupdf.pdf_dict_get(mupdf.pdf_annot_obj(annot),
PDF_NAME('Popup'))
710: (8)                     return True if obj.m_internal else False
711: (4)                 @property
712: (4)                 def info(self):
713: (8)                     """Various information details."""
714: (8)                     CheckParent(self)
715: (8)                     annot = self.this
716: (8)                     res = dict()
717: (8)                     res[dictkey_content] =
JM_UnicodeFromStr(mupdf.pdf_annot_contents(annot))
718: (8)                     o = mupdf.pdf_dict_get(mupdf.pdf_annot_obj(annot), PDF_NAME('Name'))
719: (8)                     res[dictkey_name] = JM_UnicodeFromStr(mupdf.pdf_to_name(o))
720: (8)                     o = mupdf.pdf_dict_get(mupdf.pdf_annot_obj(annot), PDF_NAME('T'))
721: (8)                     res[dictkey_title] = JM_UnicodeFromStr(mupdf.pdf_to_text_string(o))
722: (8)                     o = mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot), "CreationDate")
723: (8)                     res[dictkey_creationDate] =
JM_UnicodeFromStr(mupdf.pdf_to_text_string(o))
724: (8)                     o = mupdf.pdf_dict_get(mupdf.pdf_annot_obj(annot), PDF_NAME('M'))
725: (8)                     res[dictkey_modDate] = JM_UnicodeFromStr(mupdf.pdf_to_text_string(o))
726: (8)                     o = mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot), "Subj")
727: (8)                     res[dictkey_subject] = mupdf.pdf_to_text_string(o)
728: (8)                     o = mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot), "NM")
729: (8)                     res[dictkey_id] = JM_UnicodeFromStr(mupdf.pdf_to_text_string(o))
730: (8)                     return res
731: (4)                 @property
732: (4)                 def irt_xref(self):
733: (8)                     '''
734: (8)                     annotation IRT xref
735: (8)                     '''
736: (8)                     annot = self.this
737: (8)                     annot_obj = mupdf.pdf_annot_obj( annot)
738: (8)                     irt = mupdf.pdf_dict_get( annot_obj, PDF_NAME('IRT'))
739: (8)                     if not irt.m_internal:
740: (12)                        return 0
741: (8)                     return mupdf.pdf_to_num( irt)
742: (4)                 @property
743: (4)                 def is_open(self):
744: (8)                     """Get 'open' status of annotation or its Popup."""
```

```
745: (8)                    CheckParent(self)
746: (8)                    return mupdf.pdf_annot_is_open(self.this)
747: (4)                @property
748: (4)                def language(self):
749: (8)                    """annotation language"""
750: (8)                    this_annot = self.this
751: (8)                    lang = mupdf.pdf_annot_language(this_annot)
752: (8)                    if lang == mupdf.FZ_LANG_UNSET:
753: (12)                       return
754: (8)                    assert hasattr(mupdf, 'fz_string_from_text_language2')
755: (8)                    return mupdf.fz_string_from_text_language2(lang)
756: (4)                @property
757: (4)                def line_ends(self):
758: (8)                    """Line end codes."""
759: (8)                    CheckParent(self)
760: (8)                    annot = self.this
761: (8)                    if not mupdf.pdf_annot_has_line_ending_styles(annot):
762: (12)                       return
763: (8)                    lstart = mupdf.pdf_annot_line_start_style(annot)
764: (8)                    lend = mupdf.pdf_annot_line_end_style(annot)
765: (8)                    return lstart, lend
766: (4)                @property
767: (4)                def next(self):
768: (8)                    """Next annotation."""
769: (8)                    CheckParent(self)
770: (8)                    this_annot = self.this
771: (8)                    assert isinstance(this_annot, mupdf.PdfAnnot)
772: (8)                    assert this_annot.m_internal
773: (8)                    type_ = mupdf.pdf_annot_type(this_annot)
774: (8)                    if type_ != mupdf.PDF_ANNOT_WIDGET:
775: (12)                       annot = mupdf.pdf_next_annot(this_annot)
776: (8)                    else:
777: (12)                       annot = mupdf.pdf_next_widget(this_annot)
778: (8)                    val = Annot(annot) if annot.m_internal else None
779: (8)                    if not val:
780: (12)                       return None
781: (8)                    val.thisown = True
782: (8)                    assert val.get_parent().this.m_internal_value() ==
self.get_parent().this.m_internal_value()
783: (8)                    val.parent._annot_refs[id(val)] = val
784: (8)                    if val.type[0] == mupdf.PDF_ANNOT_WIDGET:
785: (12)                       widget = Widget()
786: (12)                       TOOLS._fill_widget(val, widget)
787: (12)                       val = widget
788: (8)                    return val
789: (4)                @property
790: (4)                def opacity(self):
791: (8)                    """Opacity."""
792: (8)                    CheckParent(self)
793: (8)                    annot = self.this
794: (8)                    opy = -1
795: (8)                    ca = mupdf.pdf_dict_get( mupdf.pdf_annot_obj(annot),
mupdf.PDF_ENUM_NAME_CA)
796: (8)                    if mupdf.pdf_is_number(ca):
797: (12)                       opy = mupdf.pdf_to_real(ca)
798: (8)                    return opy
799: (4)                @property
800: (4)                def popup_rect(self):
801: (8)                    """annotation 'Popup' rectangle"""
802: (8)                    CheckParent(self)
803: (8)                    rect = mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE)
804: (8)                    annot = self.this
805: (8)                    annot_obj = mupdf.pdf_annot_obj( annot)
806: (8)                    obj = mupdf.pdf_dict_get( annot_obj, PDF_NAME('Popup'))
807: (8)                    if obj.m_internal:
808: (12)                       rect = mupdf.pdf_dict_get_rect(obj, PDF_NAME('Rect'))
809: (8)                    val = JM_py_from_rect(rect)
810: (8)                    val = Rect(val) * self.get_parent().transformation_matrix
811: (8)                    val *= self.get_parent().derotation_matrix
```

```
812: (8)                    return val
813: (4)                @property
814: (4)                def popup_xref(self):
815: (8)                    """annotation 'Popup' xref"""
816: (8)                    CheckParent(self)
817: (8)                    xref = 0
818: (8)                    annot = self.this
819: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
820: (8)                    obj = mupdf.pdf_dict_get(annot_obj, PDF_NAME('Popup'))
821: (8)                    if obj.m_internal:
822: (12)                       xref = mupdf.pdf_to_num(obj)
823: (8)                    return xref
824: (4)                @property
825: (4)                def rect(self):
826: (8)                    """annotation rectangle"""
827: (8)                    if g_use_extra:
828: (12)                       val = extra.Annot_rect3( self.this)
829: (8)                    else:
830: (12)                       val = mupdf.pdf_bound_annot(self.this)
831: (8)                    val = Rect(val)
832: (8)                    p = self.get_parent()
833: (8)                    val *= p.derotation_matrix
834: (8)                    return val
835: (4)                @property
836: (4)                def rect_delta(self):
837: (8)                    '''
838: (8)                    annotation delta values to rectangle
839: (8)                    '''
840: (8)                    annot_obj = mupdf.pdf_annot_obj(self.this)
841: (8)                    arr = mupdf.pdf_dict_get( annot_obj, PDF_NAME('RD'))
842: (8)                    if mupdf.pdf_array_len( arr) == 4:
843: (12)                       return (
844: (20)                               mupdf.pdf_to_real( mupdf.pdf_array_get( arr, 0)),
845: (20)                               mupdf.pdf_to_real( mupdf.pdf_array_get( arr, 1)),
846: (20)                               -mupdf.pdf_to_real( mupdf.pdf_array_get( arr, 2)),
847: (20)                               -mupdf.pdf_to_real( mupdf.pdf_array_get( arr, 3)),
848: (20)                               )
849: (4)                @property
850: (4)                def rotation(self):
851: (8)                    """annotation rotation"""
852: (8)                    CheckParent(self)
853: (8)                    annot = self.this
854: (8)                    rotation = mupdf.pdf_dict_get( mupdf.pdf_annot_obj(annot),
mupdf.PDF_ENUM_NAME_Rotate)
855: (8)                    if not rotation.m_internal:
856: (12)                       return -1
857: (8)                    return mupdf.pdf_to_int( rotation)
858: (4)                def set_apn_bbox(self, bbox):
859: (8)                    """
860: (8)                    Set annotation appearance bbox.
861: (8)                    """
862: (8)                    CheckParent(self)
863: (8)                    page = self.get_parent()
864: (8)                    rot = page.rotation_matrix
865: (8)                    mat = page.transformation_matrix
866: (8)                    bbox *= rot * ~mat
867: (8)                    annot = self.this
868: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
869: (8)                    ap = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AP'), PDF_NAME('N'))
870: (8)                    if not ap.m_internal:
871: (12)                       raise RuntimeError( MSG_BAD_APN)
872: (8)                    rect = JM_rect_from_py(bbox)
873: (8)                    mupdf.pdf_dict_put_rect(ap, PDF_NAME('BBox'), rect)
874: (4)                def set_apn_matrix(self, matrix):
875: (8)                    """Set annotation appearance matrix."""
876: (8)                    CheckParent(self)
877: (8)                    annot = self.this
878: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
879: (8)                    ap = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AP'), PDF_NAME('N'))
```

```
880: (8)                    if not ap.m_internal:
881: (12)                       raise RuntimeError( MSG_BAD_APN)
882: (8)                    mat = JM_matrix_from_py(matrix)
883: (8)                    mupdf.pdf_dict_put_matrix(ap, PDF_NAME('Matrix'), mat)
884: (4)                def set_blendmode(self, blend_mode):
885: (8)                    """Set annotation BlendMode."""
886: (8)                    CheckParent(self)
887: (8)                    annot = self.this
888: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
889: (8)                    mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('BM'), blend_mode)
890: (4)                def set_border(self, border=None, width=-1, style=None, dashes=None,
clouds=-1):
891: (8)                    """Set border properties.
892: (8)                    Either a dict, or direct arguments width, style, dashes or clouds."""
893: (8)                    CheckParent(self)
894: (8)                    atype, atname = self.type[:2]  # annotation type
895: (8)                    if atype not in (
896: (16)                           mupdf.PDF_ANNOT_CIRCLE,
897: (16)                           mupdf.PDF_ANNOT_FREE_TEXT,
898: (16)                           mupdf.PDF_ANNOT_INK,
899: (16)                           mupdf.PDF_ANNOT_LINE,
900: (16)                           mupdf.PDF_ANNOT_POLY_LINE,
901: (16)                           mupdf.PDF_ANNOT_POLYGON,
902: (16)                           mupdf.PDF_ANNOT_SQUARE,
903: (16)                           ):
904: (12)                       message(f"Cannot set border for '{atname}'.")
905: (12)                       return None
906: (8)                    if atype not in (
907: (16)                           mupdf.PDF_ANNOT_CIRCLE,
908: (16)                           mupdf.PDF_ANNOT_FREE_TEXT,
909: (16)                           mupdf.PDF_ANNOT_POLYGON,
910: (16)                           mupdf.PDF_ANNOT_SQUARE,
911: (16)                           ):
912: (12)                       if clouds > 0:
913: (16)                           message(f"Cannot set cloudy border for '{atname}'.")
914: (16)                           clouds = -1  # do not set border effect
915: (8)                    if type(border) is not dict:
916: (12)                       border = {"width": width, "style": style, "dashes": dashes,
"clouds": clouds}
917: (8)                    border.setdefault("width", -1)
918: (8)                    border.setdefault("style", None)
919: (8)                    border.setdefault("dashes", None)
920: (8)                    border.setdefault("clouds", -1)
921: (8)                    if border["width"] is None:
922: (12)                       border["width"] = -1
923: (8)                    if border["clouds"] is None:
924: (12)                       border["clouds"] = -1
925: (8)                    if hasattr(border["dashes"], "__getitem__"):  # ensure sequence items
are integers
926: (12)                       border["dashes"] = tuple(border["dashes"])
927: (12)                       for item in border["dashes"]:
928: (16)                           if not isinstance(item, int):
929: (20)                               border["dashes"] = None
930: (20)                               break
931: (8)                    annot = self.this
932: (8)                    annot_obj = mupdf.pdf_annot_obj( annot)
933: (8)                    pdf = mupdf.pdf_get_bound_document( annot_obj)
934: (8)                    return JM_annot_set_border( border, pdf, annot_obj)
935: (4)                def set_colors(self, colors=None, stroke=None, fill=None):
936: (8)                    """Set 'stroke' and 'fill' colors.
937: (8)                    Use either a dict or the direct arguments.
938: (8)                    """
939: (8)                    CheckParent(self)
940: (8)                    doc = self.get_parent().parent
941: (8)                    if type(colors) is not dict:
942: (12)                       colors = {"fill": fill, "stroke": stroke}
943: (8)                    fill = colors.get("fill")
944: (8)                    stroke = colors.get("stroke")
945: (8)                    fill_annots = (mupdf.PDF_ANNOT_CIRCLE, mupdf.PDF_ANNOT_SQUARE,
```

```
mupdf.PDF_ANNOT_LINE, mupdf.PDF_ANNOT_POLY_LINE, mupdf.PDF_ANNOT_POLYGON,
946: (23)                                   mupdf.PDF_ANNOT_REDACT,)
947: (8)                    if stroke in ([], ()):
948: (12)                       doc.xref_set_key(self.xref, "C", "[]")
949: (8)                    elif stroke is not None:
950: (12)                       if hasattr(stroke, "__float__"):
951: (16)                           stroke = [float(stroke)]
952: (12)                       CheckColor(stroke)
953: (12)                       assert len(stroke) in (1, 3, 4)
954: (12)                       s = f"[{_format_g(stroke)}]"
955: (12)                       doc.xref_set_key(self.xref, "C", s)
956: (8)                    if fill and self.type[0] not in fill_annots:
957: (12)                       message("Warning: fill color ignored for annot type '%s'." %
self.type[1])
958: (12)                       return
959: (8)                    if fill in ([], ()):
960: (12)                       doc.xref_set_key(self.xref, "IC", "[]")
961: (8)                    elif fill is not None:
962: (12)                       if hasattr(fill, "__float__"):
963: (16)                           fill = [float(fill)]
964: (12)                       CheckColor(fill)
965: (12)                       assert len(fill) in (1, 3, 4)
966: (12)                       s = f"[{_format_g(fill)}]"
967: (12)                       doc.xref_set_key(self.xref, "IC", s)
968: (4)                def set_flags(self, flags):
969: (8)                    """Set annotation flags."""
970: (8)                    CheckParent(self)
971: (8)                    annot = self.this
972: (8)                    mupdf.pdf_set_annot_flags(annot, flags)
973: (4)                def set_info(self, info=None, content=None, title=None, creationDate=None,
modDate=None, subject=None):
974: (8)                    """Set various properties."""
975: (8)                    CheckParent(self)
976: (8)                    if type(info) is dict:  # build the args from the dictionary
977: (12)                       content = info.get("content", None)
978: (12)                       title = info.get("title", None)
979: (12)                       creationDate = info.get("creationDate", None)
980: (12)                       modDate = info.get("modDate", None)
981: (12)                       subject = info.get("subject", None)
982: (12)                       info = None
983: (8)                    annot = self.this
984: (8)                    is_markup = mupdf.pdf_annot_has_author(annot)
985: (8)                    if content:
986: (12)                       mupdf.pdf_set_annot_contents(annot, content)
987: (8)                    if is_markup:
988: (12)                       if title:
989: (16)                           mupdf.pdf_set_annot_author(annot, title)
990: (12)                       if creationDate:
991: (16)                           mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(annot),
PDF_NAME('CreationDate'), creationDate)
992: (12)                       if modDate:
993: (16)                           mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(annot),
PDF_NAME('M'), modDate)
994: (12)                       if subject:
995: (16)                           mupdf.pdf_dict_puts(mupdf.pdf_annot_obj(annot), "Subj",
mupdf.pdf_new_text_string(subject))
996: (4)                def set_irt_xref(self, xref):
997: (8)                    '''
998: (8)                    Set annotation IRT xref
999: (8)                    '''
1000: (8)                   annot = self.this
1001: (8)                   annot_obj = mupdf.pdf_annot_obj( annot)
1002: (8)                   page = mupdf.pdf_annot_page( annot)
1003: (8)                   if xref < 1 or xref >= mupdf.pdf_xref_len( page.doc()):
1004: (12)                      raise ValueError( MSG_BAD_XREF)
1005: (8)                   irt = mupdf.pdf_new_indirect( page.doc(), xref, 0)
1006: (8)                   subt = mupdf.pdf_dict_get( irt, PDF_NAME('Subtype'))
1007: (8)                   irt_subt = mupdf.pdf_annot_type_from_string( mupdf.pdf_to_name( subt))
1008: (8)                   if irt_subt < 0:
```

```
1009: (12)                      raise ValueError( MSG_IS_NO_ANNOT)
1010: (8)                   mupdf.pdf_dict_put( annot_obj, PDF_NAME('IRT'), irt)
1011: (4)               def set_language(self, language=None):
1012: (8)                   """Set annotation language."""
1013: (8)                   CheckParent(self)
1014: (8)                   this_annot = self.this
1015: (8)                   if not language:
1016: (12)                      lang = mupdf.FZ_LANG_UNSET
1017: (8)                   else:
1018: (12)                      lang = mupdf.fz_text_language_from_string(language)
1019: (8)                   mupdf.pdf_set_annot_language(this_annot, lang)
1020: (4)               def set_line_ends(self, start, end):
1021: (8)                   """Set line end codes."""
1022: (8)                   CheckParent(self)
1023: (8)                   annot = self.this
1024: (8)                   if mupdf.pdf_annot_has_line_ending_styles(annot):
1025: (12)                      mupdf.pdf_set_annot_line_ending_styles(annot, start, end)
1026: (8)                   else:
1027: (12)                      message_warning("bad annot type for line ends")
1028: (4)               def set_name(self, name):
1029: (8)                   """Set /Name (icon) of annotation."""
1030: (8)                   CheckParent(self)
1031: (8)                   annot = self.this
1032: (8)                   annot_obj = mupdf.pdf_annot_obj(annot)
1033: (8)                   mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('Name'), name)
1034: (4)               def set_oc(self, oc=0):
1035: (8)                   """Set / remove annotation OC xref."""
1036: (8)                   CheckParent(self)
1037: (8)                   annot = self.this
1038: (8)                   annot_obj = mupdf.pdf_annot_obj(annot)
1039: (8)                   if not oc:
1040: (12)                      mupdf.pdf_dict_del(annot_obj, PDF_NAME('OC'))
1041: (8)                   else:
1042: (12)                      JM_add_oc_object(mupdf.pdf_get_bound_document(annot_obj),
annot_obj, oc)
1043: (4)               def set_opacity(self, opacity):
1044: (8)                   """Set opacity."""
1045: (8)                   CheckParent(self)
1046: (8)                   annot = self.this
1047: (8)                   if not _INRANGE(opacity, 0.0, 1.0):
1048: (12)                      mupdf.pdf_set_annot_opacity(annot, 1)
1049: (12)                      return
1050: (8)                   mupdf.pdf_set_annot_opacity(annot, opacity)
1051: (8)                   if opacity < 1.0:
1052: (12)                      page = mupdf.pdf_annot_page(annot)
1053: (12)                      page.transparency = 1
1054: (4)               def set_open(self, is_open):
1055: (8)                   """Set 'open' status of annotation or its Popup."""
1056: (8)                   CheckParent(self)
1057: (8)                   annot = self.this
1058: (8)                   mupdf.pdf_set_annot_is_open(annot, is_open)
1059: (4)               def set_popup(self, rect):
1060: (8)                   '''
1061: (8)                   Create annotation 'Popup' or update rectangle.
1062: (8)                   '''
1063: (8)                   CheckParent(self)
1064: (8)                   annot = self.this
1065: (8)                   pdfpage = mupdf.pdf_annot_page( annot)
1066: (8)                   rot = JM_rotate_page_matrix(pdfpage)
1067: (8)                   r = mupdf.fz_transform_rect(JM_rect_from_py(rect), rot)
1068: (8)                   mupdf.pdf_set_annot_popup(annot, r)
1069: (4)               def set_rect(self, rect):
1070: (8)                   """Set annotation rectangle."""
1071: (8)                   CheckParent(self)
1072: (8)                   annot = self.this
1073: (8)                   pdfpage = mupdf.pdf_annot_page(annot)
1074: (8)                   rot = JM_rotate_page_matrix(pdfpage)
1075: (8)                   r = mupdf.fz_transform_rect(JM_rect_from_py(rect), rot)
1076: (8)                   if mupdf.fz_is_empty_rect(r) or mupdf.fz_is_infinite_rect(r):
```

```
(12)                    raise ValueError( MSG_BAD_RECT)
(8)                 try:
(12)                    mupdf.pdf_set_annot_rect(annot, r)
(8)                 except Exception as e:
(12)                    message(f'cannot set rect: {e}')
(12)                    return False
(4)             def set_rotation(self, rotate=0):
(8)                 """Set annotation rotation."""
(8)                 CheckParent(self)
(8)                 annot = self.this
(8)                 type = mupdf.pdf_annot_type(annot)
(8)                 if type not in (
(16)                        mupdf.PDF_ANNOT_CARET,
(16)                        mupdf.PDF_ANNOT_CIRCLE,
(16)                        mupdf.PDF_ANNOT_FREE_TEXT,
(16)                        mupdf.PDF_ANNOT_FILE_ATTACHMENT,
(16)                        mupdf.PDF_ANNOT_INK,
(16)                        mupdf.PDF_ANNOT_LINE,
(16)                        mupdf.PDF_ANNOT_POLY_LINE,
(16)                        mupdf.PDF_ANNOT_POLYGON,
(16)                        mupdf.PDF_ANNOT_SQUARE,
(16)                        mupdf.PDF_ANNOT_STAMP,
(16)                        mupdf.PDF_ANNOT_TEXT,
(16)                        ):
(12)                    return
(8)                 rot = rotate
(8)                 while rot < 0:
(12)                    rot += 360
(8)                 while rot >= 360:
(12)                    rot -= 360
(8)                 if type == mupdf.PDF_ANNOT_FREE_TEXT and rot % 90 != 0:
(12)                    rot = 0
(8)                 annot_obj = mupdf.pdf_annot_obj(annot)
(8)                 mupdf.pdf_dict_put_int(annot_obj, PDF_NAME('Rotate'), rot)
(4)             @property
(4)             def type(self):
(8)                 """annotation type"""
(8)                 CheckParent(self)
(8)                 if not self.this.m_internal:
(12)                    return 'null'
(8)                 type_ = mupdf.pdf_annot_type(self.this)
(8)                 c = mupdf.pdf_string_from_annot_type(type_)
(8)                 o = mupdf.pdf_dict_gets( mupdf.pdf_annot_obj(self.this), 'IT')
(8)                 if not o.m_internal or mupdf.pdf_is_name(o):
(12)                    return (type_, c)
(8)                 it = mupdf.pdf_to_name(o)
(8)                 return (type_, c, it)
(4)             def update(self,
(12)                    blend_mode: OptStr =None,
(12)                    opacity: OptFloat =None,
(12)                    fontsize: float =0,
(12)                    fontname: OptStr =None,
(12)                    text_color: OptSeq =None,
(12)                    border_color: OptSeq =None,
(12)                    fill_color: OptSeq =None,
(12)                    cross_out: bool =True,
(12)                    rotate: int =-1,
(12)                    ):
(8)                 """Update annot appearance.
(8)                 Notes:
(12)                    Depending on the annot type, some parameters make no sense,
(12)                    while others are only available in this method to achieve the
(12)                    desired result. This is especially true for 'FreeText' annots.
(8)                 Args:
(12)                    blend_mode: set the blend mode, all annotations.
(12)                    opacity: set the opacity, all annotations.
(12)                    fontsize: set fontsize, 'FreeText' only.
(12)                    fontname: set the font, 'FreeText' only.
(12)                    border_color: set border color, 'FreeText' only.
```

```
(12)                    text_color: set text color, 'FreeText' only.
(12)                    fill_color: set fill color, all annotations.
(12)                    cross_out: draw diagonal lines, 'Redact' only.
(12)                    rotate: set rotation, 'FreeText' and some others.
(8)                 """
(8)                 Annot.update_timing_test()
(8)                 CheckParent(self)
(8)                 def color_string(cs, code):
(12)                    """Return valid PDF color operator for a given color sequence.
(12)                    """
(12)                    cc = ColorCode(cs, code)
(12)                    if not cc:
(16)                        return b""
(12)                    return (cc + "\n").encode()
(8)                 annot_type = self.type[0]  # get the annot type
(8)                 dt = self.border.get("dashes", None)  # get the dashes spec
(8)                 bwidth = self.border.get("width", -1)  # get border line width
(8)                 stroke = self.colors["stroke"]  # get the stroke color
(8)                 if fill_color is not None:
(12)                    fill = fill_color
(8)                 else:
(12)                    fill = self.colors["fill"]
(8)                 rect = None  # self.rect  # prevent MuPDF fiddling with it
(8)                 apnmat = self.apn_matrix  # prevent MuPDF fiddling with it
(8)                 if rotate != -1:  # sanitize rotation value
(12)                    while rotate < 0:
(16)                        rotate += 360
(12)                    while rotate >= 360:
(16)                        rotate -= 360
(12)                    if annot_type == mupdf.PDF_ANNOT_FREE_TEXT and rotate % 90 != 0:
(16)                        rotate = 0
(8)                 if blend_mode is None:
(12)                    blend_mode = self.blendmode
(8)                 if not hasattr(opacity, "__float__"):
(12)                    opacity = self.opacity
(8)                 if 0 <= opacity < 1 or blend_mode is not None:
(12)                    opa_code = "/H gs\n"  # then we must reference this 'gs'
(8)                 else:
(12)                    opa_code = ""
(8)                 if annot_type == mupdf.PDF_ANNOT_FREE_TEXT:
(12)                    CheckColor(border_color)
(12)                    CheckColor(text_color)
(12)                    CheckColor(fill_color)
(12)                    tcol, fname, fsize = TOOLS._parse_da(self)
(12)                    update_default_appearance = False
(12)                    if fsize <= 0:
(16)                        fsize = 12
(16)                        update_default_appearance = True
(12)                    if text_color is not None:
(16)                        tcol = text_color
(16)                        update_default_appearance = True
(12)                    if fontname is not None:
(16)                        fname = fontname
(16)                        update_default_appearance = True
(12)                    if fontsize > 0:
(16)                        fsize = fontsize
(16)                        update_default_appearance = True
(12)                    if update_default_appearance:
(16)                        da_str = ""
(16)                        if len(tcol) == 3:
(20)                            fmt = "{:g} {:g} {:g} rg /{f:s} {s:g} Tf"
(16)                        elif len(tcol) == 1:
(20)                            fmt = "{:g} g /{f:s} {s:g} Tf"
(16)                        elif len(tcol) == 4:
(20)                            fmt = "{:g} {:g} {:g} {:g} k /{f:s} {s:g} Tf"
(16)                        da_str = fmt.format(*tcol, f=fname, s=fsize)
(16)                        TOOLS._update_da(self, da_str)
(8)                 val = self._update_appearance(
(12)                    opacity=opacity,
```

```
1215: (12)                      blend_mode=blend_mode,
1216: (12)                      fill_color=fill,
1217: (12)                      rotate=rotate,
1218: (8)                   )
1219: (8)                   if val is False:
1220: (12)                      raise RuntimeError("Error updating annotation.")
1221: (8)                   bfill = color_string(fill, "f")
1222: (8)                   bstroke = color_string(stroke, "c")
1223: (8)                   p_ctm = self.get_parent().transformation_matrix
1224: (8)                   imat = ~p_ctm  # inverse page transf. matrix
1225: (8)                   if dt:
1226: (12)                      dashes = "[" + " ".join(map(str, dt)) + "] 0 d\n"
1227: (12)                      dashes = dashes.encode("utf-8")
1228: (8)                   else:
1229: (12)                      dashes = None
1230: (8)                   if self.line_ends:
1231: (12)                      line_end_le, line_end_ri = self.line_ends
1232: (8)                   else:
1233: (12)                      line_end_le, line_end_ri = 0, 0  # init line end codes
1234: (8)                   ap = self._getAP()
1235: (8)                   ap_tab = ap.splitlines()  # split in single lines
1236: (8)                   ap_updated = False  # assume we did nothing
1237: (8)                   if annot_type == mupdf.PDF_ANNOT_REDACT:
1238: (12)                      if cross_out:  # create crossed-out rect
1239: (16)                          ap_updated = True
1240: (16)                          ap_tab = ap_tab[:-1]
1241: (16)                          _, LL, LR, UR, UL = ap_tab
1242: (16)                          ap_tab.append(LR)
1243: (16)                          ap_tab.append(LL)
1244: (16)                          ap_tab.append(UR)
1245: (16)                          ap_tab.append(LL)
1246: (16)                          ap_tab.append(UL)
1247: (16)                          ap_tab.append(b"S")
1248: (12)                      if bwidth > 0 or bstroke != b"":
1249: (16)                          ap_updated = True
1250: (16)                          ntab = [_format_g(bwidth).encode() + b" w"] if bwidth > 0 else
[]
1251: (16)                          for line in ap_tab:
1252: (20)                              if line.endswith(b"w"):
1253: (24)                                  continue
1254: (20)                              if line.endswith(b"RG") and bstroke != b"":
1255: (24)                                  line = bstroke[:-1]
1256: (20)                              ntab.append(line)
1257: (16)                          ap_tab = ntab
1258: (12)                      ap = b"\n".join(ap_tab)
1259: (8)                   if annot_type == mupdf.PDF_ANNOT_FREE_TEXT:
1260: (12)                      BT = ap.find(b"BT")
1261: (12)                      ET = ap.rfind(b"ET") + 2
1262: (12)                      ap = ap[BT:ET]
1263: (12)                      w, h = self.rect.width, self.rect.height
1264: (12)                      if rotate in (90, 270) or not (apnmat.b == apnmat.c == 0):
1265: (16)                          w, h = h, w
1266: (12)                      re = b"0 0 " + _format_g((w, h)).encode() + b" re"
1267: (12)                      ap = re + b"\nW\nn\n" + ap
1268: (12)                      ope = None
1269: (12)                      fill_string = color_string(fill, "f")
1270: (12)                      if fill_string:
1271: (16)                          ope = b"f"
1272: (12)                      stroke_string = color_string(border_color, "c")
1273: (12)                      if stroke_string and bwidth > 0:
1274: (16)                          ope = b"S"
1275: (16)                          bwidth = _format_g(bwidth).encode() + b" w\n"
1276: (12)                      else:
1277: (16)                          bwidth = stroke_string = b""
1278: (12)                      if fill_string and stroke_string:
1279: (16)                          ope = b"B"
1280: (12)                      if ope is not None:
1281: (16)                          ap = bwidth + fill_string + stroke_string + re + b"\n" + ope +
b"\n" + ap
```

```
1282: (12)                      if dashes is not None:  # handle dashes
1283: (16)                          ap = dashes + b"\n" + ap
1284: (16)                          dashes = None
1285: (12)                      ap_updated = True
1286: (8)                   if annot_type in (mupdf.PDF_ANNOT_POLYGON, mupdf.PDF_ANNOT_POLY_LINE):
1287: (12)                      ap = b"\n".join(ap_tab[:-1]) + b"\n"
1288: (12)                      ap_updated = True
1289: (12)                      if bfill != b"":
1290: (16)                          if annot_type == mupdf.PDF_ANNOT_POLYGON:
1291: (20)                              ap = ap + bfill + b"b"  # close, fill, and stroke
1292: (16)                          elif annot_type == mupdf.PDF_ANNOT_POLY_LINE:
1293: (20)                              ap = ap + b"S"  # stroke
1294: (12)                      else:
1295: (16)                          if annot_type == mupdf.PDF_ANNOT_POLYGON:
1296: (20)                              ap = ap + b"s"  # close and stroke
1297: (16)                          elif annot_type == mupdf.PDF_ANNOT_POLY_LINE:
1298: (20)                              ap = ap + b"S"  # stroke
1299: (8)                   if dashes is not None:  # handle dashes
1300: (12)                      ap = dashes + ap
1301: (12)                      ap = ap.replace(b"\nS\n", b"\nS\n[] 0 d\n", 1)
1302: (12)                      ap_updated = True
1303: (8)                   if opa_code:
1304: (12)                      ap = opa_code.encode("utf-8") + ap
1305: (12)                      ap_updated = True
1306: (8)                   ap = b"q\n" + ap + b"\nQ\n"
1307: (8)                   if line_end_le + line_end_ri > 0 and annot_type in
(mupdf.PDF_ANNOT_POLYGON, mupdf.PDF_ANNOT_POLY_LINE):
1308: (12)                      le_funcs = (None, TOOLS._le_square, TOOLS._le_circle,
1309: (24)                                  TOOLS._le_diamond, TOOLS._le_openarrow,
1310: (24)                                  TOOLS._le_closedarrow, TOOLS._le_butt,
1311: (24)                                  TOOLS._le_ropenarrow, TOOLS._le_rclosedarrow,
1312: (24)                                  TOOLS._le_slash)
1313: (12)                      le_funcs_range = range(1, len(le_funcs))
1314: (12)                      d = 2 * max(1, self.border["width"])
1315: (12)                      rect = self.rect + (-d, -d, d, d)
1316: (12)                      ap_updated = True
1317: (12)                      points = self.vertices
1318: (12)                      if line_end_le in le_funcs_range:
1319: (16)                          p1 = Point(points[0]) * imat
1320: (16)                          p2 = Point(points[1]) * imat
1321: (16)                          left = le_funcs[line_end_le](self, p1, p2, False, fill_color)
1322: (16)                          ap += left.encode()
1323: (12)                      if line_end_ri in le_funcs_range:
1324: (16)                          p1 = Point(points[-2]) * imat
1325: (16)                          p2 = Point(points[-1]) * imat
1326: (16)                          left = le_funcs[line_end_ri](self, p1, p2, True, fill_color)
1327: (16)                          ap += left.encode()
1328: (8)                   if ap_updated:
1329: (12)                      if rect:                        # rect modified here?
1330: (16)                          self.set_rect(rect)
1331: (16)                          self._setAP(ap, rect=1)
1332: (12)                      else:
1333: (16)                          self._setAP(ap, rect=0)
1334: (8)                   if annot_type not in (  # only these types are supported
1335: (16)                          mupdf.PDF_ANNOT_CARET,
1336: (16)                          mupdf.PDF_ANNOT_CIRCLE,
1337: (16)                          mupdf.PDF_ANNOT_FILE_ATTACHMENT,
1338: (16)                          mupdf.PDF_ANNOT_INK,
1339: (16)                          mupdf.PDF_ANNOT_LINE,
1340: (16)                          mupdf.PDF_ANNOT_POLY_LINE,
1341: (16)                          mupdf.PDF_ANNOT_POLYGON,
1342: (16)                          mupdf.PDF_ANNOT_SQUARE,
1343: (16)                          mupdf.PDF_ANNOT_STAMP,
1344: (16)                          mupdf.PDF_ANNOT_TEXT,
1345: (16)                          ):
1346: (12)                      return
1347: (8)                   rot = self.rotation  # get value from annot object
1348: (8)                   if rot == -1:  # nothing to change
1349: (12)                      return
```

```
1350: (8)                    M = (self.rect.tl + self.rect.br) / 2  # center of annot rect
1351: (8)                    if rot == 0:  # undo rotations
1352: (12)                       if abs(apnmat - Matrix(1, 1)) < 1e-5:
1353: (16)                           return  # matrix already is a no-op
1354: (12)                       quad = self.rect.morph(M, ~apnmat)  # derotate rect
1355: (12)                       self.setRect(quad.rect)
1356: (12)                       self.set_apn_matrix(Matrix(1, 1))  # appearance matrix = no-op
1357: (12)                       return
1358: (8)                    mat = Matrix(rot)
1359: (8)                    quad = self.rect.morph(M, mat)
1360: (8)                    self.set_rect(quad.rect)
1361: (8)                    self.set_apn_matrix(apnmat * mat)
1362: (4)                def update_file(self, buffer_=None, filename=None, ufilename=None,
desc=None):
1363: (8)                    """Update attached file."""
1364: (8)                    CheckParent(self)
1365: (8)                    annot = self.this
1366: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
1367: (8)                    pdf = mupdf.pdf_get_bound_document(annot_obj)  # the owning PDF
1368: (8)                    type = mupdf.pdf_annot_type(annot)
1369: (8)                    if type != mupdf.PDF_ANNOT_FILE_ATTACHMENT:
1370: (12)                       raise TypeError( MSG_BAD_ANNOT_TYPE)
1371: (8)                    stream = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('FS'),
PDF_NAME('EF'), PDF_NAME('F'))
1372: (8)                    if not stream.m_internal:
1373: (12)                       RAISEPY( "bad PDF: no /EF object", JM_Exc_FileDataError)
1374: (8)                    fs = mupdf.pdf_dict_get(annot_obj, PDF_NAME('FS'))
1375: (8)                    res = JM_BufferFromBytes(buffer_)
1376: (8)                    if buffer_ and not res.m_internal:
1377: (12)                       raise ValueError( MSG_BAD_BUFFER)
1378: (8)                    if res:
1379: (12)                       JM_update_stream(pdf, stream, res, 1)
1380: (12)                       len, _ = mupdf.fz_buffer_storage(res)
1381: (12)                       l = mupdf.pdf_new_int(len)
1382: (12)                       mupdf.pdf_dict_put(stream, PDF_NAME('DL'), l)
1383: (12)                       mupdf.pdf_dict_putl(stream, l, PDF_NAME('Params'),
PDF_NAME('Size'))
1384: (8)                    if filename:
1385: (12)                       mupdf.pdf_dict_put_text_string(stream, PDF_NAME('F'), filename)
1386: (12)                       mupdf.pdf_dict_put_text_string(fs, PDF_NAME('F'), filename)
1387: (12)                       mupdf.pdf_dict_put_text_string(stream, PDF_NAME('UF'), filename)
1388: (12)                       mupdf.pdf_dict_put_text_string(fs, PDF_NAME('UF'), filename)
1389: (12)                       mupdf.pdf_dict_put_text_string(annot_obj, PDF_NAME('Contents'),
filename)
1390: (8)                    if ufilename:
1391: (12)                       mupdf.pdf_dict_put_text_string(stream, PDF_NAME('UF'), ufilename)
1392: (12)                       mupdf.pdf_dict_put_text_string(fs, PDF_NAME('UF'), ufilename)
1393: (8)                    if desc:
1394: (12)                       mupdf.pdf_dict_put_text_string(stream, PDF_NAME('Desc'), desc)
1395: (12)                       mupdf.pdf_dict_put_text_string(fs, PDF_NAME('Desc'), desc)
1396: (4)                @staticmethod
1397: (4)                def update_timing_test():
1398: (8)                    total = 0
1399: (8)                    for i in range( 30*1000):
1400: (12)                       total += i
1401: (8)                    return total
1402: (4)                @property
1403: (4)                def vertices(self):
1404: (8)                    """annotation vertex points"""
1405: (8)                    CheckParent(self)
1406: (8)                    annot = self.this
1407: (8)                    assert isinstance(annot, mupdf.PdfAnnot)
1408: (8)                    annot_obj = mupdf.pdf_annot_obj(annot)
1409: (8)                    page = mupdf.pdf_annot_page(annot)
1410: (8)                    page_ctm = mupdf.FzMatrix()   # page transformation matrix
1411: (8)                    dummy = mupdf.FzRect()  # Out-param for mupdf.pdf_page_transform().
1412: (8)                    mupdf.pdf_page_transform(page, dummy, page_ctm)
1413: (8)                    derot = JM_derotate_page_matrix(page)
1414: (8)                    page_ctm = mupdf.fz_concat(page_ctm, derot)
```

```
1415: (8)                   o = mupdf.pdf_dict_get(annot_obj, PDF_NAME('Vertices'))
1416: (8)                   if not o.m_internal:    o = mupdf.pdf_dict_get(annot_obj,
PDF_NAME('L'))
1417: (8)                   if not o.m_internal:    o = mupdf.pdf_dict_get(annot_obj,
PDF_NAME('QuadPoints'))
1418: (8)                   if not o.m_internal:    o = mupdf.pdf_dict_gets(annot_obj, 'CL')
1419: (8)                   if o.m_internal:
1420: (12)                      res = []
1421: (12)                      for i in range(0, mupdf.pdf_array_len(o), 2):
1422: (16)                          x = mupdf.pdf_to_real(mupdf.pdf_array_get(o, i))
1423: (16)                          y = mupdf.pdf_to_real(mupdf.pdf_array_get(o, i+1))
1424: (16)                          point = mupdf.FzPoint(x, y)
1425: (16)                          point = mupdf.fz_transform_point(point, page_ctm)
1426: (16)                          res.append( (point.x, point.y))
1427: (12)                      return res
1428: (8)                   o = mupdf.pdf_dict_gets(annot_obj, 'InkList')
1429: (8)                   if o.m_internal:
1430: (12)                      res = []
1431: (12)                      for i in range(mupdf.pdf_array_len(o)):
1432: (16)                          res1 = []
1433: (16)                          o1 = mupdf.pdf_array_get(o, i)
1434: (16)                          for j in range(0, mupdf.pdf_array_len(o1), 2):
1435: (20)                              x = mupdf.pdf_to_real(mupdf.pdf_array_get(o1, j))
1436: (20)                              y = mupdf.pdf_to_real(mupdf.pdf_array_get(o1, j+1))
1437: (20)                              point = mupdf.FzPoint(x, y)
1438: (20)                              point = mupdf.fz_transform_point(point, page_ctm)
1439: (20)                              res1.append( (point.x, point.y))
1440: (16)                          res.append(res1)
1441: (12)                      return res
1442: (4)               @property
1443: (4)               def xref(self):
1444: (8)                   """annotation xref number"""
1445: (8)                   CheckParent(self)
1446: (8)                   annot = self.this
1447: (8)                   return mupdf.pdf_to_num(mupdf.pdf_annot_obj(annot))
1448: (0)           class Archive:
1449: (4)               def __init__( self, *args):
1450: (8)                   '''
1451: (8)                   Archive(dirname [, path]) - from folder
1452: (8)                   Archive(file [, path]) - from file name or object
1453: (8)                   Archive(data, name) - from memory item
1454: (8)                   Archive() - empty archive
1455: (8)                   Archive(archive [, path]) - from archive
1456: (8)                   '''
1457: (8)                   self._subarchives = list()
1458: (8)                   self.this = mupdf.fz_new_multi_archive()
1459: (8)                   if args:
1460: (12)                      self.add( *args)
1461: (4)               def __repr__( self):
1462: (8)                   return f'Archive, sub-archives: {len(self._subarchives)}'
1463: (4)               def _add_arch( self, subarch, path=None):
1464: (8)                   mupdf.fz_mount_multi_archive( self.this, subarch, path)
1465: (4)               def _add_dir( self, folder, path=None):
1466: (8)                   sub = mupdf.fz_open_directory( folder)
1467: (8)                   mupdf.fz_mount_multi_archive( self.this, sub, path)
1468: (4)               def _add_treeitem( self, memory, name, path=None):
1469: (8)                   buff = JM_BufferFromBytes( memory)
1470: (8)                   sub = mupdf.fz_new_tree_archive( mupdf.FzTree())
1471: (8)                   mupdf.fz_tree_archive_add_buffer( sub, name, buff)
1472: (8)                   mupdf.fz_mount_multi_archive( self.this, sub, path)
1473: (4)               def _add_ziptarfile( self, filepath, type_, path=None):
1474: (8)                   if type_ == 1:
1475: (12)                      sub = mupdf.fz_open_zip_archive( filepath)
1476: (8)                   else:
1477: (12)                      sub = mupdf.fz_open_tar_archive( filepath)
1478: (8)                   mupdf.fz_mount_multi_archive( self.this, sub, path)
1479: (4)               def _add_ziptarmemory( self, memory, type_, path=None):
1480: (8)                   buff = JM_BufferFromBytes( memory)
1481: (8)                   stream = mupdf.fz_open_buffer( buff)
```

```
(8)                 if type_==1:
(12)                    sub = mupdf.fz_open_zip_archive_with_stream( stream)
(8)                 else:
(12)                    sub = mupdf.fz_open_tar_archive_with_stream( stream)
(8)                 mupdf.fz_mount_multi_archive( self.this, sub, path)
(4)             def add( self, content, path=None):
(8)                 '''
(8)                 Add a sub-archive.
(8)                 Args:
(12)                    content:
(16)                        The content to be added. May be one of:
(20)                            `str` - must be path of directory or file.
(20)                            `bytes`, `bytearray`, `io.BytesIO` - raw data.
(20)                            `zipfile.Zipfile`.
(20)                            `tarfile.TarFile`.
(20)                            `pymupdf.Archive`.
(20)                            A two-item tuple `(data, name)`.
(20)                            List or tuple (but not tuple with length 2) of the above.
(12)                    path: (str) a "virtual" path name, under which the elements
(16)                        of content can be retrieved. Use it to e.g. cope with
(16)                        duplicate element names.
(8)                 '''
(8)                 def is_binary_data(x):
(12)                    return isinstance(x, (bytes, bytearray, io.BytesIO))
(8)                 def make_subarch(entries, mount, fmt):
(12)                    subarch = dict(fmt=fmt, entries=entries, path=mount)
(12)                    if fmt != "tree" or self._subarchives == []:
(16)                        self._subarchives.append(subarch)
(12)                    else:
(16)                        ltree = self._subarchives[-1]
(16)                        if ltree["fmt"] != "tree" or ltree["path"] != subarch["path"]:
(20)                            self._subarchives.append(subarch)
(16)                        else:
(20)                            ltree["entries"].extend(subarch["entries"])
(20)                            self._subarchives[-1] = ltree
(8)                 if isinstance(content, pathlib.Path):
(12)                    content = str(content)
(8)                 if isinstance(content, str):
(12)                    if os.path.isdir(content):
(16)                        self._add_dir(content, path)
(16)                        return make_subarch(os.listdir(content), path, 'dir')
(12)                    elif os.path.isfile(content):
(16)                        assert isinstance(path, str) and path != '', \
(24)                                f'Need name for binary content, but {path=}.'
(16)                        with open(content) as f:
(20)                            ff = f.read()
(16)                        self._add_treeitem(ff, path)
(16)                        return make_subarch([path], None, 'tree')
(12)                    else:
(16)                        raise ValueError(f'Not a file or directory: {content!r}')
(8)                 elif is_binary_data(content):
(12)                    assert isinstance(path, str) and path != '' \
(20)                            f'Need name for binary content, but {path=}.'
(12)                    self._add_treeitem(content, path)
(12)                    return make_subarch([path], None, 'tree')
(8)                 elif isinstance(content, zipfile.ZipFile):
(12)                    filename = getattr(content, "filename", None)
(12)                    if filename is None:
(16)                        fp = content.fp.getvalue()
(16)                        self._add_ziptarmemory(fp, 1, path)
(12)                    else:
(16)                        self._add_ziptarfile(filename, 1, path)
(12)                    return make_subarch(content.namelist(), path, 'zip')
(8)                 elif isinstance(content, tarfile.TarFile):
(12)                    filename = getattr(content.fileobj, "name", None)
(12)                    if filename is None:
(16)                        fp = content.fileobj
(16)                        if not isinstance(fp, io.BytesIO):
(20)                            fp = fp.fileobj
```

```
1551: (16)                          self._add_ziptarmemory(fp.getvalue(), 0, path)
1552: (12)                      else:
1553: (16)                          self._add_ziptarfile(filename, 0, path)
1554: (12)                      return make_subarch(content.getnames(), path, 'tar')
1555: (8)                   elif isinstance(content, Archive):
1556: (12)                      self._add_arch(content, path)
1557: (12)                      return make_subarch([], path, 'multi')
1558: (8)                   if isinstance(content, tuple) and len(content) == 2:
1559: (12)                      data, name = content
1560: (12)                      assert isinstance(name, str), f'Unexpected {type(name)=}'
1561: (12)                      if is_binary_data(data):
1562: (16)                          self._add_treeitem(data, name, path=path)
1563: (12)                      elif isinstance(data, str):
1564: (16)                          if os.path.isfile(data):
1565: (20)                              with open(data, 'rb') as f:
1566: (24)                                  ff = f.read()
1567: (20)                              self._add_treeitem(ff, name, path=path)
1568: (12)                      else:
1569: (16)                          assert 0, f'Unexpected {type(data)=}.'
1570: (12)                      return make_subarch([name], path, 'tree')
1571: (8)                   elif hasattr(content, '__getitem__'):
1572: (12)                      for item in content:
1573: (16)                          self.add(item, path)
1574: (12)                      return
1575: (8)                   else:
1576: (12)                      raise TypeError(f'Unrecognised type {type(content)}.')
1577: (8)                   assert 0
1578: (4)               @property
1579: (4)               def entry_list( self):
1580: (8)                   '''
1581: (8)                   List of sub archives.
1582: (8)                   '''
1583: (8)                   return self._subarchives
1584: (4)               def has_entry( self, name):
1585: (8)                   return mupdf.fz_has_archive_entry( self.this, name)
1586: (4)               def read_entry( self, name):
1587: (8)                   buff = mupdf.fz_read_archive_entry( self.this, name)
1588: (8)                   return JM_BinFromBuffer( buff)
1589: (0)           class Xml:
1590: (4)               def __enter__(self):
1591: (8)                   return self
1592: (4)               def __exit__(self, *args):
1593: (8)                   pass
1594: (4)               def __init__( self, rhs):
1595: (8)                   if isinstance( rhs, mupdf.FzXml):
1596: (12)                      self.this = rhs
1597: (8)                   elif isinstance( str):
1598: (12)                      buff = mupdf.fz_new_buffer_from_copied_data( rhs)
1599: (12)                      self.this = mupdf.fz_parse_xml_from_html5( buff)
1600: (8)                   else:
1601: (12)                      assert 0, f'Unsupported type for rhs: {type(rhs)}'
1602: (4)               def _get_node_tree( self):
1603: (8)                   def show_node(node, items, shift):
1604: (12)                      while node is not None:
1605: (16)                          if node.is_text:
1606: (20)                              items.append((shift, f'"{node.text}"'))
1607: (20)                              node = node.next
1608: (20)                              continue
1609: (16)                          items.append((shift, f"({node.tagname}"))
1610: (16)                          for k, v in node.get_attributes().items():
1611: (20)                              items.append((shift, f"={k} '{v}'"))
1612: (16)                          child = node.first_child
1613: (16)                          if child:
1614: (20)                              items = show_node(child, items, shift + 1)
1615: (16)                          items.append((shift, f"){node.tagname}"))
1616: (16)                          node = node.next
1617: (12)                      return items
1618: (8)                   shift = 0
1619: (8)                   items = []
```

```
1620: (8)                   items = show_node(self, items, shift)
1621: (8)                   return items
1622: (4)               def add_bullet_list(self):
1623: (8)                   """Add bulleted list ("ul" tag)"""
1624: (8)                   child = self.create_element("ul")
1625: (8)                   self.append_child(child)
1626: (8)                   return child
1627: (4)               def add_class(self, text):
1628: (8)                   """Set some class via CSS. Replaces complete class spec."""
1629: (8)                   cls = self.get_attribute_value("class")
1630: (8)                   if cls is not None and text in cls:
1631: (12)                      return self
1632: (8)                   self.remove_attribute("class")
1633: (8)                   if cls is None:
1634: (12)                      cls = text
1635: (8)                   else:
1636: (12)                      cls += " " + text
1637: (8)                   self.set_attribute("class", cls)
1638: (8)                   return self
1639: (4)               def add_code(self, text=None):
1640: (8)                   """Add a "code" tag"""
1641: (8)                   child = self.create_element("code")
1642: (8)                   if type(text) is str:
1643: (12)                      child.append_child(self.create_text_node(text))
1644: (8)                   prev = self.span_bottom()
1645: (8)                   if prev is None:
1646: (12)                      prev = self
1647: (8)                   prev.append_child(child)
1648: (8)                   return self
1649: (4)               def add_codeblock(self):
1650: (8)                   """Add monospaced lines ("pre" node)"""
1651: (8)                   child = self.create_element("pre")
1652: (8)                   self.append_child(child)
1653: (8)                   return child
1654: (4)               def add_description_list(self):
1655: (8)                   """Add description list ("dl" tag)"""
1656: (8)                   child = self.create_element("dl")
1657: (8)                   self.append_child(child)
1658: (8)                   return child
1659: (4)               def add_division(self):
1660: (8)                   """Add "div" tag"""
1661: (8)                   child = self.create_element("div")
1662: (8)                   self.append_child(child)
1663: (8)                   return child
1664: (4)               def add_header(self, level=1):
1665: (8)                   """Add header tag"""
1666: (8)                   if level not in range(1, 7):
1667: (12)                      raise ValueError("Header level must be in [1, 6]")
1668: (8)                   this_tag = self.tagname
1669: (8)                   new_tag = f"h{level}"
1670: (8)                   child = self.create_element(new_tag)
1671: (8)                   if this_tag not in ("h1", "h2", "h3", "h4", "h5", "h6", "p"):
1672: (12)                      self.append_child(child)
1673: (12)                      return child
1674: (8)                   self.parent.append_child(child)
1675: (8)                   return child
1676: (4)               def add_horizontal_line(self):
1677: (8)                   """Add horizontal line ("hr" tag)"""
1678: (8)                   child = self.create_element("hr")
1679: (8)                   self.append_child(child)
1680: (8)                   return child
1681: (4)               def add_image(self, name, width=None, height=None, imgfloat=None,
align=None):
1682: (8)                   """Add image node (tag "img")."""
1683: (8)                   child = self.create_element("img")
1684: (8)                   if width is not None:
1685: (12)                      child.set_attribute("width", f"{width}")
1686: (8)                   if height is not None:
1687: (12)                      child.set_attribute("height", f"{height}")
```

```
(8)                 if imgfloat is not None:
(12)                    child.set_attribute("style", f"float: {imgfloat}")
(8)                 if align is not None:
(12)                    child.set_attribute("align", f"{align}")
(8)                 child.set_attribute("src", f"{name}")
(8)                 self.append_child(child)
(8)                 return child
(4)             def add_link(self, href, text=None):
(8)                 """Add a hyperlink ("a" tag)"""
(8)                 child = self.create_element("a")
(8)                 if not isinstance(text, str):
(12)                    text = href
(8)                 child.set_attribute("href", href)
(8)                 child.append_child(self.create_text_node(text))
(8)                 prev = self.span_bottom()
(8)                 if prev is None:
(12)                    prev = self
(8)                 prev.append_child(child)
(8)                 return self
(4)             def add_list_item(self):
(8)                 """Add item ("li" tag) under a (numbered or bulleted) list."""
(8)                 if self.tagname not in ("ol", "ul"):
(12)                    raise ValueError("cannot add list item to", self.tagname)
(8)                 child = self.create_element("li")
(8)                 self.append_child(child)
(8)                 return child
(4)             def add_number_list(self, start=1, numtype=None):
(8)                 """Add numbered list ("ol" tag)"""
(8)                 child = self.create_element("ol")
(8)                 if start > 1:
(12)                    child.set_attribute("start", str(start))
(8)                 if numtype is not None:
(12)                    child.set_attribute("type", numtype)
(8)                 self.append_child(child)
(8)                 return child
(4)             def add_paragraph(self):
(8)                 """Add "p" tag"""
(8)                 child = self.create_element("p")
(8)                 if self.tagname != "p":
(12)                    self.append_child(child)
(8)                 else:
(12)                    self.parent.append_child(child)
(8)                 return child
(4)             def add_span(self):
(8)                 child = self.create_element("span")
(8)                 self.append_child(child)
(8)                 return child
(4)             def add_style(self, text):
(8)                 """Set some style via CSS style. Replaces complete style spec."""
(8)                 style = self.get_attribute_value("style")
(8)                 if style is not None and text in style:
(12)                    return self
(8)                 self.remove_attribute("style")
(8)                 if style is None:
(12)                    style = text
(8)                 else:
(12)                    style += ";" + text
(8)                 self.set_attribute("style", style)
(8)                 return self
(4)             def add_subscript(self, text=None):
(8)                 """Add a subscript ("sub" tag)"""
(8)                 child = self.create_element("sub")
(8)                 if type(text) is str:
(12)                    child.append_child(self.create_text_node(text))
(8)                 prev = self.span_bottom()
(8)                 if prev is None:
(12)                    prev = self
(8)                 prev.append_child(child)
(8)                 return self
```

```
(4)     def add_superscript(self, text=None):
(8)         """Add a superscript ("sup" tag)"""
(8)         child = self.create_element("sup")
(8)         if type(text) is str:
(12)            child.append_child(self.create_text_node(text))
(8)         prev = self.span_bottom()
(8)         if prev is None:
(12)            prev = self
(8)         prev.append_child(child)
(8)         return self
(4)     def add_text(self, text):
(8)         """Add text. Line breaks are honored."""
(8)         lines = text.splitlines()
(8)         line_count = len(lines)
(8)         prev = self.span_bottom()
(8)         if prev is None:
(12)            prev = self
(8)         for i, line in enumerate(lines):
(12)            prev.append_child(self.create_text_node(line))
(12)            if i < line_count - 1:
(16)                prev.append_child(self.create_element("br"))
(8)         return self
(4)     def append_child( self, child):
(8)         mupdf.fz_dom_append_child( self.this, child.this)
(4)     def append_styled_span(self, style):
(8)         span = self.create_element("span")
(8)         span.add_style(style)
(8)         prev = self.span_bottom()
(8)         if prev is None:
(12)            prev = self
(8)         prev.append_child(span)
(8)         return prev
(4)     def bodytag( self):
(8)         return Xml( mupdf.fz_dom_body( self.this))
(4)     def clone( self):
(8)         ret = mupdf.fz_dom_clone( self.this)
(8)         return Xml( ret)
(4)     @staticmethod
(4)     def color_text(color):
(8)         if type(color) is str:
(12)            return color
(8)         if type(color) is int:
(12)            return f"rgb({sRGB_to_rgb(color)})"
(8)         if type(color) in (tuple, list):
(12)            return f"rgb{tuple(color)}"
(8)         return color
(4)     def create_element( self, tag):
(8)         return Xml( mupdf.fz_dom_create_element( self.this, tag))
(4)     def create_text_node( self, text):
(8)         return Xml( mupdf.fz_dom_create_text_node( self.this, text))
(4)     def debug(self):
(8)         """Print a list of the node tree below self."""
(8)         items = self._get_node_tree()
(8)         for item in items:
(12)            message("  " * item[0] + item[1].replace("\n", "\\n"))
(4)     def find( self, tag, att, match):
(8)         ret = mupdf.fz_dom_find( self.this, tag, att, match)
(8)         if ret.m_internal:
(12)            return Xml( ret)
(4)     def find_next( self, tag, att, match):
(8)         ret = mupdf.fz_dom_find_next( self.this, tag, att, match)
(8)         if ret.m_internal:
(12)            return Xml( ret)
(4)     @property
(4)     def first_child( self):
(8)         if mupdf.fz_xml_text( self.this):
(12)            return
(8)         ret = mupdf.fz_dom_first_child( self)
(8)         if ret.m_internal:
```

```
(12)                        return Xml( ret)
(4)                 def get_attribute_value( self, key):
(8)                     assert key
(8)                     return mupdf.fz_dom_attribute( self.this, key)
(4)                 def get_attributes( self):
(8)                     if mupdf.fz_xml_text( self.this):
(12)                        return
(8)                     result = dict()
(8)                     i = 0
(8)                     while 1:
(12)                        val, key = mupdf.fz_dom_get_attribute( self.this, i)
(12)                        if not val or not key:
(16)                            break
(12)                        result[ key] = val
(12)                        i += 1
(8)                     return result
(4)                 def insert_after( self, node):
(8)                     mupdf.fz_dom_insert_after( self.this, node.this)
(4)                 def insert_before( self, node):
(8)                     mupdf.fz_dom_insert_before( self.this, node.this)
(4)                 def insert_text(self, text):
(8)                     lines = text.splitlines()
(8)                     line_count = len(lines)
(8)                     for i, line in enumerate(lines):
(12)                        self.append_child(self.create_text_node(line))
(12)                        if i < line_count - 1:
(16)                            self.append_child(self.create_element("br"))
(8)                     return self
(4)                 @property
(4)                 def is_text(self):
(8)                     """Check if this is a text node."""
(8)                     return self.text is not None
(4)                 @property
(4)                 def last_child(self):
(8)                     """Return last child node."""
(8)                     child = self.first_child
(8)                     if child is None:
(12)                        return None
(8)                     while True:
(12)                        next = child.next
(12)                        if not next:
(16)                            return child
(12)                        child = next
(4)                 @property
(4)                 def next( self):
(8)                     ret = mupdf.fz_dom_next( self.this)
(8)                     if ret.m_internal:
(12)                        return Xml( ret)
(4)                 @property
(4)                 def parent( self):
(8)                     ret = mupdf.fz_dom_parent( self.this)
(8)                     if ret.m_internal:
(12)                        return Xml( ret)
(4)                 @property
(4)                 def previous( self):
(8)                     ret = mupdf.fz_dom_previous( self.this)
(8)                     if ret.m_internal:
(12)                        return Xml( ret)
(4)                 def remove( self):
(8)                     mupdf.fz_dom_remove( self.this)
(4)                 def remove_attribute( self, key):
(8)                     assert key
(8)                     mupdf.fz_dom_remove_attribute( self.this, key)
(4)                 @property
(4)                 def root( self):
(8)                     return Xml( mupdf.fz_xml_root( self.this))
(4)                 def set_align(self, align):
(8)                     """Set text alignment via CSS style"""
(8)                     text = "text-align: %s"
```

```
1895: (8)                   if isinstance( align, str):
1896: (12)                      t = align
1897: (8)                   elif align == TEXT_ALIGN_LEFT:
1898: (12)                      t = "left"
1899: (8)                   elif align == TEXT_ALIGN_CENTER:
1900: (12)                      t = "center"
1901: (8)                   elif align == TEXT_ALIGN_RIGHT:
1902: (12)                      t = "right"
1903: (8)                   elif align == TEXT_ALIGN_JUSTIFY:
1904: (12)                      t = "justify"
1905: (8)                   else:
1906: (12)                      raise ValueError(f"Unrecognised {align=}")
1907: (8)                   text = text % t
1908: (8)                   self.add_style(text)
1909: (8)                   return self
1910: (4)               def set_attribute( self, key, value):
1911: (8)                   assert key
1912: (8)                   mupdf.fz_dom_add_attribute( self.this, key, value)
1913: (4)               def set_bgcolor(self, color):
1914: (8)                   """Set background color via CSS style"""
1915: (8)                   text = f"background-color: %s" % self.color_text(color)
1916: (8)                   self.add_style(text)  # does not work on span level
1917: (8)                   return self
1918: (4)               def set_bold(self, val=True):
1919: (8)                   """Set bold on / off via CSS style"""
1920: (8)                   if val:
1921: (12)                      val="bold"
1922: (8)                   else:
1923: (12)                      val="normal"
1924: (8)                   text = "font-weight: %s" % val
1925: (8)                   self.append_styled_span(text)
1926: (8)                   return self
1927: (4)               def set_color(self, color):
1928: (8)                   """Set text color via CSS style"""
1929: (8)                   text = f"color: %s" % self.color_text(color)
1930: (8)                   self.append_styled_span(text)
1931: (8)                   return self
1932: (4)               def set_columns(self, cols):
1933: (8)                   """Set number of text columns via CSS style"""
1934: (8)                   text = f"columns: {cols}"
1935: (8)                   self.append_styled_span(text)
1936: (8)                   return self
1937: (4)               def set_font(self, font):
1938: (8)                   """Set font-family name via CSS style"""
1939: (8)                   text = "font-family: %s" % font
1940: (8)                   self.append_styled_span(text)
1941: (8)                   return self
1942: (4)               def set_fontsize(self, fontsize):
1943: (8)                   """Set font size name via CSS style"""
1944: (8)                   if type(fontsize) is str:
1945: (12)                      px=""
1946: (8)                   else:
1947: (12)                      px="px"
1948: (8)                   text = f"font-size: {fontsize}{px}"
1949: (8)                   self.append_styled_span(text)
1950: (8)                   return self
1951: (4)               def set_id(self, unique):
1952: (8)                   """Set a unique id."""
1953: (8)                   root = self.root
1954: (8)                   if root.find(None, "id", unique):
1955: (12)                      raise ValueError(f"id '{unique}' already exists")
1956: (8)                   self.set_attribute("id", unique)
1957: (8)                   return self
1958: (4)               def set_italic(self, val=True):
1959: (8)                   """Set italic on / off via CSS style"""
1960: (8)                   if val:
1961: (12)                      val="italic"
1962: (8)                   else:
1963: (12)                      val="normal"
```

```
(8)                 text = "font-style: %s" % val
(8)                 self.append_styled_span(text)
(8)                 return self
(4)             def set_leading(self, leading):
(8)                 """Set inter-line spacing value via CSS style - block-level only."""
(8)                 text = f"-mupdf-leading: {leading}"
(8)                 self.add_style(text)
(8)                 return self
(4)             def set_letter_spacing(self, spacing):
(8)                 """Set inter-letter spacing value via CSS style"""
(8)                 text = f"letter-spacing: {spacing}"
(8)                 self.append_styled_span(text)
(8)                 return self
(4)             def set_lineheight(self, lineheight):
(8)                 """Set line height name via CSS style - block-level only."""
(8)                 text = f"line-height: {lineheight}"
(8)                 self.add_style(text)
(8)                 return self
(4)             def set_margins(self, val):
(8)                 """Set margin values via CSS style"""
(8)                 text = "margins: %s" % val
(8)                 self.append_styled_span(text)
(8)                 return self
(4)             def set_opacity(self, opacity):
(8)                 """Set opacity via CSS style"""
(8)                 text = f"opacity: {opacity}"
(8)                 self.append_styled_span(text)
(8)                 return self
(4)             def set_pagebreak_after(self):
(8)                 """Insert a page break after this node."""
(8)                 text = "page-break-after: always"
(8)                 self.add_style(text)
(8)                 return self
(4)             def set_pagebreak_before(self):
(8)                 """Insert a page break before this node."""
(8)                 text = "page-break-before: always"
(8)                 self.add_style(text)
(8)                 return self
(4)             def set_properties(
(12)                    self,
(12)                    align=None,
(12)                    bgcolor=None,
(12)                    bold=None,
(12)                    color=None,
(12)                    columns=None,
(12)                    font=None,
(12)                    fontsize=None,
(12)                    indent=None,
(12)                    italic=None,
(12)                    leading=None,
(12)                    letter_spacing=None,
(12)                    lineheight=None,
(12)                    margins=None,
(12)                    pagebreak_after=None,
(12)                    pagebreak_before=None,
(12)                    word_spacing=None,
(12)                    unqid=None,
(12)                    cls=None,
(12)                    ):
(8)                 """Set any or all properties of a node.
(8)                 To be used for existing nodes preferably.
(8)                 """
(8)                 root = self.root
(8)                 temp = root.add_division()
(8)                 if align is not None:
(12)                    temp.set_align(align)
(8)                 if bgcolor is not None:
(12)                    temp.set_bgcolor(bgcolor)
(8)                 if bold is not None:
```

```
2033: (12)                      temp.set_bold(bold)
2034: (8)                   if color is not None:
2035: (12)                      temp.set_color(color)
2036: (8)                   if columns is not None:
2037: (12)                      temp.set_columns(columns)
2038: (8)                   if font is not None:
2039: (12)                      temp.set_font(font)
2040: (8)                   if fontsize is not None:
2041: (12)                      temp.set_fontsize(fontsize)
2042: (8)                   if indent is not None:
2043: (12)                      temp.set_text_indent(indent)
2044: (8)                   if italic is not None:
2045: (12)                      temp.set_italic(italic)
2046: (8)                   if leading is not None:
2047: (12)                      temp.set_leading(leading)
2048: (8)                   if letter_spacing is not None:
2049: (12)                      temp.set_letter_spacing(letter_spacing)
2050: (8)                   if lineheight is not None:
2051: (12)                      temp.set_lineheight(lineheight)
2052: (8)                   if margins is not None:
2053: (12)                      temp.set_margins(margins)
2054: (8)                   if pagebreak_after is not None:
2055: (12)                      temp.set_pagebreak_after()
2056: (8)                   if pagebreak_before is not None:
2057: (12)                      temp.set_pagebreak_before()
2058: (8)                   if word_spacing is not None:
2059: (12)                      temp.set_word_spacing(word_spacing)
2060: (8)                   if unqid is not None:
2061: (12)                      self.set_id(unqid)
2062: (8)                   if cls is not None:
2063: (12)                      self.add_class(cls)
2064: (8)                   styles = []
2065: (8)                   top_style = temp.get_attribute_value("style")
2066: (8)                   if top_style is not None:
2067: (12)                      styles.append(top_style)
2068: (8)                   child = temp.first_child
2069: (8)                   while child:
2070: (12)                      styles.append(child.get_attribute_value("style"))
2071: (12)                      child = child.first_child
2072: (8)                   self.set_attribute("style", ";".join(styles))
2073: (8)                   temp.remove()
2074: (8)                   return self
2075: (4)               def set_text_indent(self, indent):
2076: (8)                   """Set text indentation name via CSS style - block-level only."""
2077: (8)                   text = f"text-indent: {indent}"
2078: (8)                   self.add_style(text)
2079: (8)                   return self
2080: (4)               def set_underline(self, val="underline"):
2081: (8)                   text = "text-decoration: %s" % val
2082: (8)                   self.append_styled_span(text)
2083: (8)                   return self
2084: (4)               def set_word_spacing(self, spacing):
2085: (8)                   """Set inter-word spacing value via CSS style"""
2086: (8)                   text = f"word-spacing: {spacing}"
2087: (8)                   self.append_styled_span(text)
2088: (8)                   return self
2089: (4)               def span_bottom(self):
2090: (8)                   """Find deepest level in stacked spans."""
2091: (8)                   parent = self
2092: (8)                   child = self.last_child
2093: (8)                   if child is None:
2094: (12)                      return None
2095: (8)                   while child.is_text:
2096: (12)                      child = child.previous
2097: (12)                      if child is None:
2098: (16)                          break
2099: (8)                   if child is None or child.tagname != "span":
2100: (12)                      return None
2101: (8)                   while True:
```

```
2102: (12)                      if child is None:
2103: (16)                          return parent
2104: (12)                      if child.tagname in ("a", "sub","sup","body") or child.is_text:
2105: (16)                          child = child.next
2106: (16)                          continue
2107: (12)                      if child.tagname == "span":
2108: (16)                          parent = child
2109: (16)                          child = child.first_child
2110: (12)                      else:
2111: (16)                          return parent
2112: (4)               @property
2113: (4)               def tagname( self):
2114: (8)                   return mupdf.fz_xml_tag( self.this)
2115: (4)               @property
2116: (4)               def text( self):
2117: (8)                   return mupdf.fz_xml_text( self.this)
2118: (4)               add_var = add_code
2119: (4)               add_samp = add_code
2120: (4)               add_kbd = add_code
2121: (0)           class Colorspace:
2122: (4)               def __init__(self, type_):
2123: (8)                   """Supported are GRAY, RGB and CMYK."""
2124: (8)                   if isinstance( type_, mupdf.FzColorspace):
2125: (12)                      self.this = type_
2126: (8)                   elif type_ == CS_GRAY:
2127: (12)                      self.this = mupdf.FzColorspace(mupdf.FzColorspace.Fixed_GRAY)
2128: (8)                   elif type_ == CS_CMYK:
2129: (12)                      self.this = mupdf.FzColorspace(mupdf.FzColorspace.Fixed_CMYK)
2130: (8)                   elif type_ == CS_RGB:
2131: (12)                      self.this = mupdf.FzColorspace(mupdf.FzColorspace.Fixed_RGB)
2132: (8)                   else:
2133: (12)                      self.this = mupdf.FzColorspace(mupdf.FzColorspace.Fixed_RGB)
2134: (4)               def __repr__(self):
2135: (8)                   x = ("", "GRAY", "", "RGB", "CMYK")[self.n]
2136: (8)                   return "Colorspace(CS_%s) - %s" % (x, self.name)
2137: (4)               def _name(self):
2138: (8)                   return mupdf.fz_colorspace_name(self.this)
2139: (4)               @property
2140: (4)               def n(self):
2141: (8)                   """Size of one pixel."""
2142: (8)                   return mupdf.fz_colorspace_n(self.this)
2143: (4)               @property
2144: (4)               def name(self):
2145: (8)                   """Name of the Colorspace."""
2146: (8)                   return self._name()
2147: (0)           class DeviceWrapper:
2148: (4)               def __init__(self, *args):
2149: (8)                   if args_match( args, mupdf.FzDevice):
2150: (12)                      device, = args
2151: (12)                      self.this = device
2152: (8)                   elif args_match( args, Pixmap, None):
2153: (12)                      pm, clip = args
2154: (12)                      bbox = JM_irect_from_py( clip)
2155: (12)                      if mupdf.fz_is_infinite_irect( bbox):
2156: (16)                          self.this = mupdf.fz_new_draw_device( mupdf.FzMatrix(), pm)
2157: (12)                      else:
2158: (16)                          self.this = mupdf.fz_new_draw_device_with_bbox(
mupdf.FzMatrix(), pm, bbox)
2159: (8)                   elif args_match( args, mupdf.FzDisplayList):
2160: (12)                      dl, = args
2161: (12)                      self.this = mupdf.fz_new_list_device( dl)
2162: (8)                   elif args_match( args, mupdf.FzStextPage, None):
2163: (12)                      tp, flags = args
2164: (12)                      opts = mupdf.FzStextOptions( flags)
2165: (12)                      self.this = mupdf.fz_new_stext_device( tp, opts)
2166: (8)                   else:
2167: (12)                      raise Exception( f'Unrecognised args for DeviceWrapper: {args!r}')
2168: (0)           class DisplayList:
2169: (4)               def __del__(self):
```

```
2170: (8)                   if not type(self) is DisplayList: return
2171: (8)                   self.thisown = False
2172: (4)               def __init__(self, *args):
2173: (8)                   if len(args) == 1 and isinstance(args[0], mupdf.FzRect):
2174: (12)                      self.this = mupdf.FzDisplayList(args[0])
2175: (8)                   elif len(args) == 1 and isinstance(args[0], mupdf.FzDisplayList):
2176: (12)                      self.this = args[0]
2177: (8)                   else:
2178: (12)                      assert 0, f'Unrecognised {args=}'
2179: (4)               def get_pixmap(self, matrix=None, colorspace=None, alpha=0, clip=None):
2180: (8)                   if isinstance(colorspace, Colorspace):
2181: (12)                      colorspace = colorspace.this
2182: (8)                   else:
2183: (12)                      colorspace = mupdf.FzColorspace(mupdf.FzColorspace.Fixed_RGB)
2184: (8)                   val = JM_pixmap_from_display_list(self.this, matrix, colorspace,
alpha, clip, None)
2185: (8)                   val.thisown = True
2186: (8)                   return val
2187: (4)               def get_textpage(self, flags=3):
2188: (8)                   """Make a TextPage from a DisplayList."""
2189: (8)                   stext_options = mupdf.FzStextOptions()
2190: (8)                   stext_options.flags = flags
2191: (8)                   val = mupdf.FzStextPage(self.this, stext_options)
2192: (8)                   val.thisown = True
2193: (8)                   return val
2194: (4)               @property
2195: (4)               def rect(self):
2196: (8)                   val = JM_py_from_rect(mupdf.fz_bound_display_list(self.this))
2197: (8)                   val = Rect(val)
2198: (8)                   return val
2199: (4)               def run(self, dw, m, area):
2200: (8)                   mupdf.fz_run_display_list(
2201: (16)                          self.this,
2202: (16)                          dw.device,
2203: (16)                          JM_matrix_from_py(m),
2204: (16)                          JM_rect_from_py(area),
2205: (16)                          mupdf.FzCookie(),
2206: (16)                          )
2207: (0)           if g_use_extra:
2208: (4)               extra_FzDocument_insert_pdf = extra.FzDocument_insert_pdf
2209: (0)           class Document:
2210: (4)               def __contains__(self, loc) -> bool:
2211: (8)                   if type(loc) is int:
2212: (12)                      if loc < self.page_count:
2213: (16)                          return True
2214: (12)                      return False
2215: (8)                   if type(loc) not in (tuple, list) or len(loc) != 2:
2216: (12)                      return False
2217: (8)                   chapter, pno = loc
2218: (8)                   if (0
2219: (16)                          or not isinstance(chapter, int)
2220: (16)                          or chapter < 0
2221: (16)                          or chapter >= self.chapter_count
2222: (16)                          ):
2223: (12)                      return False
2224: (8)                   if (0
2225: (16)                          or not isinstance(pno, int)
2226: (16)                          or pno < 0
2227: (16)                          or pno >= self.chapter_page_count(chapter)
2228: (16)                          ):
2229: (12)                      return False
2230: (8)                   return True
2231: (4)               def __delitem__(self, i)->None:
2232: (8)                   if not self.is_pdf:
2233: (12)                      raise ValueError("is no PDF")
2234: (8)                   if type(i) is int:
2235: (12)                      return self.delete_page(i)
2236: (8)                   if type(i) in (list, tuple, range):
2237: (12)                      return self.delete_pages(i)
```

```
2238: (8)                   if type(i) is not slice:
2239: (12)                      raise ValueError("bad argument type")
2240: (8)                   pc = self.page_count
2241: (8)                   start = i.start if i.start else 0
2242: (8)                   stop = i.stop if i.stop else pc
2243: (8)                   step = i.step if i.step else 1
2244: (8)                   while start < 0:
2245: (12)                      start += pc
2246: (8)                   if start >= pc:
2247: (12)                      raise ValueError("bad page number(s)")
2248: (8)                   while stop < 0:
2249: (12)                      stop += pc
2250: (8)                   if stop > pc:
2251: (12)                      raise ValueError("bad page number(s)")
2252: (8)                   return self.delete_pages(range(start, stop, step))
2253: (4)               def __enter__(self):
2254: (8)                   return self
2255: (4)               def __exit__(self, *args):
2256: (8)                   self.close()
2257: (4)               @typing.overload
2258: (4)               def __getitem__(self, i: int = 0) -> Page:
2259: (8)                   ...
2260: (4)               if sys.version_info >= (3, 9):
2261: (8)                   @typing.overload
2262: (8)                   def __getitem__(self, i: slice) -> list[Page]:
2263: (12)                      ...
2264: (8)                   @typing.overload
2265: (8)                   def __getitem__(self, i: tuple[int, int]) -> Page:
2266: (12)                      ...
2267: (4)               def __getitem__(self, i=0):
2268: (8)                   if isinstance(i, slice):
2269: (12)                      return [self[j] for j in range(*i.indices(len(self)))]
2270: (8)                   assert isinstance(i, int) or (isinstance(i, tuple) and len(i) == 2 and
all(isinstance(x, int) for x in i)), \
2271: (16)                          f'Invalid item number: {i=}.'
2272: (8)                   if i not in self:
2273: (12)                      raise IndexError(f"page {i} not in document")
2274: (8)                   return self.load_page(i)
2275: (4)               def __init__(self, filename=None, stream=None, filetype=None, rect=None,
width=0, height=0, fontsize=11):
2276: (8)                   """Creates a document. Use 'open' as a synonym.
2277: (8)                   Notes:
2278: (12)                      Basic usages:
2279: (12)                      open() - new PDF document
2280: (12)                      open(filename) - string or pathlib.Path, must have supported
2281: (20)                              file extension.
2282: (12)                      open(type, buffer) - type: valid extension, buffer: bytes object.
2283: (12)                      open(stream=buffer, filetype=type) - keyword version of previous.
2284: (12)                      open(filename, fileype=type) - filename with unrecognized
extension.
2285: (12)                      rect, width, height, fontsize: layout reflowable document
2286: (12)                      on open (e.g. EPUB). Ignored if n/a.
2287: (8)                   """
2288: (8)                   global JM_mupdf_show_errors
2289: (8)                   JM_mupdf_show_errors_old = JM_mupdf_show_errors
2290: (8)                   JM_mupdf_show_errors = 0
2291: (8)                   try:
2292: (12)                      self.is_closed    = False
2293: (12)                      self.is_encrypted = False
2294: (12)                      self.is_encrypted = False
2295: (12)                      self.metadata    = None
2296: (12)                      self.FontInfos   = []
2297: (12)                      self.Graftmaps   = {}
2298: (12)                      self.ShownPages  = {}
2299: (12)                      self.InsertedImages  = {}
2300: (12)                      self._page_refs  = weakref.WeakValueDictionary()
2301: (12)                      if isinstance(filename, mupdf.PdfDocument):
2302: (16)                          pdf_document = filename
2303: (16)                          self.this = pdf_document
```

```
2304: (16)                          self.this_is_pdf = True
2305: (16)                          return
2306: (12)                      if not filename or type(filename) is str:
2307: (16)                          pass
2308: (12)                      elif hasattr(filename, "absolute"):
2309: (16)                          filename = str(filename)
2310: (12)                      elif hasattr(filename, "name"):
2311: (16)                          filename = filename.name
2312: (12)                      else:
2313: (16)                          raise TypeError(f"bad filename: {type(filename)=}
{filename=}.")
2314: (12)                      if stream is not None:
2315: (16)                          if type(stream) is bytes:
2316: (20)                              self.stream = stream
2317: (16)                          elif type(stream) is bytearray:
2318: (20)                              self.stream = bytes(stream)
2319: (16)                          elif type(stream) is io.BytesIO:
2320: (20)                              self.stream = stream.getvalue()
2321: (16)                          else:
2322: (20)                              raise TypeError(f"bad stream: {type(stream)=}.")
2323: (16)                          stream = self.stream
2324: (16)                          if not (filename or filetype):
2325: (20)                              filename = 'pdf'
2326: (12)                      else:
2327: (16)                          self.stream = None
2328: (12)                      if filename and self.stream is None:
2329: (16)                          from_file = True
2330: (16)                          self._name = filename
2331: (12)                      else:
2332: (16)                          from_file = False
2333: (16)                          self._name = ""
2334: (12)                      if from_file:
2335: (16)                          if not os.path.exists(filename):
2336: (20)                              msg = f"no such file: '{filename}'"
2337: (20)                              raise FileNotFoundError(msg)
2338: (16)                          elif not os.path.isfile(filename):
2339: (20)                              msg = f"'{filename}' is no file"
2340: (20)                              raise FileDataError(msg)
2341: (12)                      if from_file and os.path.getsize(filename) == 0:
2342: (16)                          raise EmptyFileError(f'Cannot open empty file: {filename=}.')
2343: (12)                      if type(self.stream) is bytes and len(self.stream) == 0:
2344: (16)                          raise EmptyFileError(f'Cannot open empty stream.')
2345: (12)                      w = width
2346: (12)                      h = height
2347: (12)                      r = JM_rect_from_py(rect)
2348: (12)                      if not mupdf.fz_is_infinite_rect(r):
2349: (16)                          w = r.x1 - r.x0
2350: (16)                          h = r.y1 - r.y0
2351: (12)                      if stream:  # stream given, **MUST** be bytes!
2352: (16)                          assert isinstance(stream, bytes)
2353: (16)                          c = stream
2354: (16)                          if mupdf_cppyy:
2355: (20)                              buffer_ = mupdf.fz_new_buffer_from_copied_data( c)
2356: (20)                              data = mupdf.fz_open_buffer( buffer_)
2357: (16)                          else:
2358: (20)                              data = mupdf.fz_open_memory(mupdf.python_buffer_data(c),
len(c))
2359: (16)                          magic = filename
2360: (16)                          if not magic:
2361: (20)                              magic = filetype
2362: (16)                          doc = mupdf.fz_open_document_with_stream(magic, data)
2363: (12)                      else:
2364: (16)                          if filename:
2365: (20)                              if not filetype:
2366: (24)                                  try:
2367: (28)                                      doc = mupdf.fz_open_document(filename)
2368: (24)                                  except Exception as e:
2369: (28)                                      if g_exceptions_verbose > 1:    exception_info()
2370: (28)                                      raise FileDataError(f'Failed to open file
```

```
{filename!r}.') from e
2371: (20)                                else:
2372: (24)                                    handler = mupdf.ll_fz_recognize_document(filetype)
2373: (24)                                    if handler:
2374: (28)                                        if handler.open:
2375: (32)                                            try:
2376: (36)                                                if mupdf_version_tuple >= (1, 24):
2377: (40)                                                    stream = mupdf.FzStream(filename)
2378: (40)                                                    accel = mupdf.FzStream()
2379: (40)                                                    archive = mupdf.FzArchive(None)
2380: (40)                                                    if mupdf_version_tuple >= (1, 24, 8):
2381: (44)                                                        doc =
mupdf.ll_fz_document_handler_open(
2382: (52)                                                                 handler,
2383: (52)                                                                 stream.m_internal,
2384: (52)                                                                 accel.m_internal,
2385: (52)                                                                 archive.m_internal,
2386: (52)                                                                 None,   # recognize_state
2387: (52)                                                                 )
2388: (40)                                                    else:
2389: (44)                                                        doc =
mupdf.ll_fz_document_open_fn_call(
2390: (52)                                                                 handler.open,
2391: (52)                                                                 stream.m_internal,
2392: (52)                                                                 accel.m_internal,
2393: (52)                                                                 archive.m_internal,
2394: (52)                                                                 )
2395: (36)                                                else:
2396: (40)                                                    doc =
mupdf.ll_fz_document_open_fn_call( handler.open, filename)
2397: (32)                                            except Exception as e:
2398: (36)                                                if g_exceptions_verbose > 1:
exception_info()
2399: (36)                                                raise FileDataError(f'Failed to open file
{filename!r} as type {filetype!r}.') from e
2400: (32)                                            doc = mupdf.FzDocument( doc)
2401: (28)                                        else:
2402: (32)                                            if mupdf_version_tuple < (1, 24):
2403: (36)                                                if handler.open_with_stream:
2404: (40)                                                    data = mupdf.fz_open_file( filename)
2405: (40)                                                    doc =
mupdf.fz_document_open_with_stream_fn_call( handler.open_with_stream, data)
2406: (32)                                            else:
2407: (36)                                                assert 0
2408: (24)                                    else:
2409: (28)                                        raise ValueError( MSG_BAD_FILETYPE)
2410: (16)                            else:
2411: (20)                                pdf = mupdf.PdfDocument()
2412: (20)                                doc = mupdf.FzDocument(pdf)
2413: (12)                        if w > 0 and h > 0:
2414: (16)                            mupdf.fz_layout_document(doc, w, h, fontsize)
2415: (12)                        elif mupdf.fz_is_document_reflowable(doc):
2416: (16)                            mupdf.fz_layout_document(doc, 400, 600, 11)
2417: (12)                        this = doc
2418: (12)                        self.this = this
2419: (12)                        self.thisown = True
2420: (12)                        if self.thisown:
2421: (16)                            self._graft_id = TOOLS.gen_id()
2422: (16)                            if self.needs_pass:
2423: (20)                                self.is_encrypted = True
2424: (16)                            else: # we won't init until doc is decrypted
2425: (20)                                self.init_doc()
2426: (16)                            if filename and filename.lower().endswith("svg") or filetype
and "svg" in filetype.lower():
2427: (20)                                try:
2428: (24)                                    _ = self.convert_to_pdf()  # this seems to always work
2429: (20)                                except Exception as e:
2430: (24)                                    if g_exceptions_verbose > 1:    exception_info()
2431: (24)                                    raise FileDataError("cannot open broken document")
```

```
from e
2432: (12)                      if g_use_extra:
2433: (16)                          self.this_is_pdf = isinstance( self.this, mupdf.PdfDocument)
2434: (16)                          if self.this_is_pdf:
2435: (20)                              self.page_count2 = extra.page_count_pdf
2436: (16)                          else:
2437: (20)                              self.page_count2 = extra.page_count_fz
2438: (8)                   finally:
2439: (12)                      JM_mupdf_show_errors = JM_mupdf_show_errors_old
2440: (4)               def __len__(self) -> int:
2441: (8)                   return self.page_count
2442: (4)               def __repr__(self) -> str:
2443: (8)                   m = "closed " if self.is_closed else ""
2444: (8)                   if self.stream is None:
2445: (12)                      if self.name == "":
2446: (16)                          return m + "Document(<new PDF, doc# %i>)" % self._graft_id
2447: (12)                      return m + "Document('%s')" % (self.name,)
2448: (8)                   return m + "Document('%s', <memory, doc# %i>)" % (self.name,
self._graft_id)
2449: (4)               def _addFormFont(self, name, font):
2450: (8)                   """Add new form font."""
2451: (8)                   if self.is_closed or self.is_encrypted:
2452: (12)                      raise ValueError("document closed or encrypted")
2453: (8)                   pdf = _as_pdf_document(self, required=0)
2454: (8)                   if not pdf.m_internal:
2455: (12)                      return
2456: (8)                   fonts = mupdf.pdf_dict_getl(
2457: (16)                          mupdf.pdf_trailer( pdf),
2458: (16)                          PDF_NAME('Root'),
2459: (16)                          PDF_NAME('AcroForm'),
2460: (16)                          PDF_NAME('DR'),
2461: (16)                          PDF_NAME('Font'),
2462: (16)                          )
2463: (8)                   if not fonts.m_internal or not mupdf.pdf_is_dict( fonts):
2464: (12)                      raise RuntimeError( "PDF has no form fonts yet")
2465: (8)                   k = mupdf.pdf_new_name( name)
2466: (8)                   v = JM_pdf_obj_from_str( pdf, font)
2467: (8)                   mupdf.pdf_dict_put( fonts, k, v)
2468: (4)               def _delToC(self):
2469: (8)                   """Delete the TOC."""
2470: (8)                   if self.is_closed or self.is_encrypted:
2471: (12)                      raise ValueError("document closed or encrypted")
2472: (8)                   xrefs = []  # create Python list
2473: (8)                   pdf = _as_pdf_document(self, required=0)
2474: (8)                   if not pdf.m_internal:
2475: (12)                      return xrefs    # not a pdf
2476: (8)                   root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
2477: (8)                   olroot = mupdf.pdf_dict_get(root, PDF_NAME('Outlines'))
2478: (8)                   if not olroot.m_internal:
2479: (12)                      return xrefs    # no outlines or some problem
2480: (8)                   first = mupdf.pdf_dict_get(olroot, PDF_NAME('First'))  # first outline
2481: (8)                   xrefs = JM_outline_xrefs(first, xrefs)
2482: (8)                   xref_count = len(xrefs)
2483: (8)                   olroot_xref = mupdf.pdf_to_num(olroot) # delete OL root
2484: (8)                   mupdf.pdf_delete_object(pdf, olroot_xref)  # delete OL root
2485: (8)                   mupdf.pdf_dict_del(root, PDF_NAME('Outlines')) # delete OL root
2486: (8)                   for i in range(xref_count):
2487: (12)                      _, xref = JM_INT_ITEM(xrefs, i)
2488: (12)                      mupdf.pdf_delete_object(pdf, xref) # delete outline item
2489: (8)                   xrefs.append(olroot_xref)
2490: (8)                   val = xrefs
2491: (8)                   self.init_doc()
2492: (8)                   return val
2493: (4)               def _delete_page(self, pno):
2494: (8)                   pdf = _as_pdf_document(self)
2495: (8)                   mupdf.pdf_delete_page( pdf, pno)
2496: (8)                   if pdf.m_internal.rev_page_map:
2497: (12)                      mupdf.ll_pdf_drop_page_tree( pdf.m_internal)
2498: (4)               def _deleteObject(self, xref):
```

```
2499: (8)                    """Delete object."""
2500: (8)                    pdf = _as_pdf_document(self)
2501: (8)                    if not _INRANGE(xref, 1, mupdf.pdf_xref_len(pdf)-1):
2502: (12)                       raise ValueError( MSG_BAD_XREF)
2503: (8)                    mupdf.pdf_delete_object(pdf, xref)
2504: (4)                def _embeddedFileGet(self, idx):
2505: (8)                    pdf = _as_pdf_document(self)
2506: (8)                    names = mupdf.pdf_dict_getl(
2507: (16)                           mupdf.pdf_trailer(pdf),
2508: (16)                           PDF_NAME('Root'),
2509: (16)                           PDF_NAME('Names'),
2510: (16)                           PDF_NAME('EmbeddedFiles'),
2511: (16)                           PDF_NAME('Names'),
2512: (16)                           )
2513: (8)                    entry = mupdf.pdf_array_get(names, 2*idx+1)
2514: (8)                    filespec = mupdf.pdf_dict_getl(entry, PDF_NAME('EF'), PDF_NAME('F'))
2515: (8)                    buf = mupdf.pdf_load_stream(filespec)
2516: (8)                    cont = JM_BinFromBuffer(buf)
2517: (8)                    return cont
2518: (4)                def _embeddedFileIndex(self, item: typing.Union[int, str]) -> int:
2519: (8)                    filenames = self.embfile_names()
2520: (8)                    msg = "'%s' not in EmbeddedFiles array." % str(item)
2521: (8)                    if item in filenames:
2522: (12)                       idx = filenames.index(item)
2523: (8)                    elif item in range(len(filenames)):
2524: (12)                       idx = item
2525: (8)                    else:
2526: (12)                       raise ValueError(msg)
2527: (8)                    return idx
2528: (4)                def _embfile_add(self, name, buffer_, filename=None, ufilename=None,
desc=None):
2529: (8)                    pdf = _as_pdf_document(self)
2530: (8)                    data = JM_BufferFromBytes(buffer_)
2531: (8)                    if not data.m_internal:
2532: (12)                       raise TypeError( MSG_BAD_BUFFER)
2533: (8)                    names = mupdf.pdf_dict_getl(
2534: (16)                           mupdf.pdf_trailer(pdf),
2535: (16)                           PDF_NAME('Root'),
2536: (16)                           PDF_NAME('Names'),
2537: (16)                           PDF_NAME('EmbeddedFiles'),
2538: (16)                           PDF_NAME('Names'),
2539: (16)                           )
2540: (8)                    if not mupdf.pdf_is_array(names):
2541: (12)                       root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf),
PDF_NAME('Root'))
2542: (12)                       names = mupdf.pdf_new_array(pdf, 6)    # an even number!
2543: (12)                       mupdf.pdf_dict_putl(
2544: (20)                               root,
2545: (20)                               names,
2546: (20)                               PDF_NAME('Names'),
2547: (20)                               PDF_NAME('EmbeddedFiles'),
2548: (20)                               PDF_NAME('Names'),
2549: (20)                               )
2550: (8)                    fileentry = JM_embed_file(pdf, data, filename, ufilename, desc, 1)
2551: (8)                    xref = mupdf.pdf_to_num(
2552: (16)                           mupdf.pdf_dict_getl(fileentry, PDF_NAME('EF'), PDF_NAME('F'))
2553: (16)                           )
2554: (8)                    mupdf.pdf_array_push(names, mupdf.pdf_new_text_string(name))
2555: (8)                    mupdf.pdf_array_push(names, fileentry)
2556: (8)                    return xref
2557: (4)                def _embfile_del(self, idx):
2558: (8)                    pdf = _as_pdf_document(self)
2559: (8)                    names = mupdf.pdf_dict_getl(
2560: (16)                           mupdf.pdf_trailer(pdf),
2561: (16)                           PDF_NAME('Root'),
2562: (16)                           PDF_NAME('Names'),
2563: (16)                           PDF_NAME('EmbeddedFiles'),
2564: (16)                           PDF_NAME('Names'),
2565: (16)                           )
```

```
2566: (8)                    mupdf.pdf_array_delete(names, idx + 1)
2567: (8)                    mupdf.pdf_array_delete(names, idx)
2568: (4)                def _embfile_info(self, idx, infodict):
2569: (8)                    pdf = _as_pdf_document(self)
2570: (8)                    xref = 0
2571: (8)                    ci_xref=0
2572: (8)                    trailer = mupdf.pdf_trailer(pdf)
2573: (8)                    names = mupdf.pdf_dict_getl(
2574: (16)                           trailer,
2575: (16)                           PDF_NAME('Root'),
2576: (16)                           PDF_NAME('Names'),
2577: (16)                           PDF_NAME('EmbeddedFiles'),
2578: (16)                           PDF_NAME('Names'),
2579: (16)                           )
2580: (8)                    o = mupdf.pdf_array_get(names, 2*idx+1)
2581: (8)                    ci = mupdf.pdf_dict_get(o, PDF_NAME('CI'))
2582: (8)                    if ci.m_internal:
2583: (12)                       ci_xref = mupdf.pdf_to_num(ci)
2584: (8)                    infodict["collection"] = ci_xref
2585: (8)                    name = mupdf.pdf_to_text_string(mupdf.pdf_dict_get(o, PDF_NAME('F')))
2586: (8)                    infodict[dictkey_filename] = JM_EscapeStrFromStr(name)
2587: (8)                    name = mupdf.pdf_to_text_string(mupdf.pdf_dict_get(o, PDF_NAME('UF')))
2588: (8)                    infodict[dictkey_ufilename] = JM_EscapeStrFromStr(name)
2589: (8)                    name = mupdf.pdf_to_text_string(mupdf.pdf_dict_get(o,
PDF_NAME('Desc')))
2590: (8)                    infodict[dictkey_desc] = JM_UnicodeFromStr(name)
2591: (8)                    len_ = -1
2592: (8)                    DL = -1
2593: (8)                    fileentry = mupdf.pdf_dict_getl(o, PDF_NAME('EF'), PDF_NAME('F'))
2594: (8)                    xref = mupdf.pdf_to_num(fileentry)
2595: (8)                    o = mupdf.pdf_dict_get(fileentry, PDF_NAME('Length'))
2596: (8)                    if o.m_internal:
2597: (12)                       len_ = mupdf.pdf_to_int(o)
2598: (8)                    o = mupdf.pdf_dict_get(fileentry, PDF_NAME('DL'))
2599: (8)                    if o.m_internal:
2600: (12)                       DL = mupdf.pdf_to_int(o)
2601: (8)                    else:
2602: (12)                       o = mupdf.pdf_dict_getl(fileentry, PDF_NAME('Params'),
PDF_NAME('Size'))
2603: (12)                       if o.m_internal:
2604: (16)                           DL = mupdf.pdf_to_int(o)
2605: (8)                    infodict[dictkey_size] = DL
2606: (8)                    infodict[dictkey_length] = len_
2607: (8)                    return xref
2608: (4)                def _embfile_names(self, namelist):
2609: (8)                    """Get list of embedded file names."""
2610: (8)                    pdf = _as_pdf_document(self)
2611: (8)                    names = mupdf.pdf_dict_getl(
2612: (16)                           mupdf.pdf_trailer(pdf),
2613: (16)                           PDF_NAME('Root'),
2614: (16)                           PDF_NAME('Names'),
2615: (16)                           PDF_NAME('EmbeddedFiles'),
2616: (16)                           PDF_NAME('Names'),
2617: (16)                           )
2618: (8)                    if mupdf.pdf_is_array(names):
2619: (12)                       n = mupdf.pdf_array_len(names)
2620: (12)                       for i in range(0, n, 2):
2621: (16)                           val = JM_EscapeStrFromStr(
2622: (24)                                   mupdf.pdf_to_text_string(
2623: (28)                                       mupdf.pdf_array_get(names, i)
2624: (28)                                       )
2625: (24)                                   )
2626: (16)                           namelist.append(val)
2627: (4)                def _embfile_upd(self, idx, buffer_=None, filename=None, ufilename=None,
desc=None):
2628: (8)                    pdf = _as_pdf_document(self)
2629: (8)                    xref = 0
2630: (8)                    names = mupdf.pdf_dict_getl(
2631: (16)                           mupdf.pdf_trailer(pdf),
```

```
2632: (16)                          PDF_NAME('Root'),
2633: (16)                          PDF_NAME('Names'),
2634: (16)                          PDF_NAME('EmbeddedFiles'),
2635: (16)                          PDF_NAME('Names'),
2636: (16)                          )
2637: (8)                   entry = mupdf.pdf_array_get(names, 2*idx+1)
2638: (8)                   filespec = mupdf.pdf_dict_getl(entry, PDF_NAME('EF'), PDF_NAME('F'))
2639: (8)                   if not filespec.m_internal:
2640: (12)                      RAISEPY( "bad PDF: no /EF object", JM_Exc_FileDataError)
2641: (8)                   res = JM_BufferFromBytes(buffer_)
2642: (8)                   if buffer_ and buffer_.m_internal and not res.m_internal:
2643: (12)                      raise TypeError( MSG_BAD_BUFFER)
2644: (8)                   if res.m_internal and buffer_ and buffer_.m_internal:
2645: (12)                      JM_update_stream(pdf, filespec, res, 1)
2646: (12)                      len, _ = mupdf.fz_buffer_storage(res)
2647: (12)                      l = mupdf.pdf_new_int(len)
2648: (12)                      mupdf.pdf_dict_put(filespec, PDF_NAME('DL'), l)
2649: (12)                      mupdf.pdf_dict_putl(filespec, l, PDF_NAME('Params'),
PDF_NAME('Size'))
2650: (8)                   xref = mupdf.pdf_to_num(filespec)
2651: (8)                   if filename:
2652: (12)                      mupdf.pdf_dict_put_text_string(entry, PDF_NAME('F'), filename)
2653: (8)                   if ufilename:
2654: (12)                      mupdf.pdf_dict_put_text_string(entry, PDF_NAME('UF'), ufilename)
2655: (8)                   if desc:
2656: (12)                      mupdf.pdf_dict_put_text_string(entry, PDF_NAME('Desc'), desc)
2657: (8)                   return xref
2658: (4)               def _extend_toc_items(self, items):
2659: (8)                   """Add color info to all items of an extended TOC list."""
2660: (8)                   if self.is_closed:
2661: (12)                      raise ValueError("document closed")
2662: (8)                   if g_use_extra:
2663: (12)                      return extra.Document_extend_toc_items( self.this, items)
2664: (8)                   pdf = _as_pdf_document(self)
2665: (8)                   zoom = "zoom"
2666: (8)                   bold = "bold"
2667: (8)                   italic = "italic"
2668: (8)                   collapse = "collapse"
2669: (8)                   root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
2670: (8)                   if not root.m_internal:
2671: (12)                      return
2672: (8)                   olroot = mupdf.pdf_dict_get(root, PDF_NAME('Outlines'))
2673: (8)                   if not olroot.m_internal:
2674: (12)                      return
2675: (8)                   first = mupdf.pdf_dict_get(olroot, PDF_NAME('First'))
2676: (8)                   if not first.m_internal:
2677: (12)                      return
2678: (8)                   xrefs = []
2679: (8)                   xrefs = JM_outline_xrefs(first, xrefs)
2680: (8)                   n = len(xrefs)
2681: (8)                   m = len(items)
2682: (8)                   if not n:
2683: (12)                      return
2684: (8)                   if n != m:
2685: (12)                      raise IndexError( "internal error finding outline xrefs")
2686: (8)                   for i in range(n):
2687: (12)                      xref = int(xrefs[i])
2688: (12)                      item = items[i]
2689: (12)                      itemdict = item[3]
2690: (12)                      if not isinstance(itemdict, dict):
2691: (16)                          raise ValueError( "need non-simple TOC format")
2692: (12)                      itemdict[dictkey_xref] = xrefs[i]
2693: (12)                      bm = mupdf.pdf_load_object(pdf, xref)
2694: (12)                      flags = mupdf.pdf_to_int( mupdf.pdf_dict_get(bm, PDF_NAME('F')))
2695: (12)                      if flags == 1:
2696: (16)                          itemdict[italic] = True
2697: (12)                      elif flags == 2:
2698: (16)                          itemdict[bold] = True
2699: (12)                      elif flags == 3:
```

```
2700: (16)                          itemdict[italic] = True
2701: (16)                          itemdict[bold] = True
2702: (12)                      count = mupdf.pdf_to_int( mupdf.pdf_dict_get(bm,
PDF_NAME('Count')))
2703: (12)                      if count < 0:
2704: (16)                          itemdict[collapse] = True
2705: (12)                      elif count > 0:
2706: (16)                          itemdict[collapse] = False
2707: (12)                      col = mupdf.pdf_dict_get(bm, PDF_NAME('C'))
2708: (12)                      if mupdf.pdf_is_array(col) and mupdf.pdf_array_len(col) == 3:
2709: (16)                          color = (
2710: (24)                                  mupdf.pdf_to_real(mupdf.pdf_array_get(col, 0)),
2711: (24)                                  mupdf.pdf_to_real(mupdf.pdf_array_get(col, 1)),
2712: (24)                                  mupdf.pdf_to_real(mupdf.pdf_array_get(col, 2)),
2713: (24)                                  )
2714: (16)                          itemdict[dictkey_color] = color
2715: (12)                      z=0
2716: (12)                      obj = mupdf.pdf_dict_get(bm, PDF_NAME('Dest'))
2717: (12)                      if not obj.m_internal or not mupdf.pdf_is_array(obj):
2718: (16)                          obj = mupdf.pdf_dict_getl(bm, PDF_NAME('A'), PDF_NAME('D'))
2719: (12)                      if mupdf.pdf_is_array(obj) and mupdf.pdf_array_len(obj) == 5:
2720: (16)                          z = mupdf.pdf_to_real(mupdf.pdf_array_get(obj, 4))
2721: (12)                      itemdict[zoom] = float(z)
2722: (12)                      item[3] = itemdict
2723: (12)                      items[i] = item
2724: (4)               def _forget_page(self, page: Page):
2725: (8)                   """Remove a page from document page dict."""
2726: (8)                   pid = id(page)
2727: (8)                   if pid in self._page_refs:
2728: (12)                      del self._page_refs[pid]
2729: (4)               def _get_char_widths(self, xref: int, bfname: str, ext: str, ordering:
int, limit: int, idx: int = 0):
2730: (8)                   pdf = _as_pdf_document(self)
2731: (8)                   mylimit = limit
2732: (8)                   if mylimit < 256:
2733: (12)                      mylimit = 256
2734: (8)                   if ordering >= 0:
2735: (12)                      data, size, index = mupdf.fz_lookup_cjk_font(ordering)
2736: (12)                      font = mupdf.fz_new_font_from_memory(None, data, size, index, 0)
2737: (8)                   else:
2738: (12)                      data, size = mupdf.fz_lookup_base14_font(bfname)
2739: (12)                      if data:
2740: (16)                          font = mupdf.fz_new_font_from_memory(bfname, data, size, 0, 0)
2741: (12)                      else:
2742: (16)                          buf = JM_get_fontbuffer(pdf, xref)
2743: (16)                          if not buf.m_internal:
2744: (20)                              raise Exception("font at xref %d is not supported" % xref)
2745: (16)                          font = mupdf.fz_new_font_from_buffer(None, buf, idx, 0)
2746: (8)                   wlist = []
2747: (8)                   for i in range(mylimit):
2748: (12)                      glyph = mupdf.fz_encode_character(font, i)
2749: (12)                      adv = mupdf.fz_advance_glyph(font, glyph, 0)
2750: (12)                      if ordering >= 0:
2751: (16)                          glyph = i
2752: (12)                      if glyph > 0:
2753: (16)                          wlist.append( (glyph, adv))
2754: (12)                      else:
2755: (16)                          wlist.append( (glyph, 0.0))
2756: (8)                   return wlist
2757: (4)               def _get_page_labels(self):
2758: (8)                   pdf = _as_pdf_document(self)
2759: (8)                   rc = []
2760: (8)                   pagelabels = mupdf.pdf_new_name("PageLabels")
2761: (8)                   obj = mupdf.pdf_dict_getl( mupdf.pdf_trailer(pdf), PDF_NAME('Root'),
pagelabels)
2762: (8)                   if not obj.m_internal:
2763: (12)                      return rc
2764: (8)                   nums = mupdf.pdf_resolve_indirect( mupdf.pdf_dict_get( obj,
PDF_NAME('Nums')))
```

```
2765: (8)                    if nums.m_internal:
2766: (12)                       JM_get_page_labels(rc, nums)
2767: (12)                       return rc
2768: (8)                    nums = mupdf.pdf_resolve_indirect( mupdf.pdf_dict_getl(obj,
PDF_NAME('Kids'), PDF_NAME('Nums')))
2769: (8)                    if nums.m_internal:
2770: (12)                       JM_get_page_labels(rc, nums)
2771: (12)                       return rc
2772: (8)                    kids = mupdf.pdf_resolve_indirect( mupdf.pdf_dict_get( obj,
PDF_NAME('Kids')))
2773: (8)                    if not kids.m_internal or not mupdf.pdf_is_array(kids):
2774: (12)                       return rc
2775: (8)                    n = mupdf.pdf_array_len(kids)
2776: (8)                    for i in range(n):
2777: (12)                       nums = mupdf.pdf_resolve_indirect(
2778: (20)                               mupdf.pdf_dict_get( mupdf.pdf_array_get(kids, i)),
2779: (20)                               PDF_NAME('Nums'),
2780: (20)                               )
2781: (12)                       JM_get_page_labels(rc, nums)
2782: (8)                    return rc
2783: (4)                def _getMetadata(self, key):
2784: (8)                    """Get metadata."""
2785: (8)                    try:
2786: (12)                       return mupdf.fz_lookup_metadata2( self.this, key)
2787: (8)                    except Exception:
2788: (12)                       if g_exceptions_verbose > 2:    exception_info()
2789: (12)                       return ''
2790: (4)                def _getOLRootNumber(self):
2791: (8)                    """Get xref of Outline Root, create it if missing."""
2792: (8)                    if self.is_closed or self.is_encrypted:
2793: (12)                       raise ValueError("document closed or encrypted")
2794: (8)                    pdf = _as_pdf_document(self)
2795: (8)                    root = mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf), PDF_NAME('Root'))
2796: (8)                    olroot = mupdf.pdf_dict_get( root, PDF_NAME('Outlines'))
2797: (8)                    if not olroot.m_internal:
2798: (12)                       olroot = mupdf.pdf_new_dict( pdf, 4)
2799: (12)                       mupdf.pdf_dict_put( olroot, PDF_NAME('Type'),
PDF_NAME('Outlines'))
2800: (12)                       ind_obj = mupdf.pdf_add_object( pdf, olroot)
2801: (12)                       mupdf.pdf_dict_put( root, PDF_NAME('Outlines'), ind_obj)
2802: (12)                       olroot = mupdf.pdf_dict_get( root, PDF_NAME('Outlines'))
2803: (8)                    return mupdf.pdf_to_num( olroot)
2804: (4)                def _getPDFfileid(self):
2805: (8)                    """Get PDF file id."""
2806: (8)                    pdf = _as_pdf_document(self, required=0)
2807: (8)                    if not pdf.m_internal:
2808: (12)                       return
2809: (8)                    idlist = []
2810: (8)                    identity = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('ID'))
2811: (8)                    if identity.m_internal:
2812: (12)                       n = mupdf.pdf_array_len(identity)
2813: (12)                       for i in range(n):
2814: (16)                           o = mupdf.pdf_array_get(identity, i)
2815: (16)                           text = mupdf.pdf_to_text_string(o)
2816: (16)                           hex_ = binascii.hexlify(text)
2817: (16)                           idlist.append(hex_)
2818: (8)                    return idlist
2819: (4)                def _getPageInfo(self, pno, what):
2820: (8)                    """List fonts, images, XObjects used on a page."""
2821: (8)                    if self.is_closed or self.is_encrypted:
2822: (12)                       raise ValueError("document closed or encrypted")
2823: (8)                    doc = self.this
2824: (8)                    pageCount = mupdf.pdf_count_pages(doc) if isinstance(doc,
mupdf.PdfDocument) else mupdf.fz_count_pages(doc)
2825: (8)                    n = pno  # pno < 0 is allowed
2826: (8)                    while n < 0:
2827: (12)                       n += pageCount  # make it non-negative
2828: (8)                    if n >= pageCount:
2829: (12)                       raise ValueError( MSG_BAD_PAGENO)
```

```
2830: (8)                   pdf = _as_pdf_document(self)
2831: (8)                   pageref = mupdf.pdf_lookup_page_obj(pdf, n)
2832: (8)                   rsrc = mupdf.pdf_dict_get_inheritable(pageref,
mupdf.PDF_ENUM_NAME_Resources)
2833: (8)                   liste = []
2834: (8)                   tracer = []
2835: (8)                   if rsrc.m_internal:
2836: (12)                      JM_scan_resources(pdf, rsrc, liste, what, 0, tracer)
2837: (8)                   return liste
2838: (4)               def _insert_font(self, fontfile=None, fontbuffer=None):
2839: (8)                   '''
2840: (8)                   Utility: insert font from file or binary.
2841: (8)                   '''
2842: (8)                   pdf = _as_pdf_document(self)
2843: (8)                   if not fontfile and not fontbuffer:
2844: (12)                      raise ValueError( MSG_FILE_OR_BUFFER)
2845: (8)                   value = JM_insert_font(pdf, None, fontfile, fontbuffer, 0, 0, 0, 0, 0,
-1)
2846: (8)                   return value
2847: (4)               def _loadOutline(self):
2848: (8)                   """Load first outline."""
2849: (8)                   doc = self.this
2850: (8)                   assert isinstance( doc, mupdf.FzDocument)
2851: (8)                   try:
2852: (12)                      ol = mupdf.fz_load_outline( doc)
2853: (8)                   except Exception:
2854: (12)                      if g_exceptions_verbose > 1:    exception_info()
2855: (12)                      return
2856: (8)                   return Outline( ol)
2857: (4)               def _make_page_map(self):
2858: (8)                   """Make an array page number -> page object."""
2859: (8)                   if self.is_closed:
2860: (12)                      raise ValueError("document closed")
2861: (8)                   assert 0, f'_make_page_map() is no-op'
2862: (4)               def _move_copy_page(self, pno, nb, before, copy):
2863: (8)                   """Move or copy a PDF page reference."""
2864: (8)                   pdf = _as_pdf_document(self)
2865: (8)                   same = 0
2866: (8)                   page1, parent1, i1 = pdf_lookup_page_loc( pdf, pno)
2867: (8)                   kids1 = mupdf.pdf_dict_get( parent1, PDF_NAME('Kids'))
2868: (8)                   page2, parent2, i2 = pdf_lookup_page_loc( pdf, nb)
2869: (8)                   kids2 = mupdf.pdf_dict_get( parent2, PDF_NAME('Kids'))
2870: (8)                   if before:  # calc index of source page in target /Kids
2871: (12)                      pos = i2
2872: (8)                   else:
2873: (12)                      pos = i2 + 1
2874: (8)                   same = mupdf.pdf_objcmp( kids1, kids2)
2875: (8)                   if not copy and same != 0:  # update parent in page object
2876: (12)                      mupdf.pdf_dict_put( page1, PDF_NAME('Parent'), parent2)
2877: (8)                   mupdf.pdf_array_insert( kids2, page1, pos)
2878: (8)                   if same != 0:   # different /Kids arrays ----------------------
2879: (12)                      parent = parent2
2880: (12)                      while parent.m_internal:    # increase /Count objects in parents
2881: (16)                          count = mupdf.pdf_dict_get_int( parent, PDF_NAME('Count'))
2882: (16)                          mupdf.pdf_dict_put_int( parent, PDF_NAME('Count'), count + 1)
2883: (16)                          parent = mupdf.pdf_dict_get( parent, PDF_NAME('Parent'))
2884: (12)                      if not copy:    # delete original item
2885: (16)                          mupdf.pdf_array_delete( kids1, i1)
2886: (16)                          parent = parent1
2887: (16)                          while parent.m_internal:    # decrease /Count objects in
parents
2888: (20)                              count = mupdf.pdf_dict_get_int( parent, PDF_NAME('Count'))
2889: (20)                              mupdf.pdf_dict_put_int( parent, PDF_NAME('Count'), count -
1)
2890: (20)                              parent = mupdf.pdf_dict_get( parent, PDF_NAME('Parent'))
2891: (8)                   else:   # same /Kids array
2892: (12)                      if copy:    # source page is copied
2893: (16)                          parent = parent2
2894: (16)                          while parent.m_internal:    # increase /Count object in
```

```
parents
2895: (20)                                count = mupdf.pdf_dict_get_int( parent, PDF_NAME('Count'))
2896: (20)                                mupdf.pdf_dict_put_int( parent, PDF_NAME('Count'), count +
1)
2897: (20)                                parent = mupdf.pdf_dict_get( parent, PDF_NAME('Parent'))
2898: (12)                            else:
2899: (16)                                if i1 < pos:
2900: (20)                                    mupdf.pdf_array_delete( kids1, i1)
2901: (16)                                else:
2902: (20)                                    mupdf.pdf_array_delete( kids1, i1 + 1)
2903: (8)                         if pdf.m_internal.rev_page_map: # page map no longer valid: drop it
2904: (12)                            mupdf.ll_pdf_drop_page_tree( pdf.m_internal)
2905: (8)                         self._reset_page_refs()
2906: (4)                     def _newPage(self, pno=-1, width=595, height=842):
2907: (8)                         """Make a new PDF page."""
2908: (8)                         if self.is_closed or self.is_encrypted:
2909: (12)                            raise ValueError("document closed or encrypted")
2910: (8)                         if g_use_extra:
2911: (12)                            extra._newPage( self.this, pno, width, height)
2912: (8)                         else:
2913: (12)                            pdf = _as_pdf_document(self)
2914: (12)                            mediabox = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)
2915: (12)                            mediabox.x1 = width
2916: (12)                            mediabox.y1 = height
2917: (12)                            contents = mupdf.FzBuffer()
2918: (12)                            if pno < -1:
2919: (16)                                raise ValueError( MSG_BAD_PAGENO)
2920: (12)                            resources = mupdf.pdf_add_new_dict(pdf, 1)
2921: (12)                            page_obj = mupdf.pdf_add_page( pdf, mediabox, 0, resources,
contents)
2922: (12)                            mupdf.pdf_insert_page( pdf, pno, page_obj)
2923: (8)                         self._reset_page_refs()
2924: (8)                         return self[pno]
2925: (4)                     def _remove_links_to(self, numbers):
2926: (8)                         pdf = _as_pdf_document(self)
2927: (8)                         _remove_dest_range(pdf, numbers)
2928: (4)                     def _remove_toc_item(self, xref):
2929: (8)                         pdf = _as_pdf_document(self)
2930: (8)                         item = mupdf.pdf_new_indirect(pdf, xref, 0)
2931: (8)                         mupdf.pdf_dict_del( item, PDF_NAME('Dest'))
2932: (8)                         mupdf.pdf_dict_del( item, PDF_NAME('A'))
2933: (8)                         color = mupdf.pdf_new_array( pdf, 3)
2934: (8)                         for i in range(3):
2935: (12)                            mupdf.pdf_array_push_real( color, 0.8)
2936: (8)                         mupdf.pdf_dict_put( item, PDF_NAME('C'), color)
2937: (4)                     def _reset_page_refs(self):
2938: (8)                         """Invalidate all pages in document dictionary."""
2939: (8)                         if getattr(self, "is_closed", True):
2940: (12)                            return
2941: (8)                         pages = [p for p in self._page_refs.values()]
2942: (8)                         for page in pages:
2943: (12)                            if page:
2944: (16)                                page._erase()
2945: (16)                                page = None
2946: (8)                         self._page_refs.clear()
2947: (4)                     def _set_page_labels(self, labels):
2948: (8)                         pdf = _as_pdf_document(self)
2949: (8)                         pagelabels = mupdf.pdf_new_name("PageLabels")
2950: (8)                         root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
2951: (8)                         mupdf.pdf_dict_del(root, pagelabels)
2952: (8)                         mupdf.pdf_dict_putl(root, mupdf.pdf_new_array(pdf, 0), pagelabels,
PDF_NAME('Nums'))
2953: (8)                         xref = self.pdf_catalog()
2954: (8)                         text = self.xref_object(xref, compressed=True)
2955: (8)                         text = text.replace("/Nums[]", "/Nums[%s]" % labels)
2956: (8)                         self.update_object(xref, text)
2957: (4)                     def _update_toc_item(self, xref, action=None, title=None, flags=0,
collapse=None, color=None):
2958: (8)                         '''
```

```
2959: (8)                    "update" bookmark by letting it point to nowhere
2960: (8)                    '''
2961: (8)                    pdf = _as_pdf_document(self)
2962: (8)                    item = mupdf.pdf_new_indirect( pdf, xref, 0)
2963: (8)                    if title:
2964: (12)                       mupdf.pdf_dict_put_text_string( item, PDF_NAME('Title'), title)
2965: (8)                    if action:
2966: (12)                       mupdf.pdf_dict_del( item, PDF_NAME('Dest'))
2967: (12)                       obj = JM_pdf_obj_from_str( pdf, action)
2968: (12)                       mupdf.pdf_dict_put( item, PDF_NAME('A'), obj)
2969: (8)                    mupdf.pdf_dict_put_int( item, PDF_NAME('F'), flags)
2970: (8)                    if color:
2971: (12)                       c = mupdf.pdf_new_array( pdf, 3)
2972: (12)                       for i in range(3):
2973: (16)                           f = color[i]
2974: (16)                           mupdf.pdf_array_push_real( c, f)
2975: (12)                       mupdf.pdf_dict_put( item, PDF_NAME('C'), c)
2976: (8)                    elif color is not None:
2977: (12)                       mupdf.pdf_dict_del( item, PDF_NAME('C'))
2978: (8)                    if collapse is not None:
2979: (12)                       if mupdf.pdf_dict_get( item, PDF_NAME('Count')).m_internal:
2980: (16)                           i = mupdf.pdf_dict_get_int( item, PDF_NAME('Count'))
2981: (16)                           if (i < 0 and collapse is False) or (i > 0 and collapse is
True):
2982: (20)                               i = i * (-1)
2983: (20)                               mupdf.pdf_dict_put_int( item, PDF_NAME('Count'), i)
2984: (4)                @property
2985: (4)                def FormFonts(self):
2986: (8)                    """Get list of field font resource names."""
2987: (8)                    pdf = _as_pdf_document(self, required=0)
2988: (8)                    if not pdf.m_internal:
2989: (12)                       return
2990: (8)                    fonts = mupdf.pdf_dict_getl(
2991: (16)                           mupdf.pdf_trailer(pdf),
2992: (16)                           PDF_NAME('Root'),
2993: (16)                           PDF_NAME('AcroForm'),
2994: (16)                           PDF_NAME('DR'),
2995: (16)                           PDF_NAME('Font'),
2996: (16)                           )
2997: (8)                    liste = list()
2998: (8)                    if fonts.m_internal and mupdf.pdf_is_dict(fonts):   # fonts exist
2999: (12)                       n = mupdf.pdf_dict_len(fonts)
3000: (12)                       for i in range(n):
3001: (16)                           f = mupdf.pdf_dict_get_key(fonts, i)
3002: (16)                           liste.append(JM_UnicodeFromStr(mupdf.pdf_to_name(f)))
3003: (8)                    return liste
3004: (4)                def add_layer(self, name, creator=None, on=None):
3005: (8)                    """Add a new OC layer."""
3006: (8)                    pdf = _as_pdf_document(self)
3007: (8)                    JM_add_layer_config( pdf, name, creator, on)
3008: (8)                    mupdf.ll_pdf_read_ocg( pdf.m_internal)
3009: (4)                def add_ocg(self, name, config=-1, on=1, intent=None, usage=None):
3010: (8)                    """Add new optional content group."""
3011: (8)                    xref = 0
3012: (8)                    pdf = _as_pdf_document(self)
3013: (8)                    ocg = mupdf.pdf_add_new_dict(pdf, 3)
3014: (8)                    mupdf.pdf_dict_put(ocg, PDF_NAME('Type'), PDF_NAME('OCG'))
3015: (8)                    mupdf.pdf_dict_put_text_string(ocg, PDF_NAME('Name'), name)
3016: (8)                    intents = mupdf.pdf_dict_put_array(ocg, PDF_NAME('Intent'), 2)
3017: (8)                    if not intent:
3018: (12)                       mupdf.pdf_array_push(intents, PDF_NAME('View'))
3019: (8)                    elif not isinstance(intent, str):
3020: (12)                       assert 0, f'fixme: intent is not a str. {type(intent)=} {type=}'
3021: (8)                    else:
3022: (12)                       mupdf.pdf_array_push(intents, mupdf.pdf_new_name(intent))
3023: (8)                    use_for = mupdf.pdf_dict_put_dict(ocg, PDF_NAME('Usage'), 3)
3024: (8)                    ci_name = mupdf.pdf_new_name("CreatorInfo")
3025: (8)                    cre_info = mupdf.pdf_dict_put_dict(use_for, ci_name, 2)
3026: (8)                    mupdf.pdf_dict_put_text_string(cre_info, PDF_NAME('Creator'),
```

```
"PyMuPDF")
3027: (8)                    if usage:
3028: (12)                       mupdf.pdf_dict_put_name(cre_info, PDF_NAME('Subtype'), usage)
3029: (8)                    else:
3030: (12)                       mupdf.pdf_dict_put_name(cre_info, PDF_NAME('Subtype'), "Artwork")
3031: (8)                    indocg = mupdf.pdf_add_object(pdf, ocg)
3032: (8)                    ocp = JM_ensure_ocproperties(pdf)
3033: (8)                    obj = mupdf.pdf_dict_get(ocp, PDF_NAME('OCGs'))
3034: (8)                    mupdf.pdf_array_push(obj, indocg)
3035: (8)                    if config > -1:
3036: (12)                       obj = mupdf.pdf_dict_get(ocp, PDF_NAME('Configs'))
3037: (12)                       if not mupdf.pdf_is_array(obj):
3038: (16)                           raise ValueError( MSG_BAD_OC_CONFIG)
3039: (12)                       cfg = mupdf.pdf_array_get(obj, config)
3040: (12)                       if not cfg.m_internal:
3041: (16)                           raise ValueError( MSG_BAD_OC_CONFIG)
3042: (8)                    else:
3043: (12)                       cfg = mupdf.pdf_dict_get(ocp, PDF_NAME('D'))
3044: (8)                    obj = mupdf.pdf_dict_get(cfg, PDF_NAME('Order'))
3045: (8)                    if not obj.m_internal:
3046: (12)                       obj = mupdf.pdf_dict_put_array(cfg, PDF_NAME('Order'), 1)
3047: (8)                    mupdf.pdf_array_push(obj, indocg)
3048: (8)                    if on:
3049: (12)                       obj = mupdf.pdf_dict_get(cfg, PDF_NAME('ON'))
3050: (12)                       if not obj.m_internal:
3051: (16)                           obj = mupdf.pdf_dict_put_array(cfg, PDF_NAME('ON'), 1)
3052: (8)                    else:
3053: (12)                       obj =mupdf.pdf_dict_get(cfg, PDF_NAME('OFF'))
3054: (12)                       if not obj.m_internal:
3055: (16)                           obj =mupdf.pdf_dict_put_array(cfg, PDF_NAME('OFF'), 1)
3056: (8)                    mupdf.pdf_array_push(obj, indocg)
3057: (8)                    mupdf.ll_pdf_read_ocg(pdf.m_internal)
3058: (8)                    xref = mupdf.pdf_to_num(indocg)
3059: (8)                    return xref
3060: (4)                def authenticate(self, password):
3061: (8)                    """Decrypt document."""
3062: (8)                    if self.is_closed:
3063: (12)                       raise ValueError("document closed")
3064: (8)                    val = mupdf.fz_authenticate_password(self.this, password)
3065: (8)                    if val:  # the doc is decrypted successfully and we init the outline
3066: (12)                       self.is_encrypted = False
3067: (12)                       self.is_encrypted = False
3068: (12)                       self.init_doc()
3069: (12)                       self.thisown = True
3070: (8)                    return val
3071: (4)                def can_save_incrementally(self):
3072: (8)                    """Check whether incremental saves are possible."""
3073: (8)                    pdf = _as_pdf_document(self, required=0)
3074: (8)                    if not pdf.m_internal:
3075: (12)                       return False
3076: (8)                    return mupdf.pdf_can_be_saved_incrementally(pdf)
3077: (4)                def bake(self, *, annots: bool = True, widgets: bool = True) -> None:
3078: (8)                    """Convert annotations or fields to permanent content.
3079: (8)                    Notes:
3080: (12)                       Converts annotations or widgets to permanent page content, like
3081: (12)                       text and vector graphics, as appropriate.
3082: (12)                       After execution, pages will still look the same, but no longer
3083: (12)                       have annotations, respectively no fields.
3084: (12)                       If widgets are selected the PDF will no longer be a Form PDF.
3085: (8)                    Args:
3086: (12)                       annots: convert annotations
3087: (12)                       widgets: convert form fields
3088: (8)                    """
3089: (8)                    pdf = _as_pdf_document(self)
3090: (8)                    mupdf.pdf_bake_document(pdf, int(annots), int(widgets))
3091: (4)                @property
3092: (4)                def chapter_count(self):
3093: (8)                    """Number of chapters."""
3094: (8)                    if self.is_closed:
```

```
3095: (12)                      raise ValueError("document closed")
3096: (8)                   return mupdf.fz_count_chapters( self.this)
3097: (4)               def chapter_page_count(self, chapter):
3098: (8)                   """Page count of chapter."""
3099: (8)                   if self.is_closed:
3100: (12)                      raise ValueError("document closed")
3101: (8)                   chapters = mupdf.fz_count_chapters( self.this)
3102: (8)                   if chapter < 0 or chapter >= chapters:
3103: (12)                      raise ValueError( "bad chapter number")
3104: (8)                   pages = mupdf.fz_count_chapter_pages( self.this, chapter)
3105: (8)                   return pages
3106: (4)               def close(self):
3107: (8)                   """Close document."""
3108: (8)                   if getattr(self, "is_closed", True):
3109: (12)                      raise ValueError("document closed")
3110: (8)                   if hasattr(self, "_outline") and self._outline:
3111: (12)                      self._outline = None
3112: (8)                   self._reset_page_refs()
3113: (8)                   self.is_closed    = True
3114: (8)                   self.Graftmaps = {} # Fixes test_3140().
3115: (8)                   self.this = None
3116: (4)               def convert_to_pdf(self, from_page=0, to_page=-1, rotate=0):
3117: (8)                   """Convert document to a PDF, selecting page range and optional
rotation. Output bytes object."""
3118: (8)                   if self.is_closed or self.is_encrypted:
3119: (12)                      raise ValueError("document closed or encrypted")
3120: (8)                   fz_doc = self.this
3121: (8)                   fp = from_page
3122: (8)                   tp = to_page
3123: (8)                   srcCount = mupdf.fz_count_pages(fz_doc)
3124: (8)                   if fp < 0:
3125: (12)                      fp = 0
3126: (8)                   if fp > srcCount - 1:
3127: (12)                      fp = srcCount - 1
3128: (8)                   if tp < 0:
3129: (12)                      tp = srcCount - 1
3130: (8)                   if tp > srcCount - 1:
3131: (12)                      tp = srcCount - 1
3132: (8)                   len0 = len(JM_mupdf_warnings_store)
3133: (8)                   doc = JM_convert_to_pdf(fz_doc, fp, tp, rotate)
3134: (8)                   len1 = len(JM_mupdf_warnings_store)
3135: (8)                   for i in range(len0, len1):
3136: (12)                      message(f'{JM_mupdf_warnings_store[i]}')
3137: (8)                   return doc
3138: (4)               def copy_page(self, pno: int, to: int =-1):
3139: (8)                   """Copy a page within a PDF document.
3140: (8)                   This will only create another reference of the same page object.
3141: (8)                   Args:
3142: (12)                      pno: source page number
3143: (12)                      to: put before this page, '-1' means after last page.
3144: (8)                   """
3145: (8)                   if self.is_closed:
3146: (12)                      raise ValueError("document closed")
3147: (8)                   page_count = len(self)
3148: (8)                   if (
3149: (16)                          pno not in range(page_count)
3150: (16)                          or to not in range(-1, page_count)
3151: (16)                          ):
3152: (12)                      raise ValueError("bad page number(s)")
3153: (8)                   before = 1
3154: (8)                   copy = 1
3155: (8)                   if to == -1:
3156: (12)                      to = page_count - 1
3157: (12)                      before = 0
3158: (8)                   return self._move_copy_page(pno, to, before, copy)
3159: (4)               def del_xml_metadata(self):
3160: (8)                   """Delete XML metadata."""
3161: (8)                   if self.is_closed or self.is_encrypted:
3162: (12)                      raise ValueError("document closed or encrypted")
```

```
(8)                 pdf = _as_pdf_document(self)
(8)                 root = mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf), PDF_NAME('Root'))
(8)                 if root.m_internal:
(12)                    mupdf.pdf_dict_del( root, PDF_NAME('Metadata'))
(4)             def delete_page(self, pno: int =-1):
(8)                 """ Delete one page from a PDF.
(8)                 """
(8)                 if not self.is_pdf:
(12)                    raise ValueError("is no PDF")
(8)                 if self.is_closed:
(12)                    raise ValueError("document closed")
(8)                 page_count = self.page_count
(8)                 while pno < 0:
(12)                    pno += page_count
(8)                 if pno >= page_count:
(12)                    raise ValueError("bad page number(s)")
(8)                 toc = self.get_toc()
(8)                 ol_xrefs = self.get_outline_xrefs()
(8)                 for i, item in enumerate(toc):
(12)                    if item[2] == pno + 1:
(16)                        self._remove_toc_item(ol_xrefs[i])
(8)                 self._remove_links_to(frozenset((pno,)))
(8)                 self._delete_page(pno)
(8)                 self._reset_page_refs()
(4)             def delete_pages(self, *args, **kw):
(8)                 """Delete pages from a PDF.
(8)                 Args:
(12)                    Either keywords 'from_page'/'to_page', or two integers to
(12)                    specify the first/last page to delete.
(12)                    Or a list/tuple/range object, which can contain arbitrary
(12)                    page numbers.
(8)                 """
(8)                 if not self.is_pdf:
(12)                    raise ValueError("is no PDF")
(8)                 if self.is_closed:
(12)                    raise ValueError("document closed")
(8)                 page_count = self.page_count  # page count of document
(8)                 f = t = -1
(8)                 if kw:  # check if keywords were used
(12)                    if args:  # then no positional args are allowed
(16)                        raise ValueError("cannot mix keyword and positional argument")
(12)                    f = kw.get("from_page", -1)  # first page to delete
(12)                    t = kw.get("to_page", -1)  # last page to delete
(12)                    while f < 0:
(16)                        f += page_count
(12)                    while t < 0:
(16)                        t += page_count
(12)                    if not f <= t < page_count:
(16)                        raise ValueError("bad page number(s)")
(12)                    numbers = tuple(range(f, t + 1))
(8)                 else:
(12)                    if len(args) > 2 or args == []:
(16)                        raise ValueError("need 1 or 2 positional arguments")
(12)                    if len(args) == 2:
(16)                        f, t = args
(16)                        if not (type(f) is int and type(t) is int):
(20)                            raise ValueError("both arguments must be int")
(16)                        if f > t:
(20)                            f, t = t, f
(16)                        if not f <= t < page_count:
(20)                            raise ValueError("bad page number(s)")
(16)                        numbers = tuple(range(f, t + 1))
(12)                    else:
(16)                        r = args[0]
(16)                        if type(r) not in (int, range, list, tuple):
(20)                            raise ValueError("need int or sequence if one argument")
(16)                        numbers = tuple(r)
(8)                 numbers = list(map(int, set(numbers)))  # ensure unique integers
(8)                 if numbers == []:
```

```
(12)                        message("nothing to delete")
(12)                        return
(8)                     numbers.sort()
(8)                     if numbers[0] < 0 or numbers[-1] >= page_count:
(12)                        raise ValueError("bad page number(s)")
(8)                     frozen_numbers = frozenset(numbers)
(8)                     toc = self.get_toc()
(8)                     for i, xref in enumerate(self.get_outline_xrefs()):
(12)                        if toc[i][2] - 1 in frozen_numbers:
(16)                            self._remove_toc_item(xref)  # remove target in PDF object
(8)                     self._remove_links_to(frozen_numbers)
(8)                     for i in reversed(numbers):  # delete pages, last to first
(12)                        self._delete_page(i)
(8)                     self._reset_page_refs()
(4)                 def embfile_add(self,
(12)                        name: str,
(12)                        buffer_: typing.ByteString,
(12)                        filename: OptStr =None,
(12)                        ufilename: OptStr =None,
(12)                        desc: OptStr =None,
(12)                        ) -> None:
(8)                     """Add an item to the EmbeddedFiles array.
(8)                     Args:
(12)                        name: name of the new item, must not already exist.
(12)                        buffer_: (binary data) the file content.
(12)                        filename: (str) the file name, default: the name
(12)                        ufilename: (unicode) the file name, default: filename
(12)                        desc: (str) the description.
(8)                     """
(8)                     filenames = self.embfile_names()
(8)                     msg = "Name '%s' already exists." % str(name)
(8)                     if name in filenames:
(12)                        raise ValueError(msg)
(8)                     if filename is None:
(12)                        filename = name
(8)                     if ufilename is None:
(12)                        ufilename = filename
(8)                     if desc is None:
(12)                        desc = name
(8)                     xref = self._embfile_add(
(16)                            name,
(16)                            buffer_=buffer_,
(16)                            filename=filename,
(16)                            ufilename=ufilename,
(16)                            desc=desc,
(16)                            )
(8)                     date = get_pdf_now()
(8)                     self.xref_set_key(xref, "Type", "/EmbeddedFile")
(8)                     self.xref_set_key(xref, "Params/CreationDate", get_pdf_str(date))
(8)                     self.xref_set_key(xref, "Params/ModDate", get_pdf_str(date))
(8)                     return xref
(4)                 def embfile_count(self) -> int:
(8)                     """Get number of EmbeddedFiles."""
(8)                     return len(self.embfile_names())
(4)                 def embfile_del(self, item: typing.Union[int, str]):
(8)                     """Delete an entry from EmbeddedFiles.
(8)                     Notes:
(12)                        The argument must be name or index of an EmbeddedFiles item.
(12)                        Physical deletion of data will happen on save to a new
(12)                        file with appropriate garbage option.
(8)                     Args:
(12)                        item: name or number of item.
(8)                     Returns:
(12)                        None
(8)                     """
(8)                     idx = self._embeddedFileIndex(item)
(8)                     return self._embfile_del(idx)
(4)                 def embfile_get(self, item: typing.Union[int, str]) -> bytes:
(8)                     """Get the content of an item in the EmbeddedFiles array.
```

```
(8)                 Args:
(12)                    item: number or name of item.
(8)                 Returns:
(12)                    (bytes) The file content.
(8)                 """
(8)                 idx = self._embeddedFileIndex(item)
(8)                 return self._embeddedFileGet(idx)
(4)             def embfile_info(self, item: typing.Union[int, str]) -> dict:
(8)                 """Get information of an item in the EmbeddedFiles array.
(8)                 Args:
(12)                    item: number or name of item.
(8)                 Returns:
(12)                    Information dictionary.
(8)                 """
(8)                 idx = self._embeddedFileIndex(item)
(8)                 infodict = {"name": self.embfile_names()[idx]}
(8)                 xref = self._embfile_info(idx, infodict)
(8)                 t, date = self.xref_get_key(xref, "Params/CreationDate")
(8)                 if t != "null":
(12)                    infodict["creationDate"] = date
(8)                 t, date = self.xref_get_key(xref, "Params/ModDate")
(8)                 if t != "null":
(12)                    infodict["modDate"] = date
(8)                 t, md5 = self.xref_get_key(xref, "Params/CheckSum")
(8)                 if t != "null":
(12)                    infodict["checksum"] = binascii.hexlify(md5.encode()).decode()
(8)                 return infodict
(4)             def embfile_names(self) -> list:
(8)                 """Get list of names of EmbeddedFiles."""
(8)                 filenames = []
(8)                 self._embfile_names(filenames)
(8)                 return filenames
(4)             def embfile_upd(self,
(12)                    item: typing.Union[int, str],
(12)                    buffer_: OptBytes =None,
(12)                    filename: OptStr =None,
(12)                    ufilename: OptStr =None,
(12)                    desc: OptStr =None,
(12)                    ) -> None:
(8)                 """Change an item of the EmbeddedFiles array.
(8)                 Notes:
(12)                    Only provided parameters are changed. If all are omitted,
(12)                    the method is a no-op.
(8)                 Args:
(12)                    item: number or name of item.
(12)                    buffer_: (binary data) the new file content.
(12)                    filename: (str) the new file name.
(12)                    ufilename: (unicode) the new filen ame.
(12)                    desc: (str) the new description.
(8)                 """
(8)                 idx = self._embeddedFileIndex(item)
(8)                 xref = self._embfile_upd(
(16)                        idx,
(16)                        buffer_=buffer_,
(16)                        filename=filename,
(16)                        ufilename=ufilename,
(16)                        desc=desc,
(16)                        )
(8)                 date = get_pdf_now()
(8)                 self.xref_set_key(xref, "Params/ModDate", get_pdf_str(date))
(8)                 return xref
(4)             def extract_font(self, xref=0, info_only=0, named=None):
(8)                 '''
(8)                 Get a font by xref. Returns a tuple or dictionary.
(8)                 '''
(8)                 pdf = _as_pdf_document(self)
(8)                 obj = mupdf.pdf_load_object(pdf, xref)
(8)                 type_ = mupdf.pdf_dict_get(obj, PDF_NAME('Type'))
(8)                 subtype = mupdf.pdf_dict_get(obj, PDF_NAME('Subtype'))
```

```
3370: (8)                   if (mupdf.pdf_name_eq(type_, PDF_NAME('Font'))
3371: (16)                          and not mupdf.pdf_to_name( subtype).startswith('CIDFontType')
3372: (16)                          ):
3373: (12)                      basefont = mupdf.pdf_dict_get(obj, PDF_NAME('BaseFont'))
3374: (12)                      if not basefont.m_internal or mupdf.pdf_is_null(basefont):
3375: (16)                          bname = mupdf.pdf_dict_get(obj, PDF_NAME('Name'))
3376: (12)                      else:
3377: (16)                          bname = basefont
3378: (12)                      ext = JM_get_fontextension(pdf, xref)
3379: (12)                      if ext != 'n/a' and not info_only:
3380: (16)                          buffer_ = JM_get_fontbuffer(pdf, xref)
3381: (16)                          bytes_ = JM_BinFromBuffer(buffer_)
3382: (12)                      else:
3383: (16)                          bytes_ = b''
3384: (12)                      if not named:
3385: (16)                          rc = (
3386: (24)                                  JM_EscapeStrFromStr(mupdf.pdf_to_name(bname)),
3387: (24)                                  JM_UnicodeFromStr(ext),
3388: (24)                                  JM_UnicodeFromStr(mupdf.pdf_to_name(subtype)),
3389: (24)                                  bytes_,
3390: (24)                                  )
3391: (12)                      else:
3392: (16)                          rc = {
3393: (24)                                  dictkey_name:
JM_EscapeStrFromStr(mupdf.pdf_to_name(bname)),
3394: (24)                                  dictkey_ext: JM_UnicodeFromStr(ext),
3395: (24)                                  dictkey_type:
JM_UnicodeFromStr(mupdf.pdf_to_name(subtype)),
3396: (24)                                  dictkey_content: bytes_,
3397: (24)                                  }
3398: (8)                   else:
3399: (12)                      if not named:
3400: (16)                          rc = '', '', '', b''
3401: (12)                      else:
3402: (16)                          rc = {
3403: (24)                                  dictkey_name: '',
3404: (24)                                  dictkey_ext: '',
3405: (24)                                  dictkey_type: '',
3406: (24)                                  dictkey_content: b'',
3407: (24)                                  }
3408: (8)                   return rc
3409: (4)               def extract_image(self, xref):
3410: (8)                   """Get image by xref. Returns a dictionary."""
3411: (8)                   if self.is_closed or self.is_encrypted:
3412: (12)                      raise ValueError("document closed or encrypted")
3413: (8)                   pdf = _as_pdf_document(self)
3414: (8)                   img_type = 0
3415: (8)                   smask = 0
3416: (8)                   if not _INRANGE(xref, 1, mupdf.pdf_xref_len(pdf)-1):
3417: (12)                      raise ValueError( MSG_BAD_XREF)
3418: (8)                   obj = mupdf.pdf_new_indirect(pdf, xref, 0)
3419: (8)                   subtype = mupdf.pdf_dict_get(obj, PDF_NAME('Subtype'))
3420: (8)                   if not mupdf.pdf_name_eq(subtype, PDF_NAME('Image')):
3421: (12)                      raise ValueError( "not an image")
3422: (8)                   o = mupdf.pdf_dict_geta(obj, PDF_NAME('SMask'), PDF_NAME('Mask'))
3423: (8)                   if o.m_internal:
3424: (12)                      smask = mupdf.pdf_to_num(o)
3425: (8)                   if mupdf.pdf_is_jpx_image(obj):
3426: (12)                      img_type = mupdf.FZ_IMAGE_JPX
3427: (12)                      res = mupdf.pdf_load_stream(obj)
3428: (12)                      ext = "jpx"
3429: (8)                   if JM_is_jbig2_image(obj):
3430: (12)                      img_type = mupdf.FZ_IMAGE_JBIG2
3431: (12)                      res = mupdf.pdf_load_stream(obj)
3432: (12)                      ext = "jb2"
3433: (8)                   res = mupdf.pdf_load_raw_stream(obj)
3434: (8)                   if img_type == mupdf.FZ_IMAGE_UNKNOWN:
3435: (12)                      res = mupdf.pdf_load_raw_stream(obj)
3436: (12)                      _, c = mupdf.fz_buffer_storage(res)
```

```
3437: (12)                      img_type = mupdf.fz_recognize_image_format(c)
3438: (12)                      ext = JM_image_extension(img_type)
3439: (8)                   if img_type == mupdf.FZ_IMAGE_UNKNOWN:
3440: (12)                      res = None
3441: (12)                      img = mupdf.pdf_load_image(pdf, obj)
3442: (12)                      ll_cbuf = mupdf.ll_fz_compressed_image_buffer(img.m_internal)
3443: (12)                      if (ll_cbuf
3444: (20)                              and ll_cbuf.params.type not in (
3445: (24)                                  mupdf.FZ_IMAGE_RAW,
3446: (24)                                  mupdf.FZ_IMAGE_FAX,
3447: (24)                                  mupdf.FZ_IMAGE_FLATE,
3448: (24)                                  mupdf.FZ_IMAGE_LZW,
3449: (24)                                  mupdf.FZ_IMAGE_RLD,
3450: (24)                                  )
3451: (20)                              ):
3452: (16)                          img_type = ll_cbuf.params.type
3453: (16)                          ext = JM_image_extension(img_type)
3454: (16)                          res = mupdf.FzBuffer(mupdf.ll_fz_keep_buffer(ll_cbuf.buffer))
3455: (12)                      else:
3456: (16)                          res = mupdf.fz_new_buffer_from_image_as_png(
3457: (24)                                  img,
3458: (24)                                  mupdf.FzColorParams(mupdf.fz_default_color_params),
3459: (24)                                  )
3460: (16)                          ext = "png"
3461: (8)                   else:
3462: (12)                      img = mupdf.fz_new_image_from_buffer(res)
3463: (8)                   xres, yres = mupdf.fz_image_resolution(img)
3464: (8)                   width = img.w()
3465: (8)                   height = img.h()
3466: (8)                   colorspace = img.n()
3467: (8)                   bpc = img.bpc()
3468: (8)                   cs_name = mupdf.fz_colorspace_name(img.colorspace())
3469: (8)                   rc = dict()
3470: (8)                   rc[ dictkey_ext] = ext
3471: (8)                   rc[ dictkey_smask] = smask
3472: (8)                   rc[ dictkey_width] = width
3473: (8)                   rc[ dictkey_height] = height
3474: (8)                   rc[ dictkey_colorspace] = colorspace
3475: (8)                   rc[ dictkey_bpc] = bpc
3476: (8)                   rc[ dictkey_xres] = xres
3477: (8)                   rc[ dictkey_yres] = yres
3478: (8)                   rc[ dictkey_cs_name] = cs_name
3479: (8)                   rc[ dictkey_image] = JM_BinFromBuffer(res)
3480: (8)                   return rc
3481: (4)               def ez_save(
3482: (12)                      self,
3483: (12)                      filename,
3484: (12)                      garbage=3,
3485: (12)                      clean=False,
3486: (12)                      deflate=True,
3487: (12)                      deflate_images=True,
3488: (12)                      deflate_fonts=True,
3489: (12)                      incremental=False,
3490: (12)                      ascii=False,
3491: (12)                      expand=False,
3492: (12)                      linear=False,
3493: (12)                      pretty=False,
3494: (12)                      encryption=1,
3495: (12)                      permissions=4095,
3496: (12)                      owner_pw=None,
3497: (12)                      user_pw=None,
3498: (12)                      no_new_id=True,
3499: (12)                      preserve_metadata=1,
3500: (12)                      use_objstms=1,
3501: (12)                      compression_effort=0,
3502: (12)                      ):
3503: (8)                   '''
3504: (8)                   Save PDF using some different defaults
3505: (8)                   '''
```

```
3506: (8)                    return self.save(
3507: (16)                           filename,
3508: (16)                           garbage=garbage,
3509: (16)                           clean=clean,
3510: (16)                           deflate=deflate,
3511: (16)                           deflate_images=deflate_images,
3512: (16)                           deflate_fonts=deflate_fonts,
3513: (16)                           incremental=incremental,
3514: (16)                           ascii=ascii,
3515: (16)                           expand=expand,
3516: (16)                           linear=linear,
3517: (16)                           pretty=pretty,
3518: (16)                           encryption=encryption,
3519: (16)                           permissions=permissions,
3520: (16)                           owner_pw=owner_pw,
3521: (16)                           user_pw=user_pw,
3522: (16)                           no_new_id=no_new_id,
3523: (16)                           preserve_metadata=preserve_metadata,
3524: (16)                           use_objstms=use_objstms,
3525: (16)                           compression_effort=compression_effort,
3526: (16)                           )
3527: (4)                def find_bookmark(self, bm):
3528: (8)                    """Find new location after layouting a document."""
3529: (8)                    if self.is_closed or self.is_encrypted:
3530: (12)                       raise ValueError("document closed or encrypted")
3531: (8)                    location = mupdf.fz_lookup_bookmark2( self.this, bm)
3532: (8)                    return location.chapter, location.page
3533: (4)                def fullcopy_page(self, pno, to=-1):
3534: (8)                    """Make a full page duplicate."""
3535: (8)                    pdf = _as_pdf_document(self)
3536: (8)                    page_count = mupdf.pdf_count_pages( pdf)
3537: (8)                    try:
3538: (12)                       if (not _INRANGE(pno, 0, page_count - 1)
3539: (20)                               or not _INRANGE(to, -1, page_count - 1)
3540: (20)                               ):
3541: (16)                           raise ValueError( MSG_BAD_PAGENO)
3542: (12)                       page1 = mupdf.pdf_resolve_indirect( mupdf.pdf_lookup_page_obj(
pdf, pno))
3543: (12)                       page2 = mupdf.pdf_deep_copy_obj( page1)
3544: (12)                       old_annots = mupdf.pdf_dict_get( page2, PDF_NAME('Annots'))
3545: (12)                       if old_annots.m_internal:
3546: (16)                           n = mupdf.pdf_array_len( old_annots)
3547: (16)                           new_annots = mupdf.pdf_new_array( pdf, n)
3548: (16)                           for i in range(n):
3549: (20)                               o = mupdf.pdf_array_get( old_annots, i)
3550: (20)                               subtype = mupdf.pdf_dict_get( o, PDF_NAME('Subtype'))
3551: (20)                               if mupdf.pdf_name_eq( subtype, PDF_NAME('Popup')):
3552: (24)                                   continue
3553: (20)                               if mupdf.pdf_dict_gets( o, "IRT").m_internal:
3554: (24)                                   continue
3555: (20)                               copy_o = mupdf.pdf_deep_copy_obj(
mupdf.pdf_resolve_indirect( o))
3556: (20)                               xref = mupdf.pdf_create_object( pdf)
3557: (20)                               mupdf.pdf_update_object( pdf, xref, copy_o)
3558: (20)                               copy_o = mupdf.pdf_new_indirect( pdf, xref, 0)
3559: (20)                               mupdf.pdf_dict_del( copy_o, PDF_NAME('Popup'))
3560: (20)                               mupdf.pdf_dict_del( copy_o, PDF_NAME('P'))
3561: (20)                               mupdf.pdf_array_push( new_annots, copy_o)
3562: (16)                           mupdf.pdf_dict_put( page2, PDF_NAME('Annots'), new_annots)
3563: (12)                       res = JM_read_contents( page1)
3564: (12)                       if res.m_internal:
3565: (16)                           contents = mupdf.pdf_add_stream( pdf,
mupdf.fz_new_buffer_from_copied_data( b" "), mupdf.PdfObj(), 0)
3566: (16)                           JM_update_stream( pdf, contents, res, 1)
3567: (16)                           mupdf.pdf_dict_put( page2, PDF_NAME('Contents'), contents)
3568: (12)                       xref = mupdf.pdf_create_object( pdf)   # get new xref
3569: (12)                       mupdf.pdf_update_object( pdf, xref, page2) # store new page
3570: (12)                       page2 = mupdf.pdf_new_indirect( pdf, xref, 0)  # reread object
3571: (12)                       mupdf.pdf_insert_page( pdf, to, page2) # and store the page
```

```
3572: (8)                    finally:
3573: (12)                       mupdf.ll_pdf_drop_page_tree( pdf.m_internal)
3574: (8)                    self._reset_page_refs()
3575: (4)                def get_layer(self, config=-1):
3576: (8)                    """Content of ON, OFF, RBGroups of an OC layer."""
3577: (8)                    pdf = _as_pdf_document(self)
3578: (8)                    ocp = mupdf.pdf_dict_getl(
3579: (16)                           mupdf.pdf_trailer( pdf),
3580: (16)                           PDF_NAME('Root'),
3581: (16)                           PDF_NAME('OCProperties'),
3582: (16)                           )
3583: (8)                    if not ocp.m_internal:
3584: (12)                       return
3585: (8)                    if config == -1:
3586: (12)                       obj = mupdf.pdf_dict_get( ocp, PDF_NAME('D'))
3587: (8)                    else:
3588: (12)                       obj = mupdf.pdf_array_get(
3589: (20)                               mupdf.pdf_dict_get( ocp, PDF_NAME('Configs')),
3590: (20)                               config,
3591: (20)                               )
3592: (8)                    if not obj.m_internal:
3593: (12)                       raise ValueError( MSG_BAD_OC_CONFIG)
3594: (8)                    rc = JM_get_ocg_arrays( obj)
3595: (8)                    return rc
3596: (4)                def get_layers(self):
3597: (8)                    """Show optional OC layers."""
3598: (8)                    pdf = _as_pdf_document(self)
3599: (8)                    n = mupdf.pdf_count_layer_configs( pdf)
3600: (8)                    if n == 1:
3601: (12)                       obj = mupdf.pdf_dict_getl(
3602: (20)                               mupdf.pdf_trailer( pdf),
3603: (20)                               PDF_NAME('Root'),
3604: (20)                               PDF_NAME('OCProperties'),
3605: (20)                               PDF_NAME('Configs'),
3606: (20)                               )
3607: (12)                       if not mupdf.pdf_is_array( obj):
3608: (16)                           n = 0
3609: (8)                    rc = []
3610: (8)                    info = mupdf.PdfLayerConfig()
3611: (8)                    for i in range(n):
3612: (12)                       mupdf.pdf_layer_config_info( pdf, i, info)
3613: (12)                       item = {
3614: (20)                               "number": i,
3615: (20)                               "name": info.name,
3616: (20)                               "creator": info.creator,
3617: (20)                               }
3618: (12)                       rc.append( item)
3619: (8)                    return rc
3620: (4)                def get_new_xref(self):
3621: (8)                    """Make new xref."""
3622: (8)                    if self.is_closed or self.is_encrypted:
3623: (12)                       raise ValueError("document closed or encrypted")
3624: (8)                    pdf = _as_pdf_document(self)
3625: (8)                    xref = 0
3626: (8)                    ENSURE_OPERATION(pdf)
3627: (8)                    xref = mupdf.pdf_create_object(pdf)
3628: (8)                    return xref
3629: (4)                def get_ocgs(self):
3630: (8)                    """Show existing optional content groups."""
3631: (8)                    ci = mupdf.pdf_new_name( "CreatorInfo")
3632: (8)                    pdf = _as_pdf_document(self)
3633: (8)                    ocgs = mupdf.pdf_dict_getl(
3634: (16)                           mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf),
PDF_NAME('Root')),
3635: (16)                           PDF_NAME('OCProperties'),
3636: (16)                           PDF_NAME('OCGs'),
3637: (16)                           )
3638: (8)                    rc = dict()
3639: (8)                    if not mupdf.pdf_is_array( ocgs):
```

```
3640: (12)                      return rc
3641: (8)                   n = mupdf.pdf_array_len( ocgs)
3642: (8)                   for i in range(n):
3643: (12)                      ocg = mupdf.pdf_array_get( ocgs, i)
3644: (12)                      xref = mupdf.pdf_to_num( ocg)
3645: (12)                      name = mupdf.pdf_to_text_string( mupdf.pdf_dict_get( ocg,
PDF_NAME('Name')))
3646: (12)                      obj = mupdf.pdf_dict_getl( ocg, PDF_NAME('Usage'), ci,
PDF_NAME('Subtype'))
3647: (12)                      usage = None
3648: (12)                      if obj.m_internal:
3649: (16)                          usage = mupdf.pdf_to_name( obj)
3650: (12)                      intents = list()
3651: (12)                      intent = mupdf.pdf_dict_get( ocg, PDF_NAME('Intent'))
3652: (12)                      if intent.m_internal:
3653: (16)                          if mupdf.pdf_is_name( intent):
3654: (20)                              intents.append( mupdf.pdf_to_name( intent))
3655: (16)                          elif mupdf.pdf_is_array( intent):
3656: (20)                              m = mupdf.pdf_array_len( intent)
3657: (20)                              for j in range(m):
3658: (24)                                  o = mupdf.pdf_array_get( intent, j)
3659: (24)                                  if mupdf.pdf_is_name( o):
3660: (28)                                      intents.append( mupdf.pdf_to_name( o))
3661: (12)                      hidden = mupdf.pdf_is_ocg_hidden( pdf, mupdf.PdfObj(), usage, ocg)
3662: (12)                      item = {
3663: (20)                              "name": name,
3664: (20)                              "intent": intents,
3665: (20)                              "on": not hidden,
3666: (20)                              "usage": usage,
3667: (20)                              }
3668: (12)                      temp = xref
3669: (12)                      rc[ temp] = item
3670: (8)                   return rc
3671: (4)               def get_outline_xrefs(self):
3672: (8)                   """Get list of outline xref numbers."""
3673: (8)                   xrefs = []
3674: (8)                   pdf = _as_pdf_document(self, required=0)
3675: (8)                   if not pdf.m_internal:
3676: (12)                      return xrefs
3677: (8)                   root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
3678: (8)                   if not root.m_internal:
3679: (12)                      return xrefs
3680: (8)                   olroot = mupdf.pdf_dict_get(root, PDF_NAME('Outlines'))
3681: (8)                   if not olroot.m_internal:
3682: (12)                      return xrefs
3683: (8)                   first = mupdf.pdf_dict_get(olroot, PDF_NAME('First'))
3684: (8)                   if not first.m_internal:
3685: (12)                      return xrefs
3686: (8)                   xrefs = JM_outline_xrefs(first, xrefs)
3687: (8)                   return xrefs
3688: (4)               def get_page_fonts(self, pno: int, full: bool =False) -> list:
3689: (8)                   """Retrieve a list of fonts used on a page.
3690: (8)                   """
3691: (8)                   if self.is_closed or self.is_encrypted:
3692: (12)                      raise ValueError("document closed or encrypted")
3693: (8)                   if not self.is_pdf:
3694: (12)                      return ()
3695: (8)                   if type(pno) is not int:
3696: (12)                      try:
3697: (16)                          pno = pno.number
3698: (12)                      except Exception:
3699: (16)                          exception_info()
3700: (16)                          raise ValueError("need a Page or page number")
3701: (8)                   val = self._getPageInfo(pno, 1)
3702: (8)                   if full is False:
3703: (12)                      return [v[:-1] for v in val]
3704: (8)                   return val
3705: (4)               def get_page_images(self, pno: int, full: bool =False) -> list:
3706: (8)                   """Retrieve a list of images used on a page.
```

```
3707: (8)                    """
3708: (8)                    if self.is_closed or self.is_encrypted:
3709: (12)                       raise ValueError("document closed or encrypted")
3710: (8)                    if not self.is_pdf:
3711: (12)                       return ()
3712: (8)                    val = self._getPageInfo(pno, 2)
3713: (8)                    if full is False:
3714: (12)                       return [v[:-1] for v in val]
3715: (8)                    return val
3716: (4)                def get_page_xobjects(self, pno: int) -> list:
3717: (8)                    """Retrieve a list of XObjects used on a page.
3718: (8)                    """
3719: (8)                    if self.is_closed or self.is_encrypted:
3720: (12)                       raise ValueError("document closed or encrypted")
3721: (8)                    if not self.is_pdf:
3722: (12)                       return ()
3723: (8)                    val = self._getPageInfo(pno, 3)
3724: (8)                    return val
3725: (4)                def get_sigflags(self):
3726: (8)                    """Get the /SigFlags value."""
3727: (8)                    pdf = _as_pdf_document(self, required=0)
3728: (8)                    if not pdf.m_internal:
3729: (12)                       return -1   # not a PDF
3730: (8)                    sigflags = mupdf.pdf_dict_getl(
3731: (16)                           mupdf.pdf_trailer(pdf),
3732: (16)                           PDF_NAME('Root'),
3733: (16)                           PDF_NAME('AcroForm'),
3734: (16)                           PDF_NAME('SigFlags'),
3735: (16)                           )
3736: (8)                    sigflag = -1
3737: (8)                    if sigflags.m_internal:
3738: (12)                       sigflag = mupdf.pdf_to_int(sigflags)
3739: (8)                    return sigflag
3740: (4)                def get_xml_metadata(self):
3741: (8)                    """Get document XML metadata."""
3742: (8)                    xml = None
3743: (8)                    pdf = _as_pdf_document(self, required=0)
3744: (8)                    if pdf.m_internal:
3745: (12)                       xml = mupdf.pdf_dict_getl(
3746: (20)                               mupdf.pdf_trailer(pdf),
3747: (20)                               PDF_NAME('Root'),
3748: (20)                               PDF_NAME('Metadata'),
3749: (20)                               )
3750: (8)                    if xml is not None and xml.m_internal:
3751: (12)                       buff = mupdf.pdf_load_stream(xml)
3752: (12)                       rc = JM_UnicodeFromBuffer(buff)
3753: (8)                    else:
3754: (12)                       rc = ''
3755: (8)                    return rc
3756: (4)                def init_doc(self):
3757: (8)                    if self.is_encrypted:
3758: (12)                       raise ValueError("cannot initialize - document still encrypted")
3759: (8)                    self._outline = self._loadOutline()
3760: (8)                    self.metadata = dict(
3761: (20)                               [
3762: (24)                                   (k,self._getMetadata(v)) for k,v in {
3763: (28)                                       'format':'format',
3764: (28)                                       'title':'info:Title',
3765: (28)                                       'author':'info:Author',
3766: (28)                                       'subject':'info:Subject',
3767: (28)                                       'keywords':'info:Keywords',
3768: (28)                                       'creator':'info:Creator',
3769: (28)                                       'producer':'info:Producer',
3770: (28)                                       'creationDate':'info:CreationDate',
3771: (28)                                       'modDate':'info:ModDate',
3772: (28)                                       'trapped':'info:Trapped'
3773: (28)                                       }.items()
3774: (20)                               ]
3775: (16)                           )
```

```
(8)                     self.metadata['encryption'] = None if
self._getMetadata('encryption')=='None' else self._getMetadata('encryption')
(4)                 def insert_file(self,
(12)                        infile,
(12)                        from_page=-1,
(12)                        to_page=-1,
(12)                        start_at=-1,
(12)                        rotate=-1,
(12)                        links=True,
(12)                        annots=True,
(12)                        show_progress=0,
(12)                        final=1,
(12)                        ):
(8)                     '''
(8)                     Insert an arbitrary supported document to an existing PDF.
(8)                     The infile may be given as a filename, a Document or a Pixmap. Other
(8)                     parameters - where applicable - equal those of insert_pdf().
(8)                     '''
(8)                     src = None
(8)                     if isinstance(infile, Pixmap):
(12)                        if infile.colorspace.n > 3:
(16)                            infile = Pixmap(csRGB, infile)
(12)                        src = Document("png", infile.tobytes())
(8)                     elif isinstance(infile, Document):
(12)                        src = infile
(8)                     else:
(12)                        src = Document(infile)
(8)                     if not src:
(12)                        raise ValueError("bad infile parameter")
(8)                     if not src.is_pdf:
(12)                        pdfbytes = src.convert_to_pdf()
(12)                        src = Document("pdf", pdfbytes)
(8)                     return self.insert_pdf(
(16)                            src,
(16)                            from_page=from_page,
(16)                            to_page=to_page,
(16)                            start_at=start_at,
(16)                            rotate=rotate,
(16)                            links=links,
(16)                            annots=annots,
(16)                            show_progress=show_progress,
(16)                            final=final,
(16)                            )
(4)                 def insert_pdf(
(12)                        self,
(12)                        docsrc,
(12)                        from_page=-1,
(12)                        to_page=-1,
(12)                        start_at=-1,
(12)                        rotate=-1,
(12)                        links=1,
(12)                        annots=1,
(12)                        show_progress=0,
(12)                        final=1,
(12)                        _gmap=None,
(12)                        ):
(8)                     """Insert a page range from another PDF.
(8)                     Args:
(12)                        docsrc: PDF to copy from. Must be different object, but may be
same file.
(12)                        from_page: (int) first source page to copy, 0-based, default 0.
(12)                        to_page: (int) last source page to copy, 0-based, default last
page.
(12)                        start_at: (int) from_page will become this page number in target.
(12)                        rotate: (int) rotate copied pages, default -1 is no change.
(12)                        links: (int/bool) whether to also copy links.
(12)                        annots: (int/bool) whether to also copy annotations.
(12)                        show_progress: (int) progress message interval, 0 is no messages.
(12)                        final: (bool) indicates last insertion from this source PDF.
```

```
3842: (12)                      _gmap: internal use only
3843: (8)                   Copy sequence reversed if from_page > to_page."""
3844: (8)                   if self.is_closed or self.is_encrypted:
3845: (12)                      raise ValueError("document closed or encrypted")
3846: (8)                   if self._graft_id == docsrc._graft_id:
3847: (12)                      raise ValueError("source and target cannot be same object")
3848: (8)                   sa = start_at
3849: (8)                   if sa < 0:
3850: (12)                      sa = self.page_count
3851: (8)                   if len(docsrc) > show_progress > 0:
3852: (12)                      inname = os.path.basename(docsrc.name)
3853: (12)                      if not inname:
3854: (16)                          inname = "memory PDF"
3855: (12)                      outname = os.path.basename(self.name)
3856: (12)                      if not outname:
3857: (16)                          outname = "memory PDF"
3858: (12)                      message("Inserting '%s' at '%s'" % (inname, outname))
3859: (8)                   isrt = docsrc._graft_id
3860: (8)                   _gmap = self.Graftmaps.get(isrt, None)
3861: (8)                   if _gmap is None:
3862: (12)                      _gmap = Graftmap(self)
3863: (12)                      self.Graftmaps[isrt] = _gmap
3864: (8)                   if g_use_extra:
3865: (12)                      extra_FzDocument_insert_pdf(
3866: (20)                              self.this,
3867: (20)                              docsrc.this,
3868: (20)                              from_page,
3869: (20)                              to_page,
3870: (20)                              start_at,
3871: (20)                              rotate,
3872: (20)                              links,
3873: (20)                              annots,
3874: (20)                              show_progress,
3875: (20)                              final,
3876: (20)                              _gmap,
3877: (20)                              )
3878: (8)                   else:
3879: (12)                      pdfout = _as_pdf_document(self)
3880: (12)                      pdfsrc = _as_pdf_document(docsrc)
3881: (12)                      outCount = mupdf.fz_count_pages(self)
3882: (12)                      srcCount = mupdf.fz_count_pages(docsrc.this)
3883: (12)                      fp = from_page
3884: (12)                      tp = to_page
3885: (12)                      sa = start_at
3886: (12)                      fp = max(fp, 0) # -1 = first page
3887: (12)                      fp = min(fp, srcCount - 1)  # but do not exceed last page
3888: (12)                      if tp < 0:
3889: (16)                          tp = srcCount - 1   # -1 = last page
3890: (12)                      tp = min(tp, srcCount - 1)  # but do not exceed last page
3891: (12)                      if sa < 0:
3892: (16)                          sa = outCount   # -1 = behind last page
3893: (12)                      sa = min(sa, outCount)  # but that is also the limit
3894: (12)                      if not pdfout.m_internal or not pdfsrc.m_internal:
3895: (16)                          raise TypeError( "source or target not a PDF")
3896: (12)                      ENSURE_OPERATION(pdfout)
3897: (12)                      JM_merge_range(pdfout, pdfsrc, fp, tp, sa, rotate, links, annots,
show_progress, _gmap)
3898: (8)                   self._reset_page_refs()
3899: (8)                   if links:
3900: (12)                      self._do_links(docsrc, from_page = from_page, to_page = to_page,
start_at = sa)
3901: (8)                   if final == 1:
3902: (12)                      self.Graftmaps[isrt] = None
3903: (4)               @property
3904: (4)               def is_dirty(self):
3905: (8)                   pdf = _as_pdf_document(self, required=0)
3906: (8)                   if not pdf.m_internal:
3907: (12)                      return False
3908: (8)                   r = mupdf.pdf_has_unsaved_changes(pdf)
```

```
3909: (8)                   return True if r else False
3910: (4)               @property
3911: (4)               def is_fast_webaccess(self):
3912: (8)                   '''
3913: (8)                   Check whether we have a linearized PDF.
3914: (8)                   '''
3915: (8)                   pdf = _as_pdf_document(self, required=0)
3916: (8)                   if pdf.m_internal:
3917: (12)                      return mupdf.pdf_doc_was_linearized(pdf)
3918: (8)                   return False    # gracefully handle non-PDF
3919: (4)               @property
3920: (4)               def is_form_pdf(self):
3921: (8)                   """Either False or PDF field count."""
3922: (8)                   pdf = _as_pdf_document(self, required=0)
3923: (8)                   if not pdf.m_internal:
3924: (12)                      return False
3925: (8)                   count = -1
3926: (8)                   try:
3927: (12)                      fields = mupdf.pdf_dict_getl(
3928: (20)                              mupdf.pdf_trailer(pdf),
3929: (20)                              mupdf.PDF_ENUM_NAME_Root,
3930: (20)                              mupdf.PDF_ENUM_NAME_AcroForm,
3931: (20)                              mupdf.PDF_ENUM_NAME_Fields,
3932: (20)                              )
3933: (12)                      if mupdf.pdf_is_array(fields):
3934: (16)                          count = mupdf.pdf_array_len(fields)
3935: (8)                   except Exception:
3936: (12)                      if g_exceptions_verbose:    exception_info()
3937: (12)                      return False
3938: (8)                   if count >= 0:
3939: (12)                      return count
3940: (8)                   return False
3941: (4)               @property
3942: (4)               def is_pdf(self):
3943: (8)                   """Check for PDF."""
3944: (8)                   if isinstance(self.this, mupdf.PdfDocument):
3945: (12)                      return True
3946: (8)                   if mupdf.ll_pdf_specifics(self.this.m_internal):
3947: (12)                      ret = True
3948: (8)                   else:
3949: (12)                      ret = False
3950: (8)                   return ret
3951: (4)               @property
3952: (4)               def is_reflowable(self):
3953: (8)                   """Check if document is layoutable."""
3954: (8)                   if self.is_closed:
3955: (12)                      raise ValueError("document closed")
3956: (8)                   return bool(mupdf.fz_is_document_reflowable(self))
3957: (4)               @property
3958: (4)               def is_repaired(self):
3959: (8)                   """Check whether PDF was repaired."""
3960: (8)                   pdf = _as_pdf_document(self, required=0)
3961: (8)                   if not pdf.m_internal:
3962: (12)                      return False
3963: (8)                   r = mupdf.pdf_was_repaired(pdf)
3964: (8)                   if r:
3965: (12)                      return True
3966: (8)                   return False
3967: (4)               def journal_can_do(self):
3968: (8)                   """Show if undo and / or redo are possible."""
3969: (8)                   if self.is_closed or self.is_encrypted:
3970: (12)                      raise ValueError("document closed or encrypted")
3971: (8)                   undo=0
3972: (8)                   redo=0
3973: (8)                   pdf = _as_pdf_document(self)
3974: (8)                   undo = mupdf.pdf_can_undo(pdf)
3975: (8)                   redo = mupdf.pdf_can_redo(pdf)
3976: (8)                   return {'undo': bool(undo), 'redo': bool(redo)}
3977: (4)               def journal_enable(self):
```

```
3978: (8)                    """Activate document journalling."""
3979: (8)                    if self.is_closed or self.is_encrypted:
3980: (12)                       raise ValueError("document closed or encrypted")
3981: (8)                    pdf = _as_pdf_document(self)
3982: (8)                    mupdf.pdf_enable_journal(pdf)
3983: (4)                def journal_is_enabled(self):
3984: (8)                    """Check if journalling is enabled."""
3985: (8)                    if self.is_closed or self.is_encrypted:
3986: (12)                       raise ValueError("document closed or encrypted")
3987: (8)                    pdf = _as_pdf_document(self)
3988: (8)                    enabled = pdf.m_internal and pdf.m_internal.journal
3989: (8)                    return enabled
3990: (4)                def journal_load(self, filename):
3991: (8)                    """Load a journal from a file."""
3992: (8)                    if self.is_closed or self.is_encrypted:
3993: (12)                       raise ValueError("document closed or encrypted")
3994: (8)                    pdf = _as_pdf_document(self)
3995: (8)                    if isinstance(filename, str):
3996: (12)                       mupdf.pdf_load_journal(pdf, filename)
3997: (8)                    else:
3998: (12)                       res = JM_BufferFromBytes(filename)
3999: (12)                       stm = mupdf.fz_open_buffer(res)
4000: (12)                       mupdf.pdf_deserialise_journal(pdf, stm)
4001: (8)                    if not pdf.m_internal.journal:
4002: (12)                       RAISEPY( "Journal and document do not match",
JM_Exc_FileDataError)
4003: (4)                def journal_op_name(self, step):
4004: (8)                    """Show operation name for given step."""
4005: (8)                    if self.is_closed or self.is_encrypted:
4006: (12)                       raise ValueError("document closed or encrypted")
4007: (8)                    pdf = _as_pdf_document(self)
4008: (8)                    name = mupdf.pdf_undoredo_step(pdf, step)
4009: (8)                    return name
4010: (4)                def journal_position(self):
4011: (8)                    """Show journalling state."""
4012: (8)                    if self.is_closed or self.is_encrypted:
4013: (12)                       raise ValueError("document closed or encrypted")
4014: (8)                    steps=0
4015: (8)                    pdf = _as_pdf_document(self)
4016: (8)                    rc, steps = mupdf.pdf_undoredo_state(pdf)
4017: (8)                    return rc, steps
4018: (4)                def journal_redo(self):
4019: (8)                    """Move forward in the journal."""
4020: (8)                    if self.is_closed or self.is_encrypted:
4021: (12)                       raise ValueError("document closed or encrypted")
4022: (8)                    pdf = _as_pdf_document(self)
4023: (8)                    mupdf.pdf_redo(pdf)
4024: (8)                    return True
4025: (4)                def journal_save(self, filename):
4026: (8)                    """Save journal to a file."""
4027: (8)                    if self.is_closed or self.is_encrypted:
4028: (12)                       raise ValueError("document closed or encrypted")
4029: (8)                    pdf = _as_pdf_document(self)
4030: (8)                    if isinstance(filename, str):
4031: (12)                       mupdf.pdf_save_journal(pdf, filename)
4032: (8)                    else:
4033: (12)                       out = JM_new_output_fileptr(filename)
4034: (12)                       mupdf.pdf_write_journal(pdf, out)
4035: (12)                       out.fz_close_output()
4036: (4)                def journal_start_op(self, name=None):
4037: (8)                    """Begin a journalling operation."""
4038: (8)                    if self.is_closed or self.is_encrypted:
4039: (12)                       raise ValueError("document closed or encrypted")
4040: (8)                    pdf = _as_pdf_document(self)
4041: (8)                    if not pdf.m_internal.journal:
4042: (12)                       raise RuntimeError( "Journalling not enabled")
4043: (8)                    if name:
4044: (12)                       mupdf.pdf_begin_operation(pdf, name)
4045: (8)                    else:
```

```
4046: (12)                      mupdf.pdf_begin_implicit_operation(pdf)
4047: (4)               def journal_stop_op(self):
4048: (8)                   """End a journalling operation."""
4049: (8)                   if self.is_closed or self.is_encrypted:
4050: (12)                      raise ValueError("document closed or encrypted")
4051: (8)                   pdf = _as_pdf_document(self)
4052: (8)                   mupdf.pdf_end_operation(pdf)
4053: (4)               def journal_undo(self):
4054: (8)                   """Move backwards in the journal."""
4055: (8)                   if self.is_closed or self.is_encrypted:
4056: (12)                      raise ValueError("document closed or encrypted")
4057: (8)                   pdf = _as_pdf_document(self)
4058: (8)                   mupdf.pdf_undo(pdf)
4059: (8)                   return True
4060: (4)               @property
4061: (4)               def language(self):
4062: (8)                   """Document language."""
4063: (8)                   pdf = _as_pdf_document(self, required=0)
4064: (8)                   if not pdf.m_internal:
4065: (12)                      return
4066: (8)                   lang = mupdf.pdf_document_language(pdf)
4067: (8)                   if lang == mupdf.FZ_LANG_UNSET:
4068: (12)                      return
4069: (8)                   if mupdf_version_tuple < (1, 23, 7):
4070: (12)                      assert 0, 'not implemented yet'
4071: (8)                   return mupdf.fz_string_from_text_language2(lang)
4072: (4)               @property
4073: (4)               def last_location(self):
4074: (8)                   """Id (chapter, page) of last page."""
4075: (8)                   if self.is_closed:
4076: (12)                      raise ValueError("document closed")
4077: (8)                   last_loc = mupdf.fz_last_page(self.this)
4078: (8)                   return last_loc.chapter, last_loc.page
4079: (4)               def layer_ui_configs(self):
4080: (8)                   """Show OC visibility status modifiable by user."""
4081: (8)                   pdf = _as_pdf_document(self)
4082: (8)                   info = mupdf.PdfLayerConfigUi()
4083: (8)                   n = mupdf.pdf_count_layer_config_ui( pdf)
4084: (8)                   rc = []
4085: (8)                   for i in range(n):
4086: (12)                      mupdf.pdf_layer_config_ui_info( pdf, i, info)
4087: (12)                      if info.type == 1:
4088: (16)                          type_ = "checkbox"
4089: (12)                      elif info.type == 2:
4090: (16)                          type_ = "radiobox"
4091: (12)                      else:
4092: (16)                          type_ = "label"
4093: (12)                      item = {
4094: (20)                              "number": i,
4095: (20)                              "text": info.text,
4096: (20)                              "depth": info.depth,
4097: (20)                              "type": type_,
4098: (20)                              "on": info.selected,
4099: (20)                              "locked": info.locked,
4100: (20)                              }
4101: (12)                      rc.append(item)
4102: (8)                   return rc
4103: (4)               def layout(self, rect=None, width=0, height=0, fontsize=11):
4104: (8)                   """Re-layout a reflowable document."""
4105: (8)                   if self.is_closed or self.is_encrypted:
4106: (12)                      raise ValueError("document closed or encrypted")
4107: (8)                   doc = self.this
4108: (8)                   if not mupdf.fz_is_document_reflowable( doc):
4109: (12)                      return
4110: (8)                   w = width
4111: (8)                   h = height
4112: (8)                   r = JM_rect_from_py(rect)
4113: (8)                   if not mupdf.fz_is_infinite_rect(r):
4114: (12)                      w = r.x1 - r.x0
```

```
4115: (12)                      h = r.y1 - r.y0
4116: (8)                   if w <= 0.0 or h <= 0.0:
4117: (12)                      raise ValueError( "bad page size")
4118: (8)                   mupdf.fz_layout_document( doc, w, h, fontsize)
4119: (8)                   self._reset_page_refs()
4120: (8)                   self.init_doc()
4121: (4)               def load_page(self, page_id):
4122: (8)                   """Load a page.
4123: (8)                   'page_id' is either a 0-based page number or a tuple (chapter, pno),
4124: (8)                   with chapter number and page number within that chapter.
4125: (8)                   """
4126: (8)                   if self.is_closed or self.is_encrypted:
4127: (12)                      raise ValueError("document closed or encrypted")
4128: (8)                   if page_id is None:
4129: (12)                      page_id = 0
4130: (8)                   if page_id not in self:
4131: (12)                      raise ValueError("page not in document")
4132: (8)                   if type(page_id) is int and page_id < 0:
4133: (12)                      np = self.page_count
4134: (12)                      while page_id < 0:
4135: (16)                          page_id += np
4136: (8)                   if isinstance(page_id, int):
4137: (12)                      page = mupdf.fz_load_page(self.this, page_id)
4138: (8)                   else:
4139: (12)                      chapter, pagenum = page_id
4140: (12)                      page = mupdf.fz_load_chapter_page(self.this, chapter, pagenum)
4141: (8)                   val = Page(page, self)
4142: (8)                   val.thisown = True
4143: (8)                   val.parent = self
4144: (8)                   self._page_refs[id(val)] = val
4145: (8)                   val._annot_refs = weakref.WeakValueDictionary()
4146: (8)                   val.number = page_id
4147: (8)                   return val
4148: (4)               def location_from_page_number(self, pno):
4149: (8)                   """Convert pno to (chapter, page)."""
4150: (8)                   if self.is_closed:
4151: (12)                      raise ValueError("document closed")
4152: (8)                   this_doc = self.this
4153: (8)                   loc = mupdf.fz_make_location(-1, -1)
4154: (8)                   page_count = mupdf.fz_count_pages(this_doc)
4155: (8)                   while pno < 0:
4156: (12)                      pno += page_count
4157: (8)                   if pno >= page_count:
4158: (12)                      raise ValueError( MSG_BAD_PAGENO)
4159: (8)                   loc = mupdf.fz_location_from_page_number(this_doc, pno)
4160: (8)                   return loc.chapter, loc.page
4161: (4)               def make_bookmark(self, loc):
4162: (8)                   """Make a page pointer before layouting document."""
4163: (8)                   if self.is_closed or self.is_encrypted:
4164: (12)                      raise ValueError("document closed or encrypted")
4165: (8)                   loc = mupdf.FzLocation(*loc)
4166: (8)                   mark = mupdf.ll_fz_make_bookmark2( self.this.m_internal,
loc.internal())
4167: (8)                   return mark
4168: (4)               @property
4169: (4)               def markinfo(self) -> dict:
4170: (8)                   """Return the PDF MarkInfo value."""
4171: (8)                   xref = self.pdf_catalog()
4172: (8)                   if xref == 0:
4173: (12)                      return None
4174: (8)                   rc = self.xref_get_key(xref, "MarkInfo")
4175: (8)                   if rc[0] == "null":
4176: (12)                      return {}
4177: (8)                   if rc[0] == "xref":
4178: (12)                      xref = int(rc[1].split()[0])
4179: (12)                      val = self.xref_object(xref, compressed=True)
4180: (8)                   elif rc[0] == "dict":
4181: (12)                      val = rc[1]
4182: (8)                   else:
```

```
4183: (12)                      val = None
4184: (8)                   if val is None or not (val[:2] == "<<" and val[-2:] == ">>"):
4185: (12)                      return {}
4186: (8)                   valid = {"Marked": False, "UserProperties": False, "Suspects": False}
4187: (8)                   val = val[2:-2].split("/")
4188: (8)                   for v in val[1:]:
4189: (12)                      try:
4190: (16)                          key, value = v.split()
4191: (12)                      except Exception:
4192: (16)                          if g_exceptions_verbose > 1:    exception_info()
4193: (16)                          return valid
4194: (12)                      if value == "true":
4195: (16)                          valid[key] = True
4196: (8)                   return valid
4197: (4)               def move_page(self, pno: int, to: int =-1):
4198: (8)                   """Move a page within a PDF document.
4199: (8)                   Args:
4200: (12)                      pno: source page number.
4201: (12)                      to: put before this page, '-1' means after last page.
4202: (8)                   """
4203: (8)                   if self.is_closed:
4204: (12)                      raise ValueError("document closed")
4205: (8)                   page_count = len(self)
4206: (8)                   if (pno not in range(page_count) or to not in range(-1, page_count)):
4207: (12)                      raise ValueError("bad page number(s)")
4208: (8)                   before = 1
4209: (8)                   copy = 0
4210: (8)                   if to == -1:
4211: (12)                      to = page_count - 1
4212: (12)                      before = 0
4213: (8)                   return self._move_copy_page(pno, to, before, copy)
4214: (4)               @property
4215: (4)               def name(self):
4216: (8)                   return self._name
4217: (4)               def need_appearances(self, value=None):
4218: (8)                   """Get/set the NeedAppearances value."""
4219: (8)                   if not self.is_form_pdf:
4220: (12)                      return None
4221: (8)                   pdf = _as_pdf_document(self)
4222: (8)                   oldval = -1
4223: (8)                   appkey = "NeedAppearances"
4224: (8)                   form = mupdf.pdf_dict_getp(
4225: (16)                          mupdf.pdf_trailer(pdf),
4226: (16)                          "Root/AcroForm",
4227: (16)                          )
4228: (8)                   app = mupdf.pdf_dict_gets(form, appkey)
4229: (8)                   if mupdf.pdf_is_bool(app):
4230: (12)                      oldval = mupdf.pdf_to_bool(app)
4231: (8)                   if value:
4232: (12)                      mupdf.pdf_dict_puts(form, appkey, mupdf.PDF_TRUE)
4233: (8)                   else:
4234: (12)                      mupdf.pdf_dict_puts(form, appkey, mupdf.PDF_FALSE)
4235: (8)                   if value is None:
4236: (12)                      return oldval >= 0
4237: (8)                   return value
4238: (4)               @property
4239: (4)               def needs_pass(self):
4240: (8)                   """Indicate password required."""
4241: (8)                   if self.is_closed:
4242: (12)                      raise ValueError("document closed")
4243: (8)                   document = self.this if isinstance(self.this, mupdf.FzDocument) else
self.this.super()
4244: (8)                   ret = mupdf.fz_needs_password( document)
4245: (8)                   return ret
4246: (4)               def next_location(self, page_id):
4247: (8)                   """Get (chapter, page) of next page."""
4248: (8)                   if self.is_closed or self.is_encrypted:
4249: (12)                      raise ValueError("document closed or encrypted")
4250: (8)                   if type(page_id) is int:
```

```
(12)                        page_id = (0, page_id)
(8)                     if page_id not in self:
(12)                        raise ValueError("page id not in document")
(8)                     if tuple(page_id) == self.last_location:
(12)                        return ()
(8)                     this_doc = _as_fz_document(self)
(8)                     val = page_id[ 0]
(8)                     if not isinstance(val, int):
(12)                        RAISEPY(MSG_BAD_PAGEID, PyExc_ValueError)
(8)                     chapter = val
(8)                     val = page_id[ 1]
(8)                     pno = val
(8)                     loc = mupdf.fz_make_location(chapter, pno)
(8)                     next_loc = mupdf.fz_next_page( this_doc, loc)
(8)                     return next_loc.chapter, next_loc.page
(4)                 def page_annot_xrefs(self, n):
(8)                     if g_use_extra:
(12)                        return extra.page_annot_xrefs( self.this, n)
(8)                     if isinstance(self.this, mupdf.PdfDocument):
(12)                        page_count = mupdf.pdf_count_pages(self.this)
(12)                        pdf_document = self.this
(8)                     else:
(12)                        page_count = mupdf.fz_count_pages(self.this)
(12)                        pdf_document = _as_pdf_document(self)
(8)                     while n < 0:
(12)                        n += page_count
(8)                     if n > page_count:
(12)                        raise ValueError( MSG_BAD_PAGENO)
(8)                     page_obj = mupdf.pdf_lookup_page_obj(pdf_document, n)
(8)                     annots = JM_get_annot_xref_list(page_obj)
(8)                     return annots
(4)                 @property
(4)                 def page_count(self):
(8)                     """Number of pages."""
(8)                     if self.is_closed:
(12)                        raise ValueError('document closed')
(8)                     if g_use_extra:
(12)                        return self.page_count2(self)
(8)                     if isinstance( self.this, mupdf.FzDocument):
(12)                        return mupdf.fz_count_pages( self.this)
(8)                     else:
(12)                        return mupdf.pdf_count_pages( self.this)
(4)                 def page_cropbox(self, pno):
(8)                     """Get CropBox of page number (without loading page)."""
(8)                     if self.is_closed:
(12)                        raise ValueError("document closed")
(8)                     this_doc = self.this
(8)                     page_count = mupdf.fz_count_pages( this_doc)
(8)                     n = pno
(8)                     while n < 0:
(12)                        n += page_count
(8)                     pdf = _as_pdf_document(self)
(8)                     if n >= page_count:
(12)                        raise ValueError( MSG_BAD_PAGENO)
(8)                     pageref = mupdf.pdf_lookup_page_obj( pdf, n)
(8)                     cropbox = JM_cropbox(pageref)
(8)                     val = JM_py_from_rect(cropbox)
(8)                     val = Rect(val)
(8)                     return val
(4)                 def page_number_from_location(self, page_id):
(8)                     """Convert (chapter, pno) to page number."""
(8)                     if type(page_id) is int:
(12)                        np = self.page_count
(12)                        while page_id < 0:
(16)                            page_id += np
(12)                        page_id = (0, page_id)
(8)                     if page_id not in self:
(12)                        raise ValueError("page id not in document")
(8)                     chapter, pno = page_id
```

```
4320: (8)                    loc = mupdf.fz_make_location( chapter, pno)
4321: (8)                    page_n = mupdf.fz_page_number_from_location( self.this, loc)
4322: (8)                    return page_n
4323: (4)                def page_xref(self, pno):
4324: (8)                    """Get xref of page number."""
4325: (8)                    if g_use_extra:
4326: (12)                       return extra.page_xref( self.this, pno)
4327: (8)                    if self.is_closed:
4328: (12)                       raise ValueError("document closed")
4329: (8)                    page_count = mupdf.fz_count_pages(self.this)
4330: (8)                    n = pno
4331: (8)                    while n < 0:
4332: (12)                       n += page_count
4333: (8)                    pdf = _as_pdf_document(self)
4334: (8)                    xref = 0
4335: (8)                    if n >= page_count:
4336: (12)                       raise ValueError( MSG_BAD_PAGENO)
4337: (8)                    xref = mupdf.pdf_to_num(mupdf.pdf_lookup_page_obj(pdf, n))
4338: (8)                    return xref
4339: (4)                @property
4340: (4)                def pagelayout(self) -> str:
4341: (8)                    """Return the PDF PageLayout value.
4342: (8)                    """
4343: (8)                    xref = self.pdf_catalog()
4344: (8)                    if xref == 0:
4345: (12)                       return None
4346: (8)                    rc = self.xref_get_key(xref, "PageLayout")
4347: (8)                    if rc[0] == "null":
4348: (12)                       return "SinglePage"
4349: (8)                    if rc[0] == "name":
4350: (12)                       return rc[1][1:]
4351: (8)                    return "SinglePage"
4352: (4)                @property
4353: (4)                def pagemode(self) -> str:
4354: (8)                    """Return the PDF PageMode value.
4355: (8)                    """
4356: (8)                    xref = self.pdf_catalog()
4357: (8)                    if xref == 0:
4358: (12)                       return None
4359: (8)                    rc = self.xref_get_key(xref, "PageMode")
4360: (8)                    if rc[0] == "null":
4361: (12)                       return "UseNone"
4362: (8)                    if rc[0] == "name":
4363: (12)                       return rc[1][1:]
4364: (8)                    return "UseNone"
4365: (4)                if sys.implementation.version < (3, 9):
4366: (8)                    _pages_ret = collections.abc.Iterable
4367: (4)                else:
4368: (8)                    _pages_ret = collections.abc.Iterable[Page]
4369: (4)                def pages(self, start: OptInt =None, stop: OptInt =None, step: OptInt
=None) -> _pages_ret:
4370: (8)                    """Return a generator iterator over a page range.
4371: (8)                    Arguments have the same meaning as for the range() built-in.
4372: (8)                    """
4373: (8)                    start = start or 0
4374: (8)                    while start < 0:
4375: (12)                       start += self.page_count
4376: (8)                    if start not in range(self.page_count):
4377: (12)                       raise ValueError("bad start page number")
4378: (8)                    stop = stop if stop is not None and stop <= self.page_count else
self.page_count
4379: (8)                    if step == 0:
4380: (12)                       raise ValueError("arg 3 must not be zero")
4381: (8)                    if step is None:
4382: (12)                       if start > stop:
4383: (16)                           step = -1
4384: (12)                       else:
4385: (16)                           step = 1
4386: (8)                    for pno in range(start, stop, step):
```

```
4387: (12)                      yield (self.load_page(pno))
4388: (4)               def pdf_catalog(self):
4389: (8)                   """Get xref of PDF catalog."""
4390: (8)                   pdf = _as_pdf_document(self, required=0)
4391: (8)                   xref = 0
4392: (8)                   if not pdf.m_internal:
4393: (12)                      return xref
4394: (8)                   root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
4395: (8)                   xref = mupdf.pdf_to_num(root)
4396: (8)                   return xref
4397: (4)               def pdf_trailer(self, compressed=0, ascii=0):
4398: (8)                   """Get PDF trailer as a string."""
4399: (8)                   return self.xref_object(-1, compressed=compressed, ascii=ascii)
4400: (4)               @property
4401: (4)               def permissions(self):
4402: (8)                   """Document permissions."""
4403: (8)                   if self.is_encrypted:
4404: (12)                      return 0
4405: (8)                   doc =self.this
4406: (8)                   pdf = mupdf.pdf_document_from_fz_document(doc)
4407: (8)                   if pdf.m_internal:
4408: (12)                      return mupdf.pdf_document_permissions(pdf)
4409: (8)                   perm = 0xFFFFFFFC   # all permissions granted
4410: (8)                   if not mupdf.fz_has_permission(doc, mupdf.FZ_PERMISSION_PRINT):
4411: (12)                      perm = perm ^ mupdf.PDF_PERM_PRINT
4412: (8)                   if not mupdf.fz_has_permission(doc, mupdf.FZ_PERMISSION_EDIT):
4413: (12)                      perm = perm ^ mupdf.PDF_PERM_MODIFY
4414: (8)                   if not mupdf.fz_has_permission(doc, mupdf.FZ_PERMISSION_COPY):
4415: (12)                      perm = perm ^ mupdf.PDF_PERM_COPY
4416: (8)                   if not mupdf.fz_has_permission(doc, mupdf.FZ_PERMISSION_ANNOTATE):
4417: (12)                      perm = perm ^ mupdf.PDF_PERM_ANNOTATE
4418: (8)                   return perm
4419: (4)               def prev_location(self, page_id):
4420: (8)                   """Get (chapter, page) of previous page."""
4421: (8)                   if self.is_closed or self.is_encrypted:
4422: (12)                      raise ValueError("document closed or encrypted")
4423: (8)                   if type(page_id) is int:
4424: (12)                      page_id = (0, page_id)
4425: (8)                   if page_id not in self:
4426: (12)                      raise ValueError("page id not in document")
4427: (8)                   if page_id  == (0, 0):
4428: (12)                      return ()
4429: (8)                   chapter, pno = page_id
4430: (8)                   loc = mupdf.fz_make_location(chapter, pno)
4431: (8)                   prev_loc = mupdf.fz_previous_page(self.this, loc)
4432: (8)                   return prev_loc.chapter, prev_loc.page
4433: (4)               def reload_page(self, page: Page) -> Page:
4434: (8)                   """Make a fresh copy of a page."""
4435: (8)                   old_annots = {}  # copy annot references to here
4436: (8)                   pno = page.number  # save the page number
4437: (8)                   for k, v in page._annot_refs.items():  # save the annot dictionary
4438: (12)                      old_annots[k] = v
4439: (8)                   refs_old = page.this.m_internal.refs
4440: (8)                   m_internal_old = page.this.m_internal_value()
4441: (8)                   page.this = None
4442: (8)                   page._erase()  # remove the page
4443: (8)                   page = None
4444: (8)                   TOOLS.store_shrink(100)
4445: (8)                   page = self.load_page(pno)  # reload the page
4446: (8)                   for k, v in old_annots.items():
4447: (12)                      annot = old_annots[k]
4448: (12)                      page._annot_refs[k] = annot
4449: (8)                   if refs_old == 1:
4450: (12)                      pass
4451: (8)                   else:
4452: (12)                      m_internal_new = page.this.m_internal_value()
4453: (12)                      assert m_internal_new != m_internal_old, \
4454: (20)                              f'{refs_old=} {m_internal_old=:#x} {m_internal_new=:#x}'
4455: (8)                   return page
```

```
4456: (4)               def resolve_link(self, uri=None, chapters=0):
4457: (8)                   """Calculate internal link destination.
4458: (8)                   Args:
4459: (12)                      uri: (str) some Link.uri
4460: (12)                      chapters: (bool) whether to use (chapter, page) format
4461: (8)                   Returns:
4462: (12)                      (page_id, x, y) where x, y are point coordinates on the page.
4463: (12)                      page_id is either page number (if chapters=0), or (chapter, pno).
4464: (8)                   """
4465: (8)                   if not uri:
4466: (12)                      if chapters:
4467: (16)                          return (-1, -1), 0, 0
4468: (12)                      return -1, 0, 0
4469: (8)                   try:
4470: (12)                      loc, xp, yp = mupdf.fz_resolve_link(self.this, uri)
4471: (8)                   except Exception:
4472: (12)                      if g_exceptions_verbose:    exception_info()
4473: (12)                      if chapters:
4474: (16)                          return (-1, -1), 0, 0
4475: (12)                      return -1, 0, 0
4476: (8)                   if chapters:
4477: (12)                      return (loc.chapter, loc.page), xp, yp
4478: (8)                   pno = mupdf.fz_page_number_from_location(self.this, loc)
4479: (8)                   return pno, xp, yp
4480: (4)               def resolve_names(self):
4481: (8)                   """Convert the PDF's destination names into a Python dict.
4482: (8)                   The only parameter is the pymupdf.Document.
4483: (8)                   All names found in the catalog under keys "/Dests" and "/Names/Dests"
are
4484: (8)                   being included.
4485: (8)                   Returns:
4486: (12)                      A dcitionary with the following layout:
4487: (12)                      - key: (str) the name
4488: (12)                      - value: (dict) with the following layout:
4489: (16)                          * "page":  target page number (0-based). If no page number
found -1.
4490: (16)                          * "to": (x, y) target point on page - currently in PDF
coordinates,
4491: (24)                                  i.e. point (0,0) is the bottom-left of the page.
4492: (16)                          * "zoom": (float) the zoom factor
4493: (16)                          * "dest": (str) only occurs if the target location on the page
has
4494: (24)                                  not been provided as "/XYZ" or if no page number was
found.
4495: (12)                      Examples:
4496: (12)                      {'__bookmark_1': {'page': 0, 'to': (0.0, 541.0), 'zoom': 0.0},
4497: (12)                      '__bookmark_2': {'page': 0, 'to': (0.0, 481.45), 'zoom': 0.0}}
4498: (12)                      or
4499: (12)                      '21154a7c20684ceb91f9c9adc3b677c40': {'page': -1, 'dest': '/XYZ
15.75 1486 0'}, ...
4500: (8)                   """
4501: (8)                   if hasattr(self, "_resolved_names"):  # do not execute multiple times!
4502: (12)                      return self._resolved_names
4503: (8)                   page_xrefs = {self.page_xref(i): i for i in range(self.page_count)}
4504: (8)                   def obj_string(obj):
4505: (12)                      """Return string version of a PDF object definition."""
4506: (12)                      buffer = mupdf.fz_new_buffer(512)
4507: (12)                      output = mupdf.FzOutput(buffer)
4508: (12)                      mupdf.pdf_print_obj(output, obj, 1, 0)
4509: (12)                      output.fz_close_output()
4510: (12)                      return JM_UnicodeFromBuffer(buffer)
4511: (8)                   def get_array(val):
4512: (12)                      """Generate value of one item of the names dictionary."""
4513: (12)                      templ_dict = {"page": -1, "dest": ""}  # value template
4514: (12)                      if val.pdf_is_indirect():
4515: (16)                          val = mupdf.pdf_resolve_indirect(val)
4516: (12)                      if val.pdf_is_array():
4517: (16)                          array = obj_string(val)
4518: (12)                      elif val.pdf_is_dict():
```

```
4519: (16)                          array = obj_string(mupdf.pdf_dict_gets(val, "D"))
4520: (12)                      else:  # if all fails return the empty template
4521: (16)                          return templ_dict
4522: (12)                      array = array.replace("null", "0")[1:-1]
4523: (12)                      idx = array.find("/")
4524: (12)                      if idx < 1:  # this has no target page spec
4525: (16)                          templ_dict["dest"] = array  # return the orig. string
4526: (16)                          return templ_dict
4527: (12)                      subval = array[:idx]  # stuff before "/"
4528: (12)                      array = array[idx:]  # stuff from "/" onwards
4529: (12)                      templ_dict["dest"] = array
4530: (12)                      if array.startswith("/XYZ"):
4531: (16)                          del templ_dict["dest"]  # don't return orig string in this
case
4532: (16)                          t = [0, 0, 0]  # the resulting x, y, z values
4533: (16)                          for i, v in enumerate(array.replace("null", "0").split()[1:]):
4534: (20)                              t[i] = float(v)
4535: (16)                          templ_dict["to"] = (t[0], t[1])
4536: (16)                          templ_dict["zoom"] = t[2]
4537: (12)                      if "0 R" in subval:  # page xref given?
4538: (16)                          templ_dict["page"] = page_xrefs.get(int(subval.split()[0]),-1)
4539: (12)                      else:  # naked page number given
4540: (16)                          templ_dict["page"] = int(subval)
4541: (12)                      return templ_dict
4542: (8)                   def fill_dict(dest_dict, pdf_dict):
4543: (12)                      """Generate name resolution items for pdf_dict.
4544: (12)                      This may be either "/Names/Dests" or just "/Dests"
4545: (12)                      """
4546: (12)                      name_count = mupdf.pdf_dict_len(pdf_dict)
4547: (12)                      for i in range(name_count):
4548: (16)                          key = mupdf.pdf_dict_get_key(pdf_dict, i)
4549: (16)                          val = mupdf.pdf_dict_get_val(pdf_dict, i)
4550: (16)                          if key.pdf_is_name():  # this should always be true!
4551: (20)                              dict_key = key.pdf_to_name()
4552: (16)                          else:
4553: (20)                              message(f"key {i} is no /Name")
4554: (20)                              dict_key = None
4555: (16)                          if dict_key:
4556: (20)                              dest_dict[dict_key] = get_array(val)  # store key/value in
dict
4557: (8)                   pdf = mupdf.pdf_document_from_fz_document(self)
4558: (8)                   catalog = mupdf.pdf_dict_gets(mupdf.pdf_trailer(pdf), "Root")
4559: (8)                   dest_dict = {}
4560: (8)                   dests = mupdf.pdf_new_name("Dests")
4561: (8)                   old_dests = mupdf.pdf_dict_get(catalog, dests)
4562: (8)                   if old_dests.pdf_is_dict():
4563: (12)                      fill_dict(dest_dict, old_dests)
4564: (8)                   tree = mupdf.pdf_load_name_tree(pdf, dests)
4565: (8)                   if tree.pdf_is_dict():
4566: (12)                      fill_dict(dest_dict, tree)
4567: (8)                   self._resolved_names = dest_dict  # store result or reuse
4568: (8)                   return dest_dict
4569: (4)               def save(
4570: (12)                      self,
4571: (12)                      filename,
4572: (12)                      garbage=0,
4573: (12)                      clean=0,
4574: (12)                      deflate=0,
4575: (12)                      deflate_images=0,
4576: (12)                      deflate_fonts=0,
4577: (12)                      incremental=0,
4578: (12)                      ascii=0,
4579: (12)                      expand=0,
4580: (12)                      linear=0,
4581: (12)                      no_new_id=0,
4582: (12)                      appearance=0,
4583: (12)                      pretty=0,
4584: (12)                      encryption=1,
4585: (12)                      permissions=4095,
```

```
4586: (12)                      owner_pw=None,
4587: (12)                      user_pw=None,
4588: (12)                      preserve_metadata=1,
4589: (12)                      use_objstms=0,
4590: (12)                      compression_effort=0,
4591: (12)                      ):
4592: (8)                   """Save PDF to file, pathlib.Path or file pointer."""
4593: (8)                   if self.is_closed or self.is_encrypted:
4594: (12)                      raise ValueError("document closed or encrypted")
4595: (8)                   if type(filename) is str:
4596: (12)                      pass
4597: (8)                   elif hasattr(filename, "open"):  # assume: pathlib.Path
4598: (12)                      filename = str(filename)
4599: (8)                   elif hasattr(filename, "name"):  # assume: file object
4600: (12)                      filename = filename.name
4601: (8)                   elif not hasattr(filename, "seek"):  # assume file object
4602: (12)                      raise ValueError("filename must be str, Path or file object")
4603: (8)                   if filename == self.name and not incremental:
4604: (12)                      raise ValueError("save to original must be incremental")
4605: (8)                   if linear and use_objstms:
4606: (12)                      raise ValueError("'linear' and 'use_objstms' cannot both be
requested")
4607: (8)                   if self.page_count < 1:
4608: (12)                      raise ValueError("cannot save with zero pages")
4609: (8)                   if incremental:
4610: (12)                      if self.name != filename or self.stream:
4611: (16)                          raise ValueError("incremental needs original file")
4612: (8)                   if user_pw and len(user_pw) > 40 or owner_pw and len(owner_pw) > 40:
4613: (12)                      raise ValueError("password length must not exceed 40")
4614: (8)                   pdf = _as_pdf_document(self)
4615: (8)                   opts = mupdf.PdfWriteOptions()
4616: (8)                   opts.do_incremental = incremental
4617: (8)                   opts.do_ascii = ascii
4618: (8)                   opts.do_compress = deflate
4619: (8)                   opts.do_compress_images = deflate_images
4620: (8)                   opts.do_compress_fonts = deflate_fonts
4621: (8)                   opts.do_decompress = expand
4622: (8)                   opts.do_garbage = garbage
4623: (8)                   opts.do_pretty = pretty
4624: (8)                   opts.do_linear = linear
4625: (8)                   opts.do_clean = clean
4626: (8)                   opts.do_sanitize = clean
4627: (8)                   opts.dont_regenerate_id = no_new_id
4628: (8)                   opts.do_appearance = appearance
4629: (8)                   opts.do_encrypt = encryption
4630: (8)                   opts.permissions = permissions
4631: (8)                   if owner_pw is not None:
4632: (12)                      opts.opwd_utf8_set_value(owner_pw)
4633: (8)                   elif user_pw is not None:
4634: (12)                      opts.opwd_utf8_set_value(user_pw)
4635: (8)                   if user_pw is not None:
4636: (12)                      opts.upwd_utf8_set_value(user_pw)
4637: (8)                   opts.do_preserve_metadata = preserve_metadata
4638: (8)                   opts.do_use_objstms = use_objstms
4639: (8)                   opts.compression_effort = compression_effort
4640: (8)                   out = None
4641: (8)                   pdf.m_internal.resynth_required = 0
4642: (8)                   JM_embedded_clean(pdf)
4643: (8)                   if no_new_id == 0:
4644: (12)                      JM_ensure_identity(pdf)
4645: (8)                   if isinstance(filename, str):
4646: (12)                      mupdf.pdf_save_document(pdf, filename, opts)
4647: (8)                   else:
4648: (12)                      out = JM_new_output_fileptr(filename)
4649: (12)                      mupdf.pdf_write_document(pdf, out, opts)
4650: (12)                      out.fz_close_output()
4651: (4)               def save_snapshot(self, filename):
4652: (8)                   """Save a file snapshot suitable for journalling."""
4653: (8)                   if self.is_closed:
```

```
4654: (12)                      raise ValueError("doc is closed")
4655: (8)                   if type(filename) is str:
4656: (12)                      pass
4657: (8)                   elif hasattr(filename, "open"):  # assume: pathlib.Path
4658: (12)                      filename = str(filename)
4659: (8)                   elif hasattr(filename, "name"):  # assume: file object
4660: (12)                      filename = filename.name
4661: (8)                   else:
4662: (12)                      raise ValueError("filename must be str, Path or file object")
4663: (8)                   if filename == self.name:
4664: (12)                      raise ValueError("cannot snapshot to original")
4665: (8)                   pdf = _as_pdf_document(self)
4666: (8)                   mupdf.pdf_save_snapshot(pdf, filename)
4667: (4)               def saveIncr(self):
4668: (8)                   """ Save PDF incrementally"""
4669: (8)                   return self.save(self.name, incremental=True,
encryption=mupdf.PDF_ENCRYPT_KEEP)
4670: (4)               def select(self, pyliste):
4671: (8)                   """Build sub-pdf with page numbers in the list."""
4672: (8)                   if self.is_closed or self.is_encrypted:
4673: (12)                      raise ValueError("document closed or encrypted")
4674: (8)                   if not self.is_pdf:
4675: (12)                      raise ValueError("is no PDF")
4676: (8)                   if not hasattr(pyliste, "__getitem__"):
4677: (12)                      raise ValueError("sequence required")
4678: (8)                   valid_range = range(len(self))
4679: (8)                   if (len(pyliste) == 0
4680: (12)                      or min(pyliste) not in valid_range
4681: (12)                      or max(pyliste) not in valid_range
4682: (8)                   ):
4683: (12)                      raise ValueError("bad page number(s)")
4684: (8)                   pdf = _as_pdf_document(self)
4685: (8)                   if mupdf_version_tuple >= (1, 24):
4686: (12)                      mupdf.pdf_rearrange_pages2(pdf, pyliste)
4687: (8)                   else:
4688: (12)                      extra.rearrange_pages2(pdf, tuple(pyliste))
4689: (8)                   self._reset_page_refs()
4690: (4)               def set_language(self, language=None):
4691: (8)                   pdf = _as_pdf_document(self)
4692: (8)                   if not language:
4693: (12)                      lang = mupdf.FZ_LANG_UNSET
4694: (8)                   else:
4695: (12)                      lang = mupdf.fz_text_language_from_string(language)
4696: (8)                   mupdf.pdf_set_document_language(pdf, lang)
4697: (8)                   return True
4698: (4)               def set_layer(self, config, basestate=None, on=None, off=None,
rbgroups=None, locked=None):
4699: (8)                   """Set the PDF keys /ON, /OFF, /RBGroups of an OC layer."""
4700: (8)                   if self.is_closed:
4701: (12)                      raise ValueError("document closed")
4702: (8)                   ocgs = set(self.get_ocgs().keys())
4703: (8)                   if ocgs == set():
4704: (12)                      raise ValueError("document has no optional content")
4705: (8)                   if on:
4706: (12)                      if type(on) not in (list, tuple):
4707: (16)                          raise ValueError("bad type: 'on'")
4708: (12)                      s = set(on).difference(ocgs)
4709: (12)                      if s != set():
4710: (16)                          raise ValueError("bad OCGs in 'on': %s" % s)
4711: (8)                   if off:
4712: (12)                      if type(off) not in (list, tuple):
4713: (16)                          raise ValueError("bad type: 'off'")
4714: (12)                      s = set(off).difference(ocgs)
4715: (12)                      if s != set():
4716: (16)                          raise ValueError("bad OCGs in 'off': %s" % s)
4717: (8)                   if locked:
4718: (12)                      if type(locked) not in (list, tuple):
4719: (16)                          raise ValueError("bad type: 'locked'")
4720: (12)                      s = set(locked).difference(ocgs)
```

```
4721: (12)                      if s != set():
4722: (16)                          raise ValueError("bad OCGs in 'locked': %s" % s)
4723: (8)                   if rbgroups:
4724: (12)                      if type(rbgroups) not in (list, tuple):
4725: (16)                          raise ValueError("bad type: 'rbgroups'")
4726: (12)                      for x in rbgroups:
4727: (16)                          if not type(x) in (list, tuple):
4728: (20)                              raise ValueError("bad RBGroup '%s'" % x)
4729: (16)                          s = set(x).difference(ocgs)
4730: (16)                          if s != set():
4731: (20)                              raise ValueError("bad OCGs in RBGroup: %s" % s)
4732: (8)                   if basestate:
4733: (12)                      basestate = str(basestate).upper()
4734: (12)                      if basestate == "UNCHANGED":
4735: (16)                          basestate = "Unchanged"
4736: (12)                      if basestate not in ("ON", "OFF", "Unchanged"):
4737: (16)                          raise ValueError("bad 'basestate'")
4738: (8)                   pdf = _as_pdf_document(self)
4739: (8)                   ocp = mupdf.pdf_dict_getl(
4740: (16)                          mupdf.pdf_trailer( pdf),
4741: (16)                          PDF_NAME('Root'),
4742: (16)                          PDF_NAME('OCProperties'),
4743: (16)                          )
4744: (8)                   if not ocp.m_internal:
4745: (12)                      return
4746: (8)                   if config == -1:
4747: (12)                      obj = mupdf.pdf_dict_get( ocp, PDF_NAME('D'))
4748: (8)                   else:
4749: (12)                      obj = mupdf.pdf_array_get(
4750: (20)                              mupdf.pdf_dict_get( ocp, PDF_NAME('Configs')),
4751: (20)                              config,
4752: (20)                              )
4753: (8)                   if not obj.m_internal:
4754: (12)                      raise ValueError( MSG_BAD_OC_CONFIG)
4755: (8)                   JM_set_ocg_arrays( obj, basestate, on, off, rbgroups, locked)
4756: (8)                   mupdf.ll_pdf_read_ocg( pdf.m_internal)
4757: (4)               def set_layer_ui_config(self, number, action=0):
4758: (8)                   """Set / unset OC intent configuration."""
4759: (8)                   if isinstance(number, str):
4760: (12)                      select = [ui["number"] for ui in self.layer_ui_configs() if
ui["text"] == number]
4761: (12)                      if select == []:
4762: (16)                          raise ValueError(f"bad OCG '{number}'.")
4763: (12)                      number = select[0]  # this is the number for the name
4764: (8)                   pdf = _as_pdf_document(self)
4765: (8)                   if action == 1:
4766: (12)                      mupdf.pdf_toggle_layer_config_ui(pdf, number)
4767: (8)                   elif action == 2:
4768: (12)                      mupdf.pdf_deselect_layer_config_ui(pdf, number)
4769: (8)                   else:
4770: (12)                      mupdf.pdf_select_layer_config_ui(pdf, number)
4771: (4)               def set_markinfo(self, markinfo: dict) -> bool:
4772: (8)                   """Set the PDF MarkInfo values."""
4773: (8)                   xref = self.pdf_catalog()
4774: (8)                   if xref == 0:
4775: (12)                      raise ValueError("not a PDF")
4776: (8)                   if not markinfo or not isinstance(markinfo, dict):
4777: (12)                      return False
4778: (8)                   valid = {"Marked": False, "UserProperties": False, "Suspects": False}
4779: (8)                   if not set(valid.keys()).issuperset(markinfo.keys()):
4780: (12)                      badkeys = f"bad MarkInfo key(s):
{set(markinfo.keys()).difference(valid.keys())}"
4781: (12)                      raise ValueError(badkeys)
4782: (8)                   pdfdict = "<<"
4783: (8)                   valid.update(markinfo)
4784: (8)                   for key, value in valid.items():
4785: (12)                      value=str(value).lower()
4786: (12)                      if value not in ("true", "false"):
4787: (16)                          raise ValueError(f"bad key value '{key}': '{value}'")
```

```
4788: (12)                      pdfdict += f"/{key} {value}"
4789: (8)                   pdfdict += ">>"
4790: (8)                   self.xref_set_key(xref, "MarkInfo", pdfdict)
4791: (8)                   return True
4792: (4)               def set_pagelayout(self, pagelayout: str):
4793: (8)                   """Set the PDF PageLayout value."""
4794: (8)                   valid = ("SinglePage", "OneColumn", "TwoColumnLeft", "TwoColumnRight",
"TwoPageLeft", "TwoPageRight")
4795: (8)                   xref = self.pdf_catalog()
4796: (8)                   if xref == 0:
4797: (12)                      raise ValueError("not a PDF")
4798: (8)                   if not pagelayout:
4799: (12)                      raise ValueError("bad PageLayout value")
4800: (8)                   if pagelayout[0] == "/":
4801: (12)                      pagelayout = pagelayout[1:]
4802: (8)                   for v in valid:
4803: (12)                      if pagelayout.lower() == v.lower():
4804: (16)                          self.xref_set_key(xref, "PageLayout", f"/{v}")
4805: (16)                          return True
4806: (8)                   raise ValueError("bad PageLayout value")
4807: (4)               def set_pagemode(self, pagemode: str):
4808: (8)                   """Set the PDF PageMode value."""
4809: (8)                   valid = ("UseNone", "UseOutlines", "UseThumbs", "FullScreen", "UseOC",
"UseAttachments")
4810: (8)                   xref = self.pdf_catalog()
4811: (8)                   if xref == 0:
4812: (12)                      raise ValueError("not a PDF")
4813: (8)                   if not pagemode:
4814: (12)                      raise ValueError("bad PageMode value")
4815: (8)                   if pagemode[0] == "/":
4816: (12)                      pagemode = pagemode[1:]
4817: (8)                   for v in valid:
4818: (12)                      if pagemode.lower() == v.lower():
4819: (16)                          self.xref_set_key(xref, "PageMode", f"/{v}")
4820: (16)                          return True
4821: (8)                   raise ValueError("bad PageMode value")
4822: (4)               def set_xml_metadata(self, metadata):
4823: (8)                   """Store XML document level metadata."""
4824: (8)                   if self.is_closed or self.is_encrypted:
4825: (12)                      raise ValueError("document closed or encrypted")
4826: (8)                   pdf = _as_pdf_document(self)
4827: (8)                   root = mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf), PDF_NAME('Root'))
4828: (8)                   if not root.m_internal:
4829: (12)                      RAISEPY( MSG_BAD_PDFROOT, JM_Exc_FileDataError)
4830: (8)                   res = mupdf.fz_new_buffer_from_copied_data( metadata.encode('utf-8'))
4831: (8)                   xml = mupdf.pdf_dict_get( root, PDF_NAME('Metadata'))
4832: (8)                   if xml.m_internal:
4833: (12)                      JM_update_stream( pdf, xml, res, 0)
4834: (8)                   else:
4835: (12)                      xml = mupdf.pdf_add_stream( pdf, res, mupdf.PdfObj(), 0)
4836: (12)                      mupdf.pdf_dict_put( xml, PDF_NAME('Type'), PDF_NAME('Metadata'))
4837: (12)                      mupdf.pdf_dict_put( xml, PDF_NAME('Subtype'), PDF_NAME('XML'))
4838: (12)                      mupdf.pdf_dict_put( root, PDF_NAME('Metadata'), xml)
4839: (4)               def switch_layer(self, config, as_default=0):
4840: (8)                   """Activate an OC layer."""
4841: (8)                   pdf = _as_pdf_document(self)
4842: (8)                   cfgs = mupdf.pdf_dict_getl(
4843: (16)                          mupdf.pdf_trailer( pdf),
4844: (16)                          PDF_NAME('Root'),
4845: (16)                          PDF_NAME('OCProperties'),
4846: (16)                          PDF_NAME('Configs')
4847: (16)                          )
4848: (8)                   if not mupdf.pdf_is_array( cfgs) or not mupdf.pdf_array_len( cfgs):
4849: (12)                      if config < 1:
4850: (16)                          return
4851: (12)                      raise ValueError( MSG_BAD_OC_LAYER)
4852: (8)                   if config < 0:
4853: (12)                      return
4854: (8)                   mupdf.pdf_select_layer_config( pdf, config)
```

```
4855: (8)                    if as_default:
4856: (12)                       mupdf.pdf_set_layer_config_as_default( pdf)
4857: (12)                       mupdf.ll_pdf_read_ocg( pdf.m_internal)
4858: (4)                def update_object(self, xref, text, page=None):
4859: (8)                    """Replace object definition source."""
4860: (8)                    if self.is_closed or self.is_encrypted:
4861: (12)                       raise ValueError("document closed or encrypted")
4862: (8)                    pdf = _as_pdf_document(self)
4863: (8)                    xreflen = mupdf.pdf_xref_len(pdf)
4864: (8)                    if not _INRANGE(xref, 1, xreflen-1):
4865: (12)                       RAISEPY("bad xref", MSG_BAD_XREF, PyExc_ValueError)
4866: (8)                    ENSURE_OPERATION(pdf)
4867: (8)                    new_obj = JM_pdf_obj_from_str(pdf, text)
4868: (8)                    mupdf.pdf_update_object(pdf, xref, new_obj)
4869: (8)                    if page:
4870: (12)                       JM_refresh_links( _as_pdf_page(page))
4871: (4)                def update_stream(self, xref=0, stream=None, new=1, compress=1):
4872: (8)                    """Replace xref stream part."""
4873: (8)                    if self.is_closed or self.is_encrypted:
4874: (12)                       raise ValueError("document closed or encrypted")
4875: (8)                    pdf = _as_pdf_document(self)
4876: (8)                    xreflen = mupdf.pdf_xref_len(pdf)
4877: (8)                    if xref < 1 or xref > xreflen:
4878: (12)                       raise ValueError( MSG_BAD_XREF)
4879: (8)                    obj = mupdf.pdf_new_indirect(pdf, xref, 0)
4880: (8)                    if not mupdf.pdf_is_dict(obj):
4881: (12)                       raise ValueError( MSG_IS_NO_DICT)
4882: (8)                    res = JM_BufferFromBytes(stream)
4883: (8)                    if not res.m_internal:
4884: (12)                       raise TypeError( MSG_BAD_BUFFER)
4885: (8)                    JM_update_stream(pdf, obj, res, compress)
4886: (8)                    pdf.dirty = 1
4887: (4)                @property
4888: (4)                def version_count(self):
4889: (8)                    '''
4890: (8)                    Count versions of PDF document.
4891: (8)                    '''
4892: (8)                    pdf = _as_pdf_document(self, required=0)
4893: (8)                    if pdf.m_internal:
4894: (12)                       return mupdf.pdf_count_versions(pdf)
4895: (8)                    return 0
4896: (4)                def write(
4897: (12)                       self,
4898: (12)                       garbage=False,
4899: (12)                       clean=False,
4900: (12)                       deflate=False,
4901: (12)                       deflate_images=False,
4902: (12)                       deflate_fonts=False,
4903: (12)                       incremental=False,
4904: (12)                       ascii=False,
4905: (12)                       expand=False,
4906: (12)                       linear=False,
4907: (12)                       no_new_id=False,
4908: (12)                       appearance=False,
4909: (12)                       pretty=False,
4910: (12)                       encryption=1,
4911: (12)                       permissions=4095,
4912: (12)                       owner_pw=None,
4913: (12)                       user_pw=None,
4914: (12)                       preserve_metadata=1,
4915: (12)                       use_objstms=0,
4916: (12)                       compression_effort=0,
4917: (4)                ):
4918: (8)                    from io import BytesIO
4919: (8)                    bio = BytesIO()
4920: (8)                    self.save(
4921: (16)                           bio,
4922: (16)                           garbage=garbage,
4923: (16)                           clean=clean,
```

```
4924: (16)                          no_new_id=no_new_id,
4925: (16)                          appearance=appearance,
4926: (16)                          deflate=deflate,
4927: (16)                          deflate_images=deflate_images,
4928: (16)                          deflate_fonts=deflate_fonts,
4929: (16)                          incremental=incremental,
4930: (16)                          ascii=ascii,
4931: (16)                          expand=expand,
4932: (16)                          linear=linear,
4933: (16)                          pretty=pretty,
4934: (16)                          encryption=encryption,
4935: (16)                          permissions=permissions,
4936: (16)                          owner_pw=owner_pw,
4937: (16)                          user_pw=user_pw,
4938: (16)                          preserve_metadata=preserve_metadata,
4939: (16)                          use_objstms=use_objstms,
4940: (16)                          compression_effort=compression_effort,
4941: (8)                   )
4942: (8)                   return bio.getvalue()
4943: (4)               @property
4944: (4)               def xref(self):
4945: (8)                   """PDF xref number of page."""
4946: (8)                   CheckParent(self)
4947: (8)                   return self.parent.page_xref(self.number)
4948: (4)               def xref_get_key(self, xref, key):
4949: (8)                   """Get PDF dict key value of object at 'xref'."""
4950: (8)                   pdf = _as_pdf_document(self)
4951: (8)                   xreflen = mupdf.pdf_xref_len(pdf)
4952: (8)                   if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
4953: (12)                      raise ValueError( MSG_BAD_XREF)
4954: (8)                   if xref > 0:
4955: (12)                      obj = mupdf.pdf_load_object(pdf, xref)
4956: (8)                   else:
4957: (12)                      obj = mupdf.pdf_trailer(pdf)
4958: (8)                   if not obj.m_internal:
4959: (12)                      return ("null", "null")
4960: (8)                   subobj = mupdf.pdf_dict_getp(obj, key)
4961: (8)                   if not subobj.m_internal:
4962: (12)                      return ("null", "null")
4963: (8)                   text = None
4964: (8)                   if mupdf.pdf_is_indirect(subobj):
4965: (12)                      type = "xref"
4966: (12)                      text = "%i 0 R" % mupdf.pdf_to_num(subobj)
4967: (8)                   elif mupdf.pdf_is_array(subobj):
4968: (12)                      type = "array"
4969: (8)                   elif mupdf.pdf_is_dict(subobj):
4970: (12)                      type = "dict"
4971: (8)                   elif mupdf.pdf_is_int(subobj):
4972: (12)                      type = "int"
4973: (12)                      text = "%i" % mupdf.pdf_to_int(subobj)
4974: (8)                   elif mupdf.pdf_is_real(subobj):
4975: (12)                      type = "float"
4976: (8)                   elif mupdf.pdf_is_null(subobj):
4977: (12)                      type = "null"
4978: (12)                      text = "null"
4979: (8)                   elif mupdf.pdf_is_bool(subobj):
4980: (12)                      type = "bool"
4981: (12)                      if mupdf.pdf_to_bool(subobj):
4982: (16)                          text = "true"
4983: (12)                      else:
4984: (16)                          text = "false"
4985: (8)                   elif mupdf.pdf_is_name(subobj):
4986: (12)                      type = "name"
4987: (12)                      text = "/%s" % mupdf.pdf_to_name(subobj)
4988: (8)                   elif mupdf.pdf_is_string(subobj):
4989: (12)                      type = "string"
4990: (12)                      text = JM_UnicodeFromStr(mupdf.pdf_to_text_string(subobj))
4991: (8)                   else:
4992: (12)                      type = "unknown"
```

```
4993: (8)                    if text is None:
4994: (12)                       res = JM_object_to_buffer(subobj, 1, 0)
4995: (12)                       text = JM_UnicodeFromBuffer(res)
4996: (8)                    return (type, text)
4997: (4)                def xref_get_keys(self, xref):
4998: (8)                    """Get the keys of PDF dict object at 'xref'. Use -1 for the PDF
trailer."""
4999: (8)                    pdf = _as_pdf_document(self)
5000: (8)                    xreflen = mupdf.pdf_xref_len( pdf)
5001: (8)                    if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
5002: (12)                       raise ValueError( MSG_BAD_XREF)
5003: (8)                    if xref > 0:
5004: (12)                       obj = mupdf.pdf_load_object( pdf, xref)
5005: (8)                    else:
5006: (12)                       obj = mupdf.pdf_trailer( pdf)
5007: (8)                    n = mupdf.pdf_dict_len( obj)
5008: (8)                    rc = []
5009: (8)                    if n == 0:
5010: (12)                       return rc
5011: (8)                    for i in range(n):
5012: (12)                       key = mupdf.pdf_to_name( mupdf.pdf_dict_get_key( obj, i))
5013: (12)                       rc.append(key)
5014: (8)                    return rc
5015: (4)                def xref_is_font(self, xref):
5016: (8)                    """Check if xref is a font object."""
5017: (8)                    if self.is_closed or self.is_encrypted:
5018: (12)                       raise ValueError("document closed or encrypted")
5019: (8)                    if self.xref_get_key(xref, "Type")[1] == "/Font":
5020: (12)                       return True
5021: (8)                    return False
5022: (4)                def xref_is_image(self, xref):
5023: (8)                    """Check if xref is an image object."""
5024: (8)                    if self.is_closed or self.is_encrypted:
5025: (12)                       raise ValueError("document closed or encrypted")
5026: (8)                    if self.xref_get_key(xref, "Subtype")[1] == "/Image":
5027: (12)                       return True
5028: (8)                    return False
5029: (4)                def xref_is_stream(self, xref=0):
5030: (8)                    """Check if xref is a stream object."""
5031: (8)                    pdf = _as_pdf_document(self, required=0)
5032: (8)                    if not pdf.m_internal:
5033: (12)                       return False    # not a PDF
5034: (8)                    return bool(mupdf.pdf_obj_num_is_stream(pdf, xref))
5035: (4)                def xref_is_xobject(self, xref):
5036: (8)                    """Check if xref is a form xobject."""
5037: (8)                    if self.is_closed or self.is_encrypted:
5038: (12)                       raise ValueError("document closed or encrypted")
5039: (8)                    if self.xref_get_key(xref, "Subtype")[1] == "/Form":
5040: (12)                       return True
5041: (8)                    return False
5042: (4)                def xref_length(self):
5043: (8)                    """Get length of xref table."""
5044: (8)                    xreflen = 0
5045: (8)                    pdf = _as_pdf_document(self, required=0)
5046: (8)                    if pdf.m_internal:
5047: (12)                       xreflen = mupdf.pdf_xref_len(pdf)
5048: (8)                    return xreflen
5049: (4)                def xref_object(self, xref, compressed=0, ascii=0):
5050: (8)                    """Get xref object source as a string."""
5051: (8)                    if self.is_closed:
5052: (12)                       raise ValueError("document closed")
5053: (8)                    if g_use_extra:
5054: (12)                       ret = extra.xref_object( self.this, xref, compressed, ascii)
5055: (12)                       return ret
5056: (8)                    pdf = _as_pdf_document(self)
5057: (8)                    xreflen = mupdf.pdf_xref_len(pdf)
5058: (8)                    if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
5059: (12)                       raise ValueError( MSG_BAD_XREF)
5060: (8)                    if xref > 0:
```

```
5061: (12)                      obj = mupdf.pdf_load_object(pdf, xref)
5062: (8)                   else:
5063: (12)                      obj = mupdf.pdf_trailer(pdf)
5064: (8)                   res = JM_object_to_buffer(mupdf.pdf_resolve_indirect(obj), compressed,
ascii)
5065: (8)                   text = JM_EscapeStrFromBuffer(res)
5066: (8)                   return text
5067: (4)               def xref_set_key(self, xref, key, value):
5068: (8)                   """Set the value of a PDF dictionary key."""
5069: (8)                   if self.is_closed:
5070: (12)                      raise ValueError("document closed")
5071: (8)                   if not key or not isinstance(key, str) or
INVALID_NAME_CHARS.intersection(key) not in (set(), {"/"}):
5072: (12)                      raise ValueError("bad 'key'")
5073: (8)                   if not isinstance(value, str) or not value or value[0] == "/" and
INVALID_NAME_CHARS.intersection(value[1:]) != set():
5074: (12)                      raise ValueError("bad 'value'")
5075: (8)                   pdf = _as_pdf_document(self)
5076: (8)                   xreflen = mupdf.pdf_xref_len(pdf)
5077: (8)                   if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
5078: (12)                      raise ValueError( MSG_BAD_XREF)
5079: (8)                   if xref != -1:
5080: (12)                      obj = mupdf.pdf_load_object(pdf, xref)
5081: (8)                   else:
5082: (12)                      obj = mupdf.pdf_trailer(pdf)
5083: (8)                   new_obj = JM_set_object_value(obj, key, value)
5084: (8)                   if not new_obj.m_internal:
5085: (12)                      return  # did not work: skip update
5086: (8)                   if xref != -1:
5087: (12)                      mupdf.pdf_update_object(pdf, xref, new_obj)
5088: (8)                   else:
5089: (12)                      n = mupdf.pdf_dict_len(new_obj)
5090: (12)                      for i in range(n):
5091: (16)                          mupdf.pdf_dict_put(
5092: (24)                                  obj,
5093: (24)                                  mupdf.pdf_dict_get_key(new_obj, i),
5094: (24)                                  mupdf.pdf_dict_get_val(new_obj, i),
5095: (24)                                  )
5096: (4)               def xref_stream(self, xref):
5097: (8)                   """Get decompressed xref stream."""
5098: (8)                   if self.is_closed or self.is_encrypted:
5099: (12)                      raise ValueError("document closed or encrypted")
5100: (8)                   pdf = _as_pdf_document(self)
5101: (8)                   xreflen = mupdf.pdf_xref_len( pdf)
5102: (8)                   if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
5103: (12)                      raise ValueError( MSG_BAD_XREF)
5104: (8)                   if xref >= 0:
5105: (12)                      obj = mupdf.pdf_new_indirect( pdf, xref, 0)
5106: (8)                   else:
5107: (12)                      obj = mupdf.pdf_trailer( pdf)
5108: (8)                   r = None
5109: (8)                   if mupdf.pdf_is_stream( obj):
5110: (12)                      res = mupdf.pdf_load_stream_number( pdf, xref)
5111: (12)                      r = JM_BinFromBuffer( res)
5112: (8)                   return r
5113: (4)               def xref_stream_raw(self, xref):
5114: (8)                   """Get xref stream without decompression."""
5115: (8)                   if self.is_closed or self.is_encrypted:
5116: (12)                      raise ValueError("document closed or encrypted")
5117: (8)                   pdf = _as_pdf_document(self)
5118: (8)                   xreflen = mupdf.pdf_xref_len( pdf)
5119: (8)                   if not _INRANGE(xref, 1, xreflen-1) and xref != -1:
5120: (12)                      raise ValueError( MSG_BAD_XREF)
5121: (8)                   if xref >= 0:
5122: (12)                      obj = mupdf.pdf_new_indirect( pdf, xref, 0)
5123: (8)                   else:
5124: (12)                      obj = mupdf.pdf_trailer( pdf)
5125: (8)                   r = None
5126: (8)                   if mupdf.pdf_is_stream( obj):
```

```
5127: (12)                      res = mupdf.pdf_load_raw_stream_number( pdf, xref)
5128: (12)                      r = JM_BinFromBuffer( res)
5129: (8)                   return r
5130: (4)               def xref_xml_metadata(self):
5131: (8)                   """Get xref of document XML metadata."""
5132: (8)                   pdf = _as_pdf_document(self)
5133: (8)                   root = mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf), PDF_NAME('Root'))
5134: (8)                   if not root.m_internal:
5135: (12)                      RAISEPY( MSG_BAD_PDFROOT, JM_Exc_FileDataError)
5136: (8)                   xml = mupdf.pdf_dict_get( root, PDF_NAME('Metadata'))
5137: (8)                   xref = 0
5138: (8)                   if xml.m_internal:
5139: (12)                      xref = mupdf.pdf_to_num( xml)
5140: (8)                   return xref
5141: (4)               __slots__ = ('this', 'page_count2', 'this_is_pdf', '__dict__')
5142: (4)               outline = property(lambda self: self._outline)
5143: (4)               tobytes = write
5144: (4)               is_stream = xref_is_stream
5145: (0)           open = Document
5146: (0)           class DocumentWriter:
5147: (4)               def __enter__(self):
5148: (8)                   return self
5149: (4)               def __exit__(self, *args):
5150: (8)                   self.close()
5151: (4)               def __init__(self, path, options=''):
5152: (8)                   if isinstance( path, str):
5153: (12)                      pass
5154: (8)                   elif hasattr( path, 'absolute'):
5155: (12)                      path = str( path)
5156: (8)                   elif hasattr( path, 'name'):
5157: (12)                      path = path.name
5158: (8)                   if isinstance( path, str):
5159: (12)                      self.this = mupdf.FzDocumentWriter( path, options,
mupdf.FzDocumentWriter.PathType_PDF)
5160: (8)                   else:
5161: (12)                      out = JM_new_output_fileptr( path)
5162: (12)                      self.this = mupdf.FzDocumentWriter( out, options,
mupdf.FzDocumentWriter.OutputType_PDF)
5163: (12)                      assert out.m_internal_value() == 0
5164: (12)                      assert hasattr( self.this, '_out')
5165: (4)               def begin_page( self, mediabox):
5166: (8)                   mediabox2 = JM_rect_from_py(mediabox)
5167: (8)                   device = mupdf.fz_begin_page( self.this, mediabox2)
5168: (8)                   device_wrapper = DeviceWrapper( device)
5169: (8)                   return device_wrapper
5170: (4)               def close( self):
5171: (8)                   mupdf.fz_close_document_writer( self.this)
5172: (4)               def end_page( self):
5173: (8)                   mupdf.fz_end_page( self.this)
5174: (0)           class Font:
5175: (4)               def __del__(self):
5176: (8)                   if type(self) is not Font:
5177: (12)                      return None
5178: (4)               def __init__(
5179: (12)                      self,
5180: (12)                      fontname=None,
5181: (12)                      fontfile=None,
5182: (12)                      fontbuffer=None,
5183: (12)                      script=0,
5184: (12)                      language=None,
5185: (12)                      ordering=-1,
5186: (12)                      is_bold=0,
5187: (12)                      is_italic=0,
5188: (12)                      is_serif=0,
5189: (12)                      embed=1,
5190: (12)                      ):
5191: (8)                   if fontbuffer:
5192: (12)                      if hasattr(fontbuffer, "getvalue"):
5193: (16)                          fontbuffer = fontbuffer.getvalue()
```

```
5194: (12)                      elif isinstance(fontbuffer, bytearray):
5195: (16)                          fontbuffer = bytes(fontbuffer)
5196: (12)                      if not isinstance(fontbuffer, bytes):
5197: (16)                          raise ValueError("bad type: 'fontbuffer'")
5198: (8)                   if isinstance(fontname, str):
5199: (12)                      fname_lower = fontname.lower()
5200: (12)                      if "/" in fname_lower or "\\" in fname_lower or "." in
fname_lower:
5201: (16)                          message("Warning: did you mean a fontfile?")
5202: (12)                      if fname_lower in ("cjk", "china-t", "china-ts"):
5203: (16)                          ordering = 0
5204: (12)                      elif fname_lower.startswith("china-s"):
5205: (16)                          ordering = 1
5206: (12)                      elif fname_lower.startswith("korea"):
5207: (16)                          ordering = 3
5208: (12)                      elif fname_lower.startswith("japan"):
5209: (16)                          ordering = 2
5210: (12)                      elif fname_lower in fitz_fontdescriptors.keys():
5211: (16)                          import pymupdf_fonts  # optional fonts
5212: (16)                          fontbuffer = pymupdf_fonts.myfont(fname_lower)  # make a copy
5213: (16)                          fontname = None  # ensure using fontbuffer only
5214: (16)                          del pymupdf_fonts  # remove package again
5215: (12)                      elif ordering < 0:
5216: (16)                          fontname = Base14_fontdict.get(fontname, fontname)
5217: (8)                   lang = mupdf.fz_text_language_from_string(language)
5218: (8)                   font = JM_get_font(fontname, fontfile,
5219: (19)                             fontbuffer, script, lang, ordering,
5220: (19)                             is_bold, is_italic, is_serif, embed)
5221: (8)                   self.this = font
5222: (4)               def __repr__(self):
5223: (8)                   return "Font('%s')" % self.name
5224: (4)               def _valid_unicodes(self, arr):
5225: (8)                   assert 0, 'Not implemented because implementation requires
FT_Get_First_Char() etc.'
5226: (4)               @property
5227: (4)               def ascender(self):
5228: (8)                   """Return the glyph ascender value."""
5229: (8)                   return mupdf.fz_font_ascender(self.this)
5230: (4)               @property
5231: (4)               def bbox(self):
5232: (8)                   return self.this.fz_font_bbox()
5233: (4)               @property
5234: (4)               def buffer(self):
5235: (8)                   buffer_ = mupdf.FzBuffer( mupdf.ll_fz_keep_buffer(
self.this.m_internal.buffer))
5236: (8)                   return mupdf.fz_buffer_extract_copy( buffer_)
5237: (4)               def char_lengths(self, text, fontsize=11, language=None, script=0,
wmode=0, small_caps=0):
5238: (8)                   """Return tuple of char lengths of unicode 'text' under a fontsize."""
5239: (8)                   lang = mupdf.fz_text_language_from_string(language)
5240: (8)                   rc = []
5241: (8)                   for ch in text:
5242: (12)                      c = ord(ch)
5243: (12)                      if small_caps:
5244: (16)                          gid = mupdf.fz_encode_character_sc(self.this, c)
5245: (16)                          if gid >= 0:
5246: (20)                              font = self.this
5247: (12)                      else:
5248: (16)                          gid, font = mupdf.fz_encode_character_with_fallback(self.this,
c, script, lang)
5249: (12)                      rc.append(fontsize * mupdf.fz_advance_glyph(font, gid, wmode))
5250: (8)                   return rc
5251: (4)               @property
5252: (4)               def descender(self):
5253: (8)                   """Return the glyph descender value."""
5254: (8)                   return mupdf.fz_font_descender(self.this)
5255: (4)               @property
5256: (4)               def flags(self):
5257: (8)                   f = mupdf.ll_fz_font_flags(self.this.m_internal)
```

```
5258: (8)                    if not f:
5259: (12)                       return
5260: (8)                    assert isinstance( f, mupdf.fz_font_flags_t)
5261: (8)                    if mupdf_cppyy:
5262: (12)                       v = [f.is_mono]
5263: (12)                       def b(bits):
5264: (16)                           ret = v[0] & ((1 << bits)-1)
5265: (16)                           v[0] = v[0] >> bits
5266: (16)                           return ret
5267: (12)                       is_mono = b(1)
5268: (12)                       is_serif = b(1)
5269: (12)                       is_bold = b(1)
5270: (12)                       is_italic = b(1)
5271: (12)                       ft_substitute = b(1)
5272: (12)                       ft_stretch = b(1)
5273: (12)                       fake_bold = b(1)
5274: (12)                       fake_italic = b(1)
5275: (12)                       has_opentype = b(1)
5276: (12)                       invalid_bbox = b(1)
5277: (12)                       cjk_lang = b(1)
5278: (12)                       embed = b(1)
5279: (12)                       never_embed = b(1)
5280: (8)                    return {
5281: (16)                           "mono":         is_mono if mupdf_cppyy else f.is_mono,
5282: (16)                           "serif":        is_serif if mupdf_cppyy else f.is_serif,
5283: (16)                           "bold":         is_bold if mupdf_cppyy else f.is_bold,
5284: (16)                           "italic":       is_italic if mupdf_cppyy else f.is_italic,
5285: (16)                           "substitute":   ft_substitute if mupdf_cppyy else
f.ft_substitute,
5286: (16)                           "stretch":      ft_stretch if mupdf_cppyy else f.ft_stretch,
5287: (16)                           "fake-bold":    fake_bold if mupdf_cppyy else f.fake_bold,
5288: (16)                           "fake-italic":  fake_italic if mupdf_cppyy else f.fake_italic,
5289: (16)                           "opentype":     has_opentype if mupdf_cppyy else
f.has_opentype,
5290: (16)                           "invalid-bbox": invalid_bbox if mupdf_cppyy else
f.invalid_bbox,
5291: (16)                           'cjk':          cjk_lang if mupdf_cppyy else f.cjk,
5292: (16)                           'cjk-lang':     cjk_lang if mupdf_cppyy else f.cjk_lang,
5293: (16)                           'embed':        embed if mupdf_cppyy else f.embed,
5294: (16)                           'never-embed':  never_embed if mupdf_cppyy else f.never_embed,
5295: (16)                           }
5296: (4)                def glyph_advance(self, chr_, language=None, script=0, wmode=0,
small_caps=0):
5297: (8)                    """Return the glyph width of a unicode (font size 1)."""
5298: (8)                    lang = mupdf.fz_text_language_from_string(language)
5299: (8)                    if small_caps:
5300: (12)                       gid = mupdf.fz_encode_character_sc(self.this, chr_)
5301: (12)                       if gid >= 0:
5302: (16)                           font = self.this
5303: (8)                    else:
5304: (12)                       gid, font = mupdf.fz_encode_character_with_fallback(self.this,
chr_, script, lang)
5305: (8)                    return mupdf.fz_advance_glyph(font, gid, wmode)
5306: (4)                def glyph_bbox(self, char, language=None, script=0, small_caps=0):
5307: (8)                    """Return the glyph bbox of a unicode (font size 1)."""
5308: (8)                    lang = mupdf.fz_text_language_from_string(language)
5309: (8)                    if small_caps:
5310: (12)                       gid = mupdf.fz_encode_character_sc( self.this, char)
5311: (12)                       if gid >= 0:
5312: (16)                           font = self.this
5313: (8)                    else:
5314: (12)                       gid, font = mupdf.fz_encode_character_with_fallback( self.this,
char, script, lang)
5315: (8)                    return Rect(mupdf.fz_bound_glyph( font, gid, mupdf.FzMatrix()))
5316: (4)                @property
5317: (4)                def glyph_count(self):
5318: (8)                    return self.this.m_internal.glyph_count
5319: (4)                def glyph_name_to_unicode(self, name):
5320: (8)                    """Return the unicode for a glyph name."""
```

```
5321: (8)                   return glyph_name_to_unicode(name)
5322: (4)               def has_glyph(self, chr, language=None, script=0, fallback=0,
small_caps=0):
5323: (8)                   """Check whether font has a glyph for this unicode."""
5324: (8)                   if fallback:
5325: (12)                      lang = mupdf.fz_text_language_from_string(language)
5326: (12)                      gid, font = mupdf.fz_encode_character_with_fallback(self.this,
chr, script, lang)
5327: (8)                   else:
5328: (12)                      if small_caps:
5329: (16)                          gid = mupdf.fz_encode_character_sc(self.this, chr)
5330: (12)                      else:
5331: (16)                          gid = mupdf.fz_encode_character(self.this, chr)
5332: (8)                   return gid
5333: (4)               @property
5334: (4)               def is_bold(self):
5335: (8)                   return mupdf.fz_font_is_bold( self.this)
5336: (4)               @property
5337: (4)               def is_italic(self):
5338: (8)                   return mupdf.fz_font_is_italic( self.this)
5339: (4)               @property
5340: (4)               def is_monospaced(self):
5341: (8)                   return mupdf.fz_font_is_monospaced( self.this)
5342: (4)               @property
5343: (4)               def is_serif(self):
5344: (8)                   return mupdf.fz_font_is_serif( self.this)
5345: (4)               @property
5346: (4)               def is_writable(self):
5347: (8)                   return True # see pymupdf commit ef4056ee4da2
5348: (8)                   font = self.this
5349: (8)                   flags = mupdf.ll_fz_font_flags(font.m_internal)
5350: (8)                   if mupdf_cppyy:
5351: (12)                      import cppyy
5352: (12)                      ft_substitute = cppyy.gbl.mupdf_mfz_font_flags_ft_substitute(
flags)
5353: (8)                   else:
5354: (12)                      ft_substitute = flags.ft_substitute
5355: (8)                   if ( mupdf.ll_fz_font_t3_procs(font.m_internal)
5356: (16)                          or ft_substitute
5357: (16)                          or not mupdf.pdf_font_writing_supported(font)
5358: (16)                          ):
5359: (12)                      return False
5360: (8)                   return True
5361: (4)               @property
5362: (4)               def name(self):
5363: (8)                   ret = mupdf.fz_font_name(self.this)
5364: (8)                   return ret
5365: (4)               def text_length(self, text, fontsize=11, language=None, script=0, wmode=0,
small_caps=0):
5366: (8)                   """Return length of unicode 'text' under a fontsize."""
5367: (8)                   thisfont = self.this
5368: (8)                   lang = mupdf.fz_text_language_from_string(language)
5369: (8)                   rc = 0
5370: (8)                   if not isinstance(text, str):
5371: (12)                      raise TypeError( MSG_BAD_TEXT)
5372: (8)                   for ch in text:
5373: (12)                      c = ord(ch)
5374: (12)                      if small_caps:
5375: (16)                          gid = mupdf.fz_encode_character_sc(thisfont, c)
5376: (16)                          if gid >= 0:
5377: (20)                              font = thisfont
5378: (12)                      else:
5379: (16)                          gid, font = mupdf.fz_encode_character_with_fallback(thisfont,
c, script, lang)
5380: (12)                      rc += mupdf.fz_advance_glyph(font, gid, wmode)
5381: (8)                   rc *= fontsize
5382: (8)                   return rc
5383: (4)               def unicode_to_glyph_name(self, ch):
5384: (8)                   """Return the glyph name for a unicode."""
```

```
5385: (8)                   return unicode_to_glyph_name(ch)
5386: (4)               def valid_codepoints(self):
5387: (8)                   '''
5388: (8)                   list of valid unicodes of a fz_font
5389: (8)                   '''
5390: (8)                   return []
5391: (8)                   from array import array
5392: (8)                   gc = self.glyph_count
5393: (8)                   cp = array("l", (0,) * gc)
5394: (8)                   arr = cp.buffer_info()
5395: (8)                   self._valid_unicodes(arr)
5396: (8)                   return array("l", sorted(set(cp))[1:])
5397: (0)           class Graftmap:
5398: (4)               def __del__(self):
5399: (8)                   if not type(self) is Graftmap:
5400: (12)                      return
5401: (8)                   self.thisown = False
5402: (4)               def __init__(self, doc):
5403: (8)                   dst = _as_pdf_document(doc)
5404: (8)                   map_ = mupdf.pdf_new_graft_map(dst)
5405: (8)                   self.this = map_
5406: (8)                   self.thisown = True
5407: (0)           class Link:
5408: (4)               def __del__(self):
5409: (8)                   self._erase()
5410: (4)               def __init__( self, this):
5411: (8)                   assert isinstance( this, mupdf.FzLink)
5412: (8)                   self.this = this
5413: (4)               def __repr__(self):
5414: (8)                   CheckParent(self)
5415: (8)                   return "link on " + str(self.parent)
5416: (4)               def __str__(self):
5417: (8)                   CheckParent(self)
5418: (8)                   return "link on " + str(self.parent)
5419: (4)               def _border(self, doc, xref):
5420: (8)                   pdf = _as_pdf_document(doc, required=0)
5421: (8)                   if not pdf.m_internal:
5422: (12)                      return
5423: (8)                   link_obj = mupdf.pdf_new_indirect(pdf, xref, 0)
5424: (8)                   if not link_obj.m_internal:
5425: (12)                      return
5426: (8)                   b = JM_annot_border(link_obj)
5427: (8)                   return b
5428: (4)               def _colors(self, doc, xref):
5429: (8)                   pdf = _as_pdf_document(doc, required=0)
5430: (8)                   if not pdf.m_internal:
5431: (12)                      return
5432: (8)                   link_obj = mupdf.pdf_new_indirect( pdf, xref, 0)
5433: (8)                   if not link_obj.m_internal:
5434: (12)                      raise ValueError( MSG_BAD_XREF)
5435: (8)                   b = JM_annot_colors( link_obj)
5436: (8)                   return b
5437: (4)               def _erase(self):
5438: (8)                   self.parent = None
5439: (8)                   self.thisown = False
5440: (4)               def _setBorder(self, border, doc, xref):
5441: (8)                   pdf = _as_pdf_document(doc, required=0)
5442: (8)                   if not pdf.m_internal:
5443: (12)                      return
5444: (8)                   link_obj = mupdf.pdf_new_indirect(pdf, xref, 0)
5445: (8)                   if not link_obj.m_internal:
5446: (12)                      return
5447: (8)                   b = JM_annot_set_border(border, pdf, link_obj)
5448: (8)                   return b
5449: (4)               @property
5450: (4)               def border(self):
5451: (8)                   return self._border(self.parent.parent.this, self.xref)
5452: (4)               @property
5453: (4)               def colors(self):
```

```
5454: (8)                   return self._colors(self.parent.parent.this, self.xref)
5455: (4)               @property
5456: (4)               def dest(self):
5457: (8)                   """Create link destination details."""
5458: (8)                   if hasattr(self, "parent") and self.parent is None:
5459: (12)                      raise ValueError("orphaned object: parent is None")
5460: (8)                   if self.parent.parent.is_closed or self.parent.parent.is_encrypted:
5461: (12)                      raise ValueError("document closed or encrypted")
5462: (8)                   doc = self.parent.parent
5463: (8)                   if self.is_external or self.uri.startswith("#"):
5464: (12)                      uri = None
5465: (8)                   else:
5466: (12)                      uri = doc.resolve_link(self.uri)
5467: (8)                   return linkDest(self, uri, doc)
5468: (4)               @property
5469: (4)               def flags(self)->int:
5470: (8)                   CheckParent(self)
5471: (8)                   doc = self.parent.parent
5472: (8)                   if not doc.is_pdf:
5473: (12)                      return 0
5474: (8)                   f = doc.xref_get_key(self.xref, "F")
5475: (8)                   if f[1] != "null":
5476: (12)                      return int(f[1])
5477: (8)                   return 0
5478: (4)               @property
5479: (4)               def is_external(self):
5480: (8)                   """Flag the link as external."""
5481: (8)                   CheckParent(self)
5482: (8)                   if g_use_extra:
5483: (12)                      return extra.Link_is_external( self.this)
5484: (8)                   this_link = self.this
5485: (8)                   if not this_link.m_internal or not this_link.m_internal.uri:
5486: (12)                      return False
5487: (8)                   return bool( mupdf.fz_is_external_link( this_link.m_internal.uri))
5488: (4)               @property
5489: (4)               def next(self):
5490: (8)                   """Next link."""
5491: (8)                   if not self.this.m_internal:
5492: (12)                      return None
5493: (8)                   CheckParent(self)
5494: (8)                   if 0 and g_use_extra:
5495: (12)                      val = extra.Link_next( self.this)
5496: (8)                   else:
5497: (12)                      val = self.this.next()
5498: (8)                   if not val.m_internal:
5499: (12)                      return None
5500: (8)                   val = Link( val)
5501: (8)                   if val:
5502: (12)                      val.thisown = True
5503: (12)                      val.parent = self.parent  # copy owning page from prev link
5504: (12)                      val.parent._annot_refs[id(val)] = val
5505: (12)                      if self.xref > 0:  # prev link has an xref
5506: (16)                          link_xrefs = [x[0] for x in self.parent.annot_xrefs() if x[1]
== mupdf.PDF_ANNOT_LINK]
5507: (16)                          link_ids = [x[2] for x in self.parent.annot_xrefs() if x[1] ==
mupdf.PDF_ANNOT_LINK]
5508: (16)                          idx = link_xrefs.index(self.xref)
5509: (16)                          val.xref = link_xrefs[idx + 1]
5510: (16)                          val.id = link_ids[idx + 1]
5511: (12)                      else:
5512: (16)                          val.xref = 0
5513: (16)                          val.id = ""
5514: (8)                   return val
5515: (4)               @property
5516: (4)               def rect(self):
5517: (8)                   """Rectangle ('hot area')."""
5518: (8)                   CheckParent(self)
5519: (8)                   if self.this is None or not self.this.m_internal:
5520: (12)                      raise Exception( 'self.this.m_internal not available')
```

```
5521: (8)                   val = JM_py_from_rect( self.this.rect())
5522: (8)                   val = Rect(val)
5523: (8)                   return val
5524: (4)               def set_border(self, border=None, width=0, dashes=None, style=None):
5525: (8)                   if type(border) is not dict:
5526: (12)                      border = {"width": width, "style": style, "dashes": dashes}
5527: (8)                   return self._setBorder(border, self.parent.parent.this, self.xref)
5528: (4)               def set_colors(self, colors=None, stroke=None, fill=None):
5529: (8)                   """Set border colors."""
5530: (8)                   CheckParent(self)
5531: (8)                   doc = self.parent.parent
5532: (8)                   if type(colors) is not dict:
5533: (12)                      colors = {"fill": fill, "stroke": stroke}
5534: (8)                   fill = colors.get("fill")
5535: (8)                   stroke = colors.get("stroke")
5536: (8)                   if fill is not None:
5537: (12)                      message("warning: links have no fill color")
5538: (8)                   if stroke in ([], ()):
5539: (12)                      doc.xref_set_key(self.xref, "C", "[]")
5540: (12)                      return
5541: (8)                   if hasattr(stroke, "__float__"):
5542: (12)                      stroke = [float(stroke)]
5543: (8)                   CheckColor(stroke)
5544: (8)                   assert len(stroke) in (1, 3, 4)
5545: (8)                   s = f"[{_format_g(stroke)}]"
5546: (8)                   doc.xref_set_key(self.xref, "C", s)
5547: (4)               def set_flags(self, flags):
5548: (8)                   CheckParent(self)
5549: (8)                   doc = self.parent.parent
5550: (8)                   if not doc.is_pdf:
5551: (12)                      raise ValueError("is no PDF")
5552: (8)                   if not type(flags) is int:
5553: (12)                      raise ValueError("bad 'flags' value")
5554: (8)                   doc.xref_set_key(self.xref, "F", str(flags))
5555: (8)                   return None
5556: (4)               @property
5557: (4)               def uri(self):
5558: (8)                   """Uri string."""
5559: (8)                   if g_use_extra:
5560: (12)                      return extra.link_uri(self.this)
5561: (8)                   this_link = self.this
5562: (8)                   return this_link.m_internal.uri if this_link.m_internal else ''
5563: (4)               page = -1
5564: (0)           class Matrix:
5565: (4)               def __abs__(self):
5566: (8)                   return math.sqrt(sum([c*c for c in self]))
5567: (4)               def __add__(self, m):
5568: (8)                   if hasattr(m, "__float__"):
5569: (12)                      return Matrix(self.a + m, self.b + m, self.c + m,
5570: (26)                                    self.d + m, self.e + m, self.f + m)
5571: (8)                   if len(m) != 6:
5572: (12)                      raise ValueError("Matrix: bad seq len")
5573: (8)                   return Matrix(self.a + m[0], self.b + m[1], self.c + m[2],
5574: (26)                                self.d + m[3], self.e + m[4], self.f + m[5])
5575: (4)               def __bool__(self):
5576: (8)                   return not (max(self) == min(self) == 0)
5577: (4)               def __eq__(self, mat):
5578: (8)                   if not hasattr(mat, "__len__"):
5579: (12)                      return False
5580: (8)                   return len(mat) == 6 and bool(self - mat) is False
5581: (4)               def __getitem__(self, i):
5582: (8)                   return (self.a, self.b, self.c, self.d, self.e, self.f)[i]
5583: (4)               def __init__(self, *args, a=None, b=None, c=None, d=None, e=None, f=None):
5584: (8)                   """
5585: (8)                   Matrix() - all zeros
5586: (8)                   Matrix(a, b, c, d, e, f)
5587: (8)                   Matrix(zoom-x, zoom-y) - zoom
5588: (8)                   Matrix(shear-x, shear-y, 1) - shear
5589: (8)                   Matrix(degree) - rotate
```

```
5590: (8)                   Matrix(Matrix) - new copy
5591: (8)                   Matrix(sequence) - from 'sequence'
5592: (8)                   Matrix(mupdf.FzMatrix) - from MuPDF class wrapper for fz_matrix.
5593: (8)                   Explicit keyword args a, b, c, d, e, f override any earlier settings
if
5594: (8)                   not None.
5595: (8)                   """
5596: (8)                   if not args:
5597: (12)                      self.a = self.b = self.c = self.d = self.e = self.f = 0.0
5598: (8)                   elif len(args) > 6:
5599: (12)                      raise ValueError("Matrix: bad seq len")
5600: (8)                   elif len(args) == 6:  # 6 numbers
5601: (12)                      self.a, self.b, self.c, self.d, self.e, self.f = map(float, args)
5602: (8)                   elif len(args) == 1:  # either an angle or a sequ
5603: (12)                      if isinstance(args[0], mupdf.FzMatrix):
5604: (16)                          self.a = args[0].a
5605: (16)                          self.b = args[0].b
5606: (16)                          self.c = args[0].c
5607: (16)                          self.d = args[0].d
5608: (16)                          self.e = args[0].e
5609: (16)                          self.f = args[0].f
5610: (12)                      elif hasattr(args[0], "__float__"):
5611: (16)                          theta = math.radians(args[0])
5612: (16)                          c_ = round(math.cos(theta), 8)
5613: (16)                          s_ = round(math.sin(theta), 8)
5614: (16)                          self.a = self.d = c_
5615: (16)                          self.b = s_
5616: (16)                          self.c = -s_
5617: (16)                          self.e = self.f = 0.0
5618: (12)                      else:
5619: (16)                          self.a, self.b, self.c, self.d, self.e, self.f = map(float,
args[0])
5620: (8)                   elif len(args) == 2 or len(args) == 3 and args[2] == 0:
5621: (12)                      self.a, self.b, self.c, self.d, self.e, self.f = float(args[0]), \
5622: (16)                          0.0, 0.0, float(args[1]), 0.0, 0.0
5623: (8)                   elif len(args) == 3 and args[2] == 1:
5624: (12)                      self.a, self.b, self.c, self.d, self.e, self.f = 1.0, \
5625: (16)                          float(args[1]), float(args[0]), 1.0, 0.0, 0.0
5626: (8)                   else:
5627: (12)                      raise ValueError("Matrix: bad args")
5628: (8)                   if a is not None:   self.a = a
5629: (8)                   if b is not None:   self.b = b
5630: (8)                   if c is not None:   self.c = c
5631: (8)                   if d is not None:   self.d = d
5632: (8)                   if e is not None:   self.e = e
5633: (8)                   if f is not None:   self.f = f
5634: (4)               def __invert__(self):
5635: (8)                   """Calculate inverted matrix."""
5636: (8)                   m1 = Matrix()
5637: (8)                   m1.invert(self)
5638: (8)                   return m1
5639: (4)               def __len__(self):
5640: (8)                   return 6
5641: (4)               def __mul__(self, m):
5642: (8)                   if hasattr(m, "__float__"):
5643: (12)                      return Matrix(self.a * m, self.b * m, self.c * m,
5644: (26)                                    self.d * m, self.e * m, self.f * m)
5645: (8)                   m1 = Matrix(1,1)
5646: (8)                   return m1.concat(self, m)
5647: (4)               def __neg__(self):
5648: (8)                   return Matrix(-self.a, -self.b, -self.c, -self.d, -self.e, -self.f)
5649: (4)               def __nonzero__(self):
5650: (8)                   return not (max(self) == min(self) == 0)
5651: (4)               def __pos__(self):
5652: (8)                   return Matrix(self)
5653: (4)               def __repr__(self):
5654: (8)                   return "Matrix" + str(tuple(self))
5655: (4)               def __setitem__(self, i, v):
5656: (8)                   v = float(v)
```

```
5657: (8)                   if   i == 0: self.a = v
5658: (8)                   elif i == 1: self.b = v
5659: (8)                   elif i == 2: self.c = v
5660: (8)                   elif i == 3: self.d = v
5661: (8)                   elif i == 4: self.e = v
5662: (8)                   elif i == 5: self.f = v
5663: (8)                   else:
5664: (12)                      raise IndexError("index out of range")
5665: (8)                   return
5666: (4)               def __sub__(self, m):
5667: (8)                   if hasattr(m, "__float__"):
5668: (12)                      return Matrix(self.a - m, self.b - m, self.c - m,
5669: (26)                                    self.d - m, self.e - m, self.f - m)
5670: (8)                   if len(m) != 6:
5671: (12)                      raise ValueError("Matrix: bad seq len")
5672: (8)                   return Matrix(self.a - m[0], self.b - m[1], self.c - m[2],
5673: (26)                                    self.d - m[3], self.e - m[4], self.f - m[5])
5674: (4)               def __truediv__(self, m):
5675: (8)                   if hasattr(m, "__float__"):
5676: (12)                      return Matrix(self.a * 1./m, self.b * 1./m, self.c * 1./m,
5677: (26)                                    self.d * 1./m, self.e * 1./m, self.f * 1./m)
5678: (8)                   m1 = util_invert_matrix(m)[1]
5679: (8)                   if not m1:
5680: (12)                      raise ZeroDivisionError("matrix not invertible")
5681: (8)                   m2 = Matrix(1,1)
5682: (8)                   return m2.concat(self, m1)
5683: (4)               def concat(self, one, two):
5684: (8)                   """Multiply two matrices and replace current one."""
5685: (8)                   if not len(one) == len(two) == 6:
5686: (12)                      raise ValueError("Matrix: bad seq len")
5687: (8)                   self.a, self.b, self.c, self.d, self.e, self.f =
util_concat_matrix(one, two)
5688: (8)                   return self
5689: (4)               def invert(self, src=None):
5690: (8)                   """Calculate the inverted matrix. Return 0 if successful and replace
5691: (8)                   current one. Else return 1 and do nothing.
5692: (8)                   """
5693: (8)                   if src is None:
5694: (12)                      dst = util_invert_matrix(self)
5695: (8)                   else:
5696: (12)                      dst = util_invert_matrix(src)
5697: (8)                   if dst[0] == 1:
5698: (12)                      return 1
5699: (8)                   self.a, self.b, self.c, self.d, self.e, self.f = dst[1]
5700: (8)                   return 0
5701: (4)               @property
5702: (4)               def is_rectilinear(self):
5703: (8)                   """True if rectangles are mapped to rectangles."""
5704: (8)                   return (abs(self.b) < EPSILON and abs(self.c) < EPSILON) or \
5705: (12)                      (abs(self.a) < EPSILON and abs(self.d) < EPSILON)
5706: (4)               def prerotate(self, theta):
5707: (8)                   """Calculate pre rotation and replace current matrix."""
5708: (8)                   theta = float(theta)
5709: (8)                   while theta < 0: theta += 360
5710: (8)                   while theta >= 360: theta -= 360
5711: (8)                   if abs(0 - theta) < EPSILON:
5712: (12)                      pass
5713: (8)                   elif abs(90.0 - theta) < EPSILON:
5714: (12)                      a = self.a
5715: (12)                      b = self.b
5716: (12)                      self.a = self.c
5717: (12)                      self.b = self.d
5718: (12)                      self.c = -a
5719: (12)                      self.d = -b
5720: (8)                   elif abs(180.0 - theta) < EPSILON:
5721: (12)                      self.a = -self.a
5722: (12)                      self.b = -self.b
5723: (12)                      self.c = -self.c
5724: (12)                      self.d = -self.d
```

```
5725: (8)                   elif abs(270.0 - theta) < EPSILON:
5726: (12)                      a = self.a
5727: (12)                      b = self.b
5728: (12)                      self.a = -self.c
5729: (12)                      self.b = -self.d
5730: (12)                      self.c = a
5731: (12)                      self.d = b
5732: (8)                   else:
5733: (12)                      rad = math.radians(theta)
5734: (12)                      s = math.sin(rad)
5735: (12)                      c = math.cos(rad)
5736: (12)                      a = self.a
5737: (12)                      b = self.b
5738: (12)                      self.a = c * a + s * self.c
5739: (12)                      self.b = c * b + s * self.d
5740: (12)                      self.c =-s * a + c * self.c
5741: (12)                      self.d =-s * b + c * self.d
5742: (8)                   return self
5743: (4)               def prescale(self, sx, sy):
5744: (8)                   """Calculate pre scaling and replace current matrix."""
5745: (8)                   sx = float(sx)
5746: (8)                   sy = float(sy)
5747: (8)                   self.a *= sx
5748: (8)                   self.b *= sx
5749: (8)                   self.c *= sy
5750: (8)                   self.d *= sy
5751: (8)                   return self
5752: (4)               def preshear(self, h, v):
5753: (8)                   """Calculate pre shearing and replace current matrix."""
5754: (8)                   h = float(h)
5755: (8)                   v = float(v)
5756: (8)                   a, b = self.a, self.b
5757: (8)                   self.a += v * self.c
5758: (8)                   self.b += v * self.d
5759: (8)                   self.c += h * a
5760: (8)                   self.d += h * b
5761: (8)                   return self
5762: (4)               def pretranslate(self, tx, ty):
5763: (8)                   """Calculate pre translation and replace current matrix."""
5764: (8)                   tx = float(tx)
5765: (8)                   ty = float(ty)
5766: (8)                   self.e += tx * self.a + ty * self.c
5767: (8)                   self.f += tx * self.b + ty * self.d
5768: (8)                   return self
5769: (4)               __inv__ = __invert__
5770: (4)               __div__ = __truediv__
5771: (4)               norm = __abs__
5772: (0)           class IdentityMatrix(Matrix):
5773: (4)               """Identity matrix [1, 0, 0, 1, 0, 0]"""
5774: (4)               def __hash__(self):
5775: (8)                   return hash((1,0,0,1,0,0))
5776: (4)               def __init__(self):
5777: (8)                   Matrix.__init__(self, 1.0, 1.0)
5778: (4)               def __repr__(self):
5779: (8)                   return "IdentityMatrix(1.0, 0.0, 0.0, 1.0, 0.0, 0.0)"
5780: (4)               def __setattr__(self, name, value):
5781: (8)                   if name in "ad":
5782: (12)                      self.__dict__[name] = 1.0
5783: (8)                   elif name in "bcef":
5784: (12)                      self.__dict__[name] = 0.0
5785: (8)                   else:
5786: (12)                      self.__dict__[name] = value
5787: (4)               def checkargs(*args):
5788: (8)                   raise NotImplementedError("Identity is readonly")
5789: (0)           Identity = IdentityMatrix()
5790: (0)           class linkDest:
5791: (4)               """link or outline destination details"""
5792: (4)               def __init__(self, obj, rlink, document=None):
5793: (8)                   isExt = obj.is_external
```

```
5794: (8)                   isInt = not isExt
5795: (8)                   self.dest = ""
5796: (8)                   self.file_spec = ""
5797: (8)                   self.flags = 0
5798: (8)                   self.is_map = False
5799: (8)                   self.is_uri = False
5800: (8)                   self.kind = LINK_NONE
5801: (8)                   self.lt = Point(0, 0)
5802: (8)                   self.named = dict()
5803: (8)                   self.new_window = ""
5804: (8)                   self.page = obj.page
5805: (8)                   self.rb = Point(0, 0)
5806: (8)                   self.uri = obj.uri
5807: (8)                   def uri_to_dict(uri):
5808: (12)                      items = self.uri[1:].split('&')
5809: (12)                      ret = dict()
5810: (12)                      for item in items:
5811: (16)                          eq = item.find('=')
5812: (16)                          if eq >= 0:
5813: (20)                              ret[item[:eq]] = item[eq+1:]
5814: (16)                          else:
5815: (20)                              ret[item] = None
5816: (12)                      return ret
5817: (8)                   def unescape(name):
5818: (12)                      """Unescape '%AB' substrings to chr(0xAB)."""
5819: (12)                      split = name.replace("%%", "%25")  # take care of escaped '%'
5820: (12)                      split = split.split("%")
5821: (12)                      newname = split[0]
5822: (12)                      for item in split[1:]:
5823: (16)                          piece = item[:2]
5824: (16)                          newname += chr(int(piece, base=16))
5825: (16)                          newname += item[2:]
5826: (12)                      return newname
5827: (8)                   if rlink and not self.uri.startswith("#"):
5828: (12)                      self.uri = f"#page={rlink[0] + 1}&zoom=0,{_format_g(rlink[1])},
{_format_g(rlink[2])}"
5829: (8)                   if obj.is_external:
5830: (12)                      self.page = -1
5831: (12)                      self.kind = LINK_URI
5832: (8)                   if not self.uri:
5833: (12)                      self.page = -1
5834: (12)                      self.kind = LINK_NONE
5835: (8)                   if isInt and self.uri:
5836: (12)                      self.uri = self.uri.replace("&zoom=nan", "&zoom=0")
5837: (12)                      if self.uri.startswith("#"):
5838: (16)                          self.kind = LINK_GOTO
5839: (16)                          m = re.match('^#page=([0-9]+)&zoom=([0-9.]+),(-?[0-9.]+),(-?
[0-9.]+)$', self.uri)
5840: (16)                          if m:
5841: (20)                              self.page = int(m.group(1)) - 1
5842: (20)                              self.lt = Point(float((m.group(3))), float(m.group(4)))
5843: (20)                              self.flags = self.flags | LINK_FLAG_L_VALID |
LINK_FLAG_T_VALID
5844: (16)                          else:
5845: (20)                              m = re.match('^#page=([0-9]+)$', self.uri)
5846: (20)                              if m:
5847: (24)                                  self.page = int(m.group(1)) - 1
5848: (20)                              else:
5849: (24)                                  self.kind = LINK_NAMED
5850: (24)                                  m = re.match('^#nameddest=(.*)', self.uri)
5851: (24)                                  assert document
5852: (24)                                  if document and m:
5853: (28)                                      named = unescape(m.group(1))
5854: (28)                                      self.named = document.resolve_names().get(named)
5855: (28)                                      if self.named is None:
5856: (32)                                          self.named = dict()
5857: (28)                                      self.named['nameddest'] = named
5858: (24)                                  else:
5859: (28)                                      self.named = uri_to_dict(self.uri[1:])
```

```
5860: (12)                      else:
5861: (16)                          self.kind = LINK_NAMED
5862: (16)                          self.named = uri_to_dict(self.uri)
5863: (8)                   if obj.is_external:
5864: (12)                      if not self.uri:
5865: (16)                          pass
5866: (12)                      elif self.uri.startswith("file:"):
5867: (16)                          self.file_spec = self.uri[5:]
5868: (16)                          if self.file_spec.startswith("//"):
5869: (20)                              self.file_spec = self.file_spec[2:]
5870: (16)                          self.is_uri = False
5871: (16)                          self.uri = ""
5872: (16)                          self.kind = LINK_LAUNCH
5873: (16)                          ftab = self.file_spec.split("#")
5874: (16)                          if len(ftab) == 2:
5875: (20)                              if ftab[1].startswith("page="):
5876: (24)                                  self.kind = LINK_GOTOR
5877: (24)                                  self.file_spec = ftab[0]
5878: (24)                                  self.page = int(ftab[1].split("&")[0][5:]) - 1
5879: (12)                      elif ":" in self.uri:
5880: (16)                          self.is_uri = True
5881: (16)                          self.kind = LINK_URI
5882: (12)                      else:
5883: (16)                          self.is_uri = True
5884: (16)                          self.kind = LINK_LAUNCH
5885: (8)                   assert isinstance(self.named, dict)
5886: (0)           class Widget:
5887: (4)               '''
5888: (4)               Class describing a PDF form field ("widget")
5889: (4)               '''
5890: (4)               def __init__(self):
5891: (8)                   self.border_color = None
5892: (8)                   self.border_style = "S"
5893: (8)                   self.border_width = 0
5894: (8)                   self.border_dashes = None
5895: (8)                   self.choice_values = None  # choice fields only
5896: (8)                   self.rb_parent = None   # radio buttons only: xref of owning parent
5897: (8)                   self.field_name = None  # field name
5898: (8)                   self.field_label = None # field label
5899: (8)                   self.field_value = None
5900: (8)                   self.field_flags = 0
5901: (8)                   self.field_display = 0
5902: (8)                   self.field_type = 0  # valid range 1 through 7
5903: (8)                   self.field_type_string = None  # field type as string
5904: (8)                   self.fill_color = None
5905: (8)                   self.button_caption = None  # button caption
5906: (8)                   self.is_signed = None  # True / False if signature
5907: (8)                   self.text_color = (0, 0, 0)
5908: (8)                   self.text_font = "Helv"
5909: (8)                   self.text_fontsize = 0
5910: (8)                   self.text_maxlen = 0  # text fields only
5911: (8)                   self.text_format = 0  # text fields only
5912: (8)                   self._text_da = ""  # /DA = default appearance
5913: (8)                   self.script = None  # JavaScript (/A)
5914: (8)                   self.script_stroke = None  # JavaScript (/AA/K)
5915: (8)                   self.script_format = None  # JavaScript (/AA/F)
5916: (8)                   self.script_change = None  # JavaScript (/AA/V)
5917: (8)                   self.script_calc = None  # JavaScript (/AA/C)
5918: (8)                   self.script_blur = None  # JavaScript (/AA/Bl)
5919: (8)                   self.script_focus = None  # JavaScript (/AA/Fo) codespell:ignore
5920: (8)                   self.rect = None  # annot value
5921: (8)                   self.xref = 0  # annot value
5922: (4)               def __repr__(self):
5923: (8)                   return f'Widget:(field_type={self.field_type_string} script=
{self.script})'
5924: (8)                   return "'%s' widget" % (self.field_type_string)
5925: (4)               def _adjust_font(self):
5926: (8)                   """Ensure text_font is from our list and correctly spelled.
5927: (8)                   """
```

```
5928: (8)                    if not self.text_font:
5929: (12)                       self.text_font = "Helv"
5930: (12)                       return
5931: (8)                    valid_fonts = ("Cour", "TiRo", "Helv", "ZaDb")
5932: (8)                    for f in valid_fonts:
5933: (12)                       if self.text_font.lower() == f.lower():
5934: (16)                           self.text_font = f
5935: (16)                           return
5936: (8)                    self.text_font = "Helv"
5937: (8)                    return
5938: (4)                def _checker(self):
5939: (8)                    """Any widget type checks.
5940: (8)                    """
5941: (8)                    if self.field_type not in range(1, 8):
5942: (12)                       raise ValueError("bad field type")
5943: (8)                    if self.field_type == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON and
self.field_value not in (False, "Off") and hasattr(self, "parent"):
5944: (12)                       doc = self.parent.parent
5945: (12)                       kids_type, kids_value = doc.xref_get_key(self.xref, "Parent/Kids")
5946: (12)                       if kids_type == "array":
5947: (16)                           xrefs = tuple(map(int, kids_value[1:-1].replace("0
R","").split()))
5948: (16)                           for xref in xrefs:
5949: (20)                               if xref != self.xref:
5950: (24)                                   doc.xref_set_key(xref, "AS", "/Off")
5951: (4)                def _parse_da(self):
5952: (8)                    """Extract font name, size and color from default appearance string
(/DA object).
5953: (8)                    Equivalent to 'pdf_parse_default_appearance' function in MuPDF's 'pdf-
annot.c'.
5954: (8)                    """
5955: (8)                    if not self._text_da:
5956: (12)                       return
5957: (8)                    font = "Helv"
5958: (8)                    fsize = 0
5959: (8)                    col = (0, 0, 0)
5960: (8)                    dat = self._text_da.split()  # split on any whitespace
5961: (8)                    for i, item in enumerate(dat):
5962: (12)                       if item == "Tf":
5963: (16)                           font = dat[i - 2][1:]
5964: (16)                           fsize = float(dat[i - 1])
5965: (16)                           dat[i] = dat[i-1] = dat[i-2] = ""
5966: (16)                           continue
5967: (12)                       if item == "g":  # unicolor text
5968: (16)                           col = [(float(dat[i - 1]))]
5969: (16)                           dat[i] = dat[i-1] = ""
5970: (16)                           continue
5971: (12)                       if item == "rg":  # RGB colored text
5972: (16)                           col = [float(f) for f in dat[i - 3:i]]
5973: (16)                           dat[i] = dat[i-1] = dat[i-2] = dat[i-3] = ""
5974: (16)                           continue
5975: (8)                    self.text_font = font
5976: (8)                    self.text_fontsize = fsize
5977: (8)                    self.text_color = col
5978: (8)                    self._text_da = ""
5979: (8)                    return
5980: (4)                def _validate(self):
5981: (8)                    """Validate the class entries.
5982: (8)                    """
5983: (8)                    if (self.rect.is_infinite
5984: (12)                       or self.rect.is_empty
5985: (11)                      ):
5986: (12)                       raise ValueError("bad rect")
5987: (8)                    if not self.field_name:
5988: (12)                       raise ValueError("field name missing")
5989: (8)                    if self.field_label == "Unnamed":
5990: (12)                       self.field_label = None
5991: (8)                    CheckColor(self.border_color)
5992: (8)                    CheckColor(self.fill_color)
```

```
(8)                 if not self.text_color:
(12)                    self.text_color = (0, 0, 0)
(8)                 CheckColor(self.text_color)
(8)                 if not self.border_width:
(12)                    self.border_width = 0
(8)                 if not self.text_fontsize:
(12)                    self.text_fontsize = 0
(8)                 self.border_style = self.border_style.upper()[0:1]
(8)                 btn_type = self.field_type in (
(16)                        mupdf.PDF_WIDGET_TYPE_BUTTON,
(16)                        mupdf.PDF_WIDGET_TYPE_CHECKBOX,
(16)                        mupdf.PDF_WIDGET_TYPE_RADIOBUTTON,
(16)                        )
(8)                 if not self.script:
(12)                    self.script = None
(8)                 elif type(self.script) is not str:
(12)                    raise ValueError("script content must be a string")
(8)                 if btn_type or not self.script_calc:
(12)                    self.script_calc = None
(8)                 elif type(self.script_calc) is not str:
(12)                    raise ValueError("script_calc content must be a string")
(8)                 if btn_type or not self.script_change:
(12)                    self.script_change = None
(8)                 elif type(self.script_change) is not str:
(12)                    raise ValueError("script_change content must be a string")
(8)                 if btn_type or not self.script_format:
(12)                    self.script_format = None
(8)                 elif type(self.script_format) is not str:
(12)                    raise ValueError("script_format content must be a string")
(8)                 if btn_type or not self.script_stroke:
(12)                    self.script_stroke = None
(8)                 elif type(self.script_stroke) is not str:
(12)                    raise ValueError("script_stroke content must be a string")
(8)                 if btn_type or not self.script_blur:
(12)                    self.script_blur = None
(8)                 elif type(self.script_blur) is not str:
(12)                    raise ValueError("script_blur content must be a string")
(8)                 if btn_type or not self.script_focus:
(12)                    self.script_focus = None
(8)                 elif type(self.script_focus) is not str:
(12)                    raise ValueError("script_focus content must be a string")
(8)                 self._checker()  # any field_type specific checks
(4)             def button_states(self):
(8)                 """Return the on/off state names for button widgets.
(8)                 A button may have 'normal' or 'pressed down' appearances. While the
'Off'
(8)                 state is usually called like this, the 'On' state is often given a
name
(8)                 relating to the functional context.
(8)                 """
(8)                 if self.field_type not in (2, 5):
(12)                    return None  # no button type
(8)                 if hasattr(self, "parent"):  # field already exists on page
(12)                    doc = self.parent.parent
(8)                 else:
(12)                    return
(8)                 xref = self.xref
(8)                 states = {"normal": None, "down": None}
(8)                 APN = doc.xref_get_key(xref, "AP/N")
(8)                 if APN[0] == "dict":
(12)                    nstates = []
(12)                    APN = APN[1][2:-2]
(12)                    apnt = APN.split("/")[1:]
(12)                    for x in apnt:
(16)                        nstates.append(x.split()[0])
(12)                    states["normal"] = nstates
(8)                 if APN[0] == "xref":
(12)                    nstates = []
(12)                    nxref = int(APN[1].split(" ")[0])
```

```
6060: (12)                      APN = doc.xref_object(nxref)
6061: (12)                      apnt = APN.split("/")[1:]
6062: (12)                      for x in apnt:
6063: (16)                          nstates.append(x.split()[0])
6064: (12)                      states["normal"] = nstates
6065: (8)                   APD = doc.xref_get_key(xref, "AP/D")
6066: (8)                   if APD[0] == "dict":
6067: (12)                      dstates = []
6068: (12)                      APD = APD[1][2:-2]
6069: (12)                      apdt = APD.split("/")[1:]
6070: (12)                      for x in apdt:
6071: (16)                          dstates.append(x.split()[0])
6072: (12)                      states["down"] = dstates
6073: (8)                   if APD[0] == "xref":
6074: (12)                      dstates = []
6075: (12)                      dxref = int(APD[1].split(" ")[0])
6076: (12)                      APD = doc.xref_object(dxref)
6077: (12)                      apdt = APD.split("/")[1:]
6078: (12)                      for x in apdt:
6079: (16)                          dstates.append(x.split()[0])
6080: (12)                      states["down"] = dstates
6081: (8)                   return states
6082: (4)               @property
6083: (4)               def next(self):
6084: (8)                   return self._annot.next
6085: (4)               def on_state(self):
6086: (8)                   """Return the "On" value for button widgets.
6087: (8)                   This is useful for radio buttons mainly. Checkboxes will always return
6088: (8)                   "Yes". Radio buttons will return the string that is unequal to "Off"
6089: (8)                   as returned by method button_states().
6090: (8)                   If the radio button is new / being created, it does not yet have an
6091: (8)                   "On" value. In this case, a warning is shown and True is returned.
6092: (8)                   """
6093: (8)                   if self.field_type not in (2, 5):
6094: (12)                      return None  # no checkbox or radio button
6095: (8)                   if self.field_type == 2:
6096: (12)                      return "Yes"
6097: (8)                   bstate = self.button_states()
6098: (8)                   if bstate is None:
6099: (12)                      bstate = dict()
6100: (8)                   for k in bstate.keys():
6101: (12)                      for v in bstate[k]:
6102: (16)                          if v != "Off":
6103: (20)                              return v
6104: (8)                   message("warning: radio button has no 'On' value.")
6105: (8)                   return True
6106: (4)               def reset(self):
6107: (8)                   """Reset the field value to its default.
6108: (8)                   """
6109: (8)                   TOOLS._reset_widget(self._annot)
6110: (4)               def update(self):
6111: (8)                   """Reflect Python object in the PDF.
6112: (8)                   """
6113: (8)                   self._validate()
6114: (8)                   self._adjust_font()  # ensure valid text_font name
6115: (8)                   self._text_da = ""
6116: (8)                   if   len(self.text_color) == 3:
6117: (12)                      fmt = "{:g} {:g} {:g} rg /{f:s} {s:g} Tf" + self._text_da
6118: (8)                   elif len(self.text_color) == 1:
6119: (12)                      fmt = "{:g} g /{f:s} {s:g} Tf" + self._text_da
6120: (8)                   elif len(self.text_color) == 4:
6121: (12)                      fmt = "{:g} {:g} {:g} {:g} k /{f:s} {s:g} Tf" + self._text_da
6122: (8)                   self._text_da = fmt.format(*self.text_color, f=self.text_font,
6123: (36)                                             s=self.text_fontsize)
6124: (8)                   if self.script_calc:  # there is a "calculation" script:
6125: (12)                      util_ensure_widget_calc(self._annot)
6126: (8)                   TOOLS._save_widget(self._annot, self)
6127: (8)                   self._text_da = ""
6128: (0)           from . import _extra
```

```
(0)            class Outline:
(4)                def __init__(self, ol):
(8)                    self.this = ol
(4)                @property
(4)                def dest(self):
(8)                    '''outline destination details'''
(8)                    return linkDest(self, None, None)
(4)                def destination(self, document):
(8)                    '''
(8)                    Like `dest` property but uses `document` to resolve destinations for
(8)                    kind=LINK_NAMED.
(8)                    '''
(8)                    return linkDest(self, None, document)
(4)                @property
(4)                def down(self):
(8)                    ol = self.this
(8)                    down_ol = ol.down()
(8)                    if not down_ol.m_internal:
(12)                       return
(8)                    return Outline(down_ol)
(4)                @property
(4)                def is_external(self):
(8)                    if g_use_extra:
(12)                       return _extra.Outline_is_external( self.this)
(8)                    ol = self.this
(8)                    if not ol.m_internal:
(12)                       return False
(8)                    uri = ol.m_internal.uri if 1 else ol.uri()
(8)                    if uri is None:
(12)                       return False
(8)                    return mupdf.fz_is_external_link(uri)
(4)                @property
(4)                def is_open(self):
(8)                    if 1:
(12)                       return self.this.m_internal.is_open
(8)                    return self.this.is_open()
(4)                @property
(4)                def next(self):
(8)                    ol = self.this
(8)                    next_ol = ol.next()
(8)                    if not next_ol.m_internal:
(12)                       return
(8)                    return Outline(next_ol)
(4)                @property
(4)                def page(self):
(8)                    if 1:
(12)                       return self.this.m_internal.page.page
(8)                    return self.this.page().page
(4)                @property
(4)                def title(self):
(8)                    return self.this.m_internal.title
(4)                @property
(4)                def uri(self):
(8)                    ol = self.this
(8)                    if not ol.m_internal:
(12)                       return None
(8)                    return ol.m_internal.uri
(4)                @property
(4)                def x(self):
(8)                    return self.this.m_internal.x
(4)                @property
(4)                def y(self):
(8)                    return self.this.m_internal.y
(4)                __slots__ = [ 'this']
(0)            def _make_PdfFilterOptions(
(8)                    recurse=0,
(8)                    instance_forms=0,
(8)                    ascii=0,
(8)                    no_update=0,
```

```
6198: (8)                   sanitize=0,
6199: (8)                   sopts=None,
6200: (8)                   ):
6201: (4)               '''
6202: (4)               Returns a mupdf.PdfFilterOptions instance.
6203: (4)               '''
6204: (4)               filter_ = mupdf.PdfFilterOptions()
6205: (4)               filter_.recurse = recurse
6206: (4)               filter_.instance_forms = instance_forms
6207: (4)               filter_.ascii = ascii
6208: (4)               filter_.no_update = no_update
6209: (4)               if sanitize:
6210: (8)                   if sopts:
6211: (12)                      assert isinstance(sopts, mupdf.PdfSanitizeFilterOptions)
6212: (8)                   else:
6213: (12)                      sopts = mupdf.PdfSanitizeFilterOptions()
6214: (8)                   class Factory(mupdf.PdfFilterFactory2):
6215: (12)                      def __init__(self):
6216: (16)                          super().__init__()
6217: (16)                          self.use_virtual_filter()
6218: (16)                          self.sopts = sopts
6219: (12)                      def filter(self, ctx, doc, chain, struct_parents, transform,
options):
6220: (16)                          if 0:
6221: (20)                              log(f'sanitize filter.filter():')
6222: (20)                              log(f'    {self=}')
6223: (20)                              log(f'    {ctx=}')
6224: (20)                              log(f'    {doc=}')
6225: (20)                              log(f'    {chain=}')
6226: (20)                              log(f'    {struct_parents=}')
6227: (20)                              log(f'    {transform=}')
6228: (20)                              log(f'    {options=}')
6229: (20)                              log(f'    {self.sopts.internal()=}')
6230: (16)                          return mupdf.ll_pdf_new_sanitize_filter(
6231: (24)                                  doc,
6232: (24)                                  chain,
6233: (24)                                  struct_parents,
6234: (24)                                  transform,
6235: (24)                                  options,
6236: (24)                                  self.sopts.internal(),
6237: (24)                                  )
6238: (8)                   factory = Factory()
6239: (8)                   filter_.add_factory(factory.internal())
6240: (8)                   filter_._factory = factory
6241: (4)               return filter_
6242: (0)           class Page:
6243: (4)               def __init__(self, page, document):
6244: (8)                   assert isinstance(page, (mupdf.FzPage, mupdf.PdfPage)), f'page is:
{page}'
6245: (8)                   self.this = page
6246: (8)                   self.thisown = True
6247: (8)                   self.last_point = None
6248: (8)                   self.draw_cont = ''
6249: (8)                   self._annot_refs = dict()
6250: (8)                   self.parent = document
6251: (8)                   if page.m_internal:
6252: (12)                      if isinstance( page, mupdf.PdfPage):
6253: (16)                          self.number = page.m_internal.super.number
6254: (12)                      else:
6255: (16)                          self.number = page.m_internal.number
6256: (8)                   else:
6257: (12)                      self.number = None
6258: (4)               def __repr__(self):
6259: (8)                   return self.__str__()
6260: (8)                   CheckParent(self)
6261: (8)                   x = self.parent.name
6262: (8)                   if self.parent.stream is not None:
6263: (12)                      x = "<memory, doc# %i>" % (self.parent._graft_id,)
6264: (8)                   if x == "":
```

```
6265: (12)                      x = "<new PDF, doc# %i>" % self.parent._graft_id
6266: (8)                   return "page %s of %s" % (self.number, x)
6267: (4)               def __str__(self):
6268: (8)                   parent = getattr(self, 'parent', None)
6269: (8)                   if isinstance(self.this.m_internal, mupdf.pdf_page):
6270: (12)                      number = self.this.m_internal.super.number
6271: (8)                   else:
6272: (12)                      number = self.this.m_internal.number
6273: (8)                   ret = f'page {number}'
6274: (8)                   if parent:
6275: (12)                      x = self.parent.name
6276: (12)                      if self.parent.stream is not None:
6277: (16)                          x = "<memory, doc# %i>" % (self.parent._graft_id,)
6278: (12)                      if x == "":
6279: (16)                          x = "<new PDF, doc# %i>" % self.parent._graft_id
6280: (12)                      ret += f' of {x}'
6281: (8)                   return ret
6282: (4)               def _add_caret_annot(self, point):
6283: (8)                   if g_use_extra:
6284: (12)                      annot = extra._add_caret_annot( self.this,
JM_point_from_py(point))
6285: (8)                   else:
6286: (12)                      page = self._pdf_page()
6287: (12)                      annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_CARET)
6288: (12)                      if point:
6289: (16)                          p = JM_point_from_py(point)
6290: (16)                          r = mupdf.pdf_annot_rect(annot)
6291: (16)                          r = mupdf.FzRect(p.x, p.y, p.x + r.x1 - r.x0, p.y + r.y1 -
r.y0)
6292: (16)                          mupdf.pdf_set_annot_rect(annot, r)
6293: (12)                      mupdf.pdf_update_annot(annot)
6294: (12)                      JM_add_annot_id(annot, "A")
6295: (8)                   return annot
6296: (4)               def _add_file_annot(self, point, buffer_, filename, ufilename=None,
desc=None, icon=None):
6297: (8)                   page = self._pdf_page()
6298: (8)                   uf = ufilename if ufilename else filename
6299: (8)                   d = desc if desc else filename
6300: (8)                   p = JM_point_from_py(point)
6301: (8)                   filebuf = JM_BufferFromBytes(buffer_)
6302: (8)                   if not filebuf.m_internal:
6303: (12)                      raise TypeError( MSG_BAD_BUFFER)
6304: (8)                   annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_FILE_ATTACHMENT)
6305: (8)                   r = mupdf.pdf_annot_rect(annot)
6306: (8)                   r = mupdf.fz_make_rect(p.x, p.y, p.x + r.x1 - r.x0, p.y + r.y1 - r.y0)
6307: (8)                   mupdf.pdf_set_annot_rect(annot, r)
6308: (8)                   flags = mupdf.PDF_ANNOT_IS_PRINT
6309: (8)                   mupdf.pdf_set_annot_flags(annot, flags)
6310: (8)                   if icon:
6311: (12)                      mupdf.pdf_set_annot_icon_name(annot, icon)
6312: (8)                   val = JM_embed_file(page.doc(), filebuf, filename, uf, d, 1)
6313: (8)                   mupdf.pdf_dict_put(mupdf.pdf_annot_obj(annot), PDF_NAME('FS'), val)
6314: (8)                   mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(annot),
PDF_NAME('Contents'), filename)
6315: (8)                   mupdf.pdf_update_annot(annot)
6316: (8)                   mupdf.pdf_set_annot_rect(annot, r)
6317: (8)                   mupdf.pdf_set_annot_flags(annot, flags)
6318: (8)                   JM_add_annot_id(annot, "A")
6319: (8)                   return Annot(annot)
6320: (4)               def _add_freetext_annot(
6321: (12)                      self, rect,
6322: (12)                      text,
6323: (12)                      fontsize=11,
6324: (12)                      fontname=None,
6325: (12)                      text_color=None,
6326: (12)                      fill_color=None,
6327: (12)                      border_color=None,
6328: (12)                      align=0,
6329: (12)                      rotate=0,
```

```
6330: (12)                      ):
6331: (8)                   page = self._pdf_page()
6332: (8)                   nfcol, fcol = JM_color_FromSequence(fill_color)
6333: (8)                   ntcol, tcol = JM_color_FromSequence(text_color)
6334: (8)                   r = JM_rect_from_py(rect)
6335: (8)                   if mupdf.fz_is_infinite_rect(r) or mupdf.fz_is_empty_rect(r):
6336: (12)                      raise ValueError( MSG_BAD_RECT)
6337: (8)                   annot = mupdf.pdf_create_annot( page, mupdf.PDF_ANNOT_FREE_TEXT)
6338: (8)                   annot_obj = mupdf.pdf_annot_obj( annot)
6339: (8)                   mupdf.pdf_set_annot_contents( annot, text)
6340: (8)                   mupdf.pdf_set_annot_rect( annot, r)
6341: (8)                   mupdf.pdf_dict_put_int( annot_obj, PDF_NAME('Rotate'), rotate)
6342: (8)                   mupdf.pdf_dict_put_int( annot_obj, PDF_NAME('Q'), align)
6343: (8)                   if nfcol > 0:
6344: (12)                      mupdf.pdf_set_annot_color( annot, fcol[:nfcol])
6345: (8)                   JM_make_annot_DA(annot, ntcol, tcol, fontname, fontsize)
6346: (8)                   mupdf.pdf_update_annot( annot)
6347: (8)                   JM_add_annot_id(annot, "A")
6348: (8)                   val = Annot(annot)
6349: (8)                   ap = val._getAP()
6350: (8)                   BT = ap.find(b"BT")
6351: (8)                   ET = ap.rfind(b"ET") + 2
6352: (8)                   ap = ap[BT:ET]
6353: (8)                   w = rect[2]-rect[0]
6354: (8)                   h = rect[3]-rect[1]
6355: (8)                   if rotate in (90, -90, 270):
6356: (12)                      w, h = h, w
6357: (8)                   re = f"0 0 {_format_g((w, h))} re".encode()
6358: (8)                   ap = re + b"\nW\nn\n" + ap
6359: (8)                   ope = None
6360: (8)                   bwidth = b""
6361: (8)                   fill_string = ColorCode(fill_color, "f").encode()
6362: (8)                   if fill_string:
6363: (12)                      fill_string += b"\n"
6364: (12)                      ope = b"f"
6365: (8)                   stroke_string = ColorCode(border_color, "c").encode()
6366: (8)                   if stroke_string:
6367: (12)                      stroke_string += b"\n"
6368: (12)                      bwidth = b"1 w\n"
6369: (12)                      ope = b"S"
6370: (8)                   if fill_string and stroke_string:
6371: (12)                      ope = b"B"
6372: (8)                   if ope is not None:
6373: (12)                      ap = bwidth + fill_string + stroke_string + re + b"\n" + ope +
b"\n" + ap
6374: (8)                   val._setAP(ap)
6375: (8)                   return val
6376: (4)               def _add_ink_annot(self, list):
6377: (8)                   page = _as_pdf_page(self.this)
6378: (8)                   if not PySequence_Check(list):
6379: (12)                      raise ValueError( MSG_BAD_ARG_INK_ANNOT)
6380: (8)                   ctm = mupdf.FzMatrix()
6381: (8)                   mupdf.pdf_page_transform(page, mupdf.FzRect(0), ctm)
6382: (8)                   inv_ctm = mupdf.fz_invert_matrix(ctm)
6383: (8)                   annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_INK)
6384: (8)                   annot_obj = mupdf.pdf_annot_obj(annot)
6385: (8)                   n0 = len(list)
6386: (8)                   inklist = mupdf.pdf_new_array(page.doc(), n0)
6387: (8)                   for j in range(n0):
6388: (12)                      sublist = list[j]
6389: (12)                      n1 = len(sublist)
6390: (12)                      stroke = mupdf.pdf_new_array(page.doc(), 2 * n1)
6391: (12)                      for i in range(n1):
6392: (16)                          p = sublist[i]
6393: (16)                          if not PySequence_Check(p) or PySequence_Size(p) != 2:
6394: (20)                              raise ValueError( MSG_BAD_ARG_INK_ANNOT)
6395: (16)                          point = mupdf.fz_transform_point(JM_point_from_py(p), inv_ctm)
6396: (16)                          mupdf.pdf_array_push_real(stroke, point.x)
6397: (16)                          mupdf.pdf_array_push_real(stroke, point.y)
```

```
6398: (12)                      mupdf.pdf_array_push(inklist, stroke)
6399: (8)                   mupdf.pdf_dict_put(annot_obj, PDF_NAME('InkList'), inklist)
6400: (8)                   mupdf.pdf_update_annot(annot)
6401: (8)                   JM_add_annot_id(annot, "A")
6402: (8)                   return Annot(annot)
6403: (4)               def _add_line_annot(self, p1, p2):
6404: (8)                   page = self._pdf_page()
6405: (8)                   annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_LINE)
6406: (8)                   a = JM_point_from_py(p1)
6407: (8)                   b = JM_point_from_py(p2)
6408: (8)                   mupdf.pdf_set_annot_line(annot, a, b)
6409: (8)                   mupdf.pdf_update_annot(annot)
6410: (8)                   JM_add_annot_id(annot, "A")
6411: (8)                   assert annot.m_internal
6412: (8)                   return Annot(annot)
6413: (4)               def _add_multiline(self, points, annot_type):
6414: (8)                   page = self._pdf_page()
6415: (8)                   if len(points) < 2:
6416: (12)                      raise ValueError( MSG_BAD_ARG_POINTS)
6417: (8)                   annot = mupdf.pdf_create_annot(page, annot_type)
6418: (8)                   for p in points:
6419: (12)                      if (PySequence_Size(p) != 2):
6420: (16)                          raise ValueError( MSG_BAD_ARG_POINTS)
6421: (12)                      point = JM_point_from_py(p)
6422: (12)                      mupdf.pdf_add_annot_vertex(annot, point)
6423: (8)                   mupdf.pdf_update_annot(annot)
6424: (8)                   JM_add_annot_id(annot, "A")
6425: (8)                   return Annot(annot)
6426: (4)               def _add_redact_annot(self, quad, text=None, da_str=None, align=0,
fill=None, text_color=None):
6427: (8)                   page = self._pdf_page()
6428: (8)                   fcol = [ 1, 1, 1, 0]
6429: (8)                   nfcol = 0
6430: (8)                   annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_REDACT)
6431: (8)                   q = JM_quad_from_py(quad)
6432: (8)                   r = mupdf.fz_rect_from_quad(q)
6433: (8)                   mupdf.pdf_set_annot_rect(annot, r)
6434: (8)                   if fill:
6435: (12)                      nfcol, fcol = JM_color_FromSequence(fill)
6436: (12)                      arr = mupdf.pdf_new_array(page.doc(), nfcol)
6437: (12)                      for i in range(nfcol):
6438: (16)                          mupdf.pdf_array_push_real(arr, fcol[i])
6439: (12)                      mupdf.pdf_dict_put(mupdf.pdf_annot_obj(annot), PDF_NAME('IC'),
arr)
6440: (8)                   if text:
6441: (12)                      mupdf.pdf_dict_puts(
6442: (20)                              mupdf.pdf_annot_obj(annot),
6443: (20)                              "OverlayText",
6444: (20)                              mupdf.pdf_new_text_string(text),
6445: (20)                              )
6446: (12)                      mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(annot),
PDF_NAME('DA'), da_str)
6447: (12)                      mupdf.pdf_dict_put_int(mupdf.pdf_annot_obj(annot), PDF_NAME('Q'),
align)
6448: (8)                   mupdf.pdf_update_annot(annot)
6449: (8)                   JM_add_annot_id(annot, "A")
6450: (8)                   annot = mupdf.ll_pdf_keep_annot(annot.m_internal)
6451: (8)                   annot = mupdf.PdfAnnot( annot)
6452: (8)                   return Annot(annot)
6453: (4)               def _add_square_or_circle(self, rect, annot_type):
6454: (8)                   page = self._pdf_page()
6455: (8)                   r = JM_rect_from_py(rect)
6456: (8)                   if mupdf.fz_is_infinite_rect(r) or mupdf.fz_is_empty_rect(r):
6457: (12)                      raise ValueError( MSG_BAD_RECT)
6458: (8)                   annot = mupdf.pdf_create_annot(page, annot_type)
6459: (8)                   mupdf.pdf_set_annot_rect(annot, r)
6460: (8)                   mupdf.pdf_update_annot(annot)
6461: (8)                   JM_add_annot_id(annot, "A")
6462: (8)                   assert annot.m_internal
```

```
6463: (8)                    return Annot(annot)
6464: (4)                def _add_stamp_annot(self, rect, stamp=0):
6465: (8)                    page = self._pdf_page()
6466: (8)                    stamp_id = [
6467: (16)                           PDF_NAME('Approved'),
6468: (16)                           PDF_NAME('AsIs'),
6469: (16)                           PDF_NAME('Confidential'),
6470: (16)                           PDF_NAME('Departmental'),
6471: (16)                           PDF_NAME('Experimental'),
6472: (16)                           PDF_NAME('Expired'),
6473: (16)                           PDF_NAME('Final'),
6474: (16)                           PDF_NAME('ForComment'),
6475: (16)                           PDF_NAME('ForPublicRelease'),
6476: (16)                           PDF_NAME('NotApproved'),
6477: (16)                           PDF_NAME('NotForPublicRelease'),
6478: (16)                           PDF_NAME('Sold'),
6479: (16)                           PDF_NAME('TopSecret'),
6480: (16)                           PDF_NAME('Draft'),
6481: (16)                           ]
6482: (8)                    n = len(stamp_id)
6483: (8)                    name = stamp_id[0]
6484: (8)                    r = JM_rect_from_py(rect)
6485: (8)                    if mupdf.fz_is_infinite_rect(r) or mupdf.fz_is_empty_rect(r):
6486: (12)                       raise ValueError( MSG_BAD_RECT)
6487: (8)                    if _INRANGE(stamp, 0, n-1):
6488: (12)                       name = stamp_id[stamp]
6489: (8)                    annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_STAMP)
6490: (8)                    mupdf.pdf_set_annot_rect(annot, r)
6491: (8)                    try:
6492: (12)                       n = PDF_NAME('Name')
6493: (12)                       mupdf.pdf_dict_put(mupdf.pdf_annot_obj(annot), PDF_NAME('Name'),
name)
6494: (8)                    except Exception:
6495: (12)                       if g_exceptions_verbose:    exception_info()
6496: (12)                       raise
6497: (8)                    mupdf.pdf_set_annot_contents(
6498: (16)                           annot,
6499: (16)                           mupdf.pdf_dict_get_name(mupdf.pdf_annot_obj(annot),
PDF_NAME('Name')),
6500: (16)                           )
6501: (8)                    mupdf.pdf_update_annot(annot)
6502: (8)                    JM_add_annot_id(annot, "A")
6503: (8)                    return Annot(annot)
6504: (4)                def _add_text_annot(self, point, text, icon=None):
6505: (8)                    page = self._pdf_page()
6506: (8)                    p = JM_point_from_py( point)
6507: (8)                    annot = mupdf.pdf_create_annot(page, mupdf.PDF_ANNOT_TEXT)
6508: (8)                    r = mupdf.pdf_annot_rect(annot)
6509: (8)                    r = mupdf.fz_make_rect(p.x, p.y, p.x + r.x1 - r.x0, p.y + r.y1 - r.y0)
6510: (8)                    mupdf.pdf_set_annot_rect(annot, r)
6511: (8)                    mupdf.pdf_set_annot_contents(annot, text)
6512: (8)                    if icon:
6513: (12)                       mupdf.pdf_set_annot_icon_name(annot, icon)
6514: (8)                    mupdf.pdf_update_annot(annot)
6515: (8)                    JM_add_annot_id(annot, "A")
6516: (8)                    return Annot(annot)
6517: (4)                def _add_text_marker(self, quads, annot_type):
6518: (8)                    CheckParent(self)
6519: (8)                    if not self.parent.is_pdf:
6520: (12)                       raise ValueError("is no PDF")
6521: (8)                    val = Page__add_text_marker(self, quads, annot_type)
6522: (8)                    if not val:
6523: (12)                       return None
6524: (8)                    val.parent = weakref.proxy(self)
6525: (8)                    self._annot_refs[id(val)] = val
6526: (8)                    return val
6527: (4)                def _addAnnot_FromString(self, linklist):
6528: (8)                    """Add links from list of object sources."""
6529: (8)                    CheckParent(self)
```

```
6530: (8)                   if g_use_extra:
6531: (12)                      self.__class__._addAnnot_FromString =
extra.Page_addAnnot_FromString
6532: (12)                      return extra.Page_addAnnot_FromString( self.this, linklist)
6533: (8)                   page = _as_pdf_page(self.this)
6534: (8)                   lcount = len(linklist)  # link count
6535: (8)                   if lcount < 1:
6536: (12)                      return
6537: (8)                   i = -1
6538: (8)                   if not isinstance(linklist, tuple):
6539: (12)                      raise ValueError( "bad 'linklist' argument")
6540: (8)                   if not mupdf.pdf_dict_get( page.obj(), PDF_NAME('Annots')).m_internal:
6541: (12)                      mupdf.pdf_dict_put_array( page.obj(), PDF_NAME('Annots'), lcount)
6542: (8)                   annots = mupdf.pdf_dict_get( page.obj(), PDF_NAME('Annots'))
6543: (8)                   assert annots.m_internal, f'{lcount=} {annots.m_internal=}'
6544: (8)                   for i in range(lcount):
6545: (12)                      txtpy = linklist[i]
6546: (12)                      text = JM_StrAsChar(txtpy)
6547: (12)                      if not text:
6548: (16)                          message("skipping bad link / annot item %i.", i)
6549: (16)                          continue
6550: (12)                      try:
6551: (16)                          annot = mupdf.pdf_add_object( page.doc(), JM_pdf_obj_from_str(
page.doc(), text))
6552: (16)                          ind_obj = mupdf.pdf_new_indirect( page.doc(),
mupdf.pdf_to_num( annot), 0)
6553: (16)                          mupdf.pdf_array_push( annots, ind_obj)
6554: (12)                      except Exception:
6555: (16)                          if g_exceptions_verbose:    exception_info()
6556: (16)                          message("skipping bad link / annot item %i.\n" % i)
6557: (4)               def _addWidget(self, field_type, field_name):
6558: (8)                   page = self._pdf_page()
6559: (8)                   pdf = page.doc()
6560: (8)                   annot = JM_create_widget(pdf, page, field_type, field_name)
6561: (8)                   if not annot.m_internal:
6562: (12)                      raise RuntimeError( "cannot create widget")
6563: (8)                   JM_add_annot_id(annot, "W")
6564: (8)                   return Annot(annot)
6565: (4)               def _apply_redactions(self, text, images, graphics):
6566: (8)                   page = self._pdf_page()
6567: (8)                   opts = mupdf.PdfRedactOptions()
6568: (8)                   opts.black_boxes = 0  # no black boxes
6569: (8)                   opts.text = text  # how to treat text
6570: (8)                   opts.image_method = images  # how to treat images
6571: (8)                   opts.line_art = graphics  # how to treat vector graphics
6572: (8)                   success = mupdf.pdf_redact_page(page.doc(), page, opts)
6573: (8)                   return success
6574: (4)               def _erase(self):
6575: (8)                   self._reset_annot_refs()
6576: (8)                   try:
6577: (12)                      self.parent._forget_page(self)
6578: (8)                   except Exception:
6579: (12)                      exception_info()
6580: (12)                      pass
6581: (8)                   self.parent = None
6582: (8)                   self.thisown = False
6583: (8)                   self.number = None
6584: (8)                   self.this = None
6585: (4)               def _count_q_balance(self):
6586: (8)                   """Count missing graphic state pushs and pops.
6587: (8)                   Returns:
6588: (12)                      A pair of integers (push, pop). Push is the number of missing
6589: (12)                      PDF "q" commands, pop is the number of "Q" commands.
6590: (12)                      A balanced graphics state for the page will be reached if its
6591: (12)                      /Contents is prepended with 'push' copies of string "q\n"
6592: (12)                      and appended with 'pop' copies of "\nQ".
6593: (8)                   """
6594: (8)                   page = _as_pdf_page(self)  # need the underlying PDF page
6595: (8)                   res = mupdf.pdf_dict_get(  # access /Resources
```

```
6596: (12)                      page.obj(),
6597: (12)                      mupdf.PDF_ENUM_NAME_Resources,
6598: (8)                   )
6599: (8)                   cont = mupdf.pdf_dict_get(  # access /Contents
6600: (12)                      page.obj(),
6601: (12)                      mupdf.PDF_ENUM_NAME_Contents,
6602: (8)                   )
6603: (8)                   pdf = _as_pdf_document(self.parent)  # need underlying PDF document
6604: (8)                   return mupdf.pdf_count_q_balance_outparams_fn(pdf, res, cont)
6605: (4)               def _get_optional_content(self, oc: OptInt) -> OptStr:
6606: (8)                   if oc is None or oc == 0:
6607: (12)                      return None
6608: (8)                   doc = self.parent
6609: (8)                   check = doc.xref_object(oc, compressed=True)
6610: (8)                   if not ("/Type/OCG" in check or "/Type/OCMD" in check):
6611: (12)                      raise ValueError("bad optional content: 'oc'")
6612: (8)                   props = {}
6613: (8)                   for p, x in self._get_resource_properties():
6614: (12)                      props[x] = p
6615: (8)                   if oc in props.keys():
6616: (12)                      return props[oc]
6617: (8)                   i = 0
6618: (8)                   mc = "MC%i" % i
6619: (8)                   while mc in props.values():
6620: (12)                      i += 1
6621: (12)                      mc = "MC%i" % i
6622: (8)                   self._set_resource_property(mc, oc)
6623: (8)                   return mc
6624: (4)               def _get_resource_properties(self):
6625: (8)                   '''
6626: (8)                   page list Resource/Properties
6627: (8)                   '''
6628: (8)                   page = self._pdf_page()
6629: (8)                   rc = JM_get_resource_properties(page.obj())
6630: (8)                   return rc
6631: (4)               def _get_textpage(self, clip=None, flags=0, matrix=None):
6632: (8)                   if g_use_extra:
6633: (12)                      ll_tpage = extra.page_get_textpage(self.this, clip, flags, matrix)
6634: (12)                      tpage = mupdf.FzStextPage(ll_tpage)
6635: (12)                      return tpage
6636: (8)                   page = self.this
6637: (8)                   options = mupdf.FzStextOptions(flags)
6638: (8)                   rect = JM_rect_from_py(clip)
6639: (8)                   rect = mupdf.fz_bound_page(page) if clip is None else
JM_rect_from_py(clip)
6640: (8)                   ctm = JM_matrix_from_py(matrix)
6641: (8)                   tpage = mupdf.FzStextPage(rect)
6642: (8)                   dev = mupdf.fz_new_stext_device(tpage, options)
6643: (8)                   if _globals.no_device_caching:
6644: (12)                      mupdf.fz_enable_device_hints( dev, mupdf.FZ_NO_CACHE)
6645: (8)                   if isinstance(page, mupdf.FzPage):
6646: (12)                      pass
6647: (8)                   elif isinstance(page, mupdf.PdfPage):
6648: (12)                      page = page.super()
6649: (8)                   else:
6650: (12)                      assert 0, f'Unrecognised {type(page)=}'
6651: (8)                   mupdf.fz_run_page(page, dev, ctm, mupdf.FzCookie())
6652: (8)                   mupdf.fz_close_device(dev)
6653: (8)                   return tpage
6654: (4)               def _insert_image(self,
6655: (12)                      filename=None, pixmap=None, stream=None, imask=None, clip=None,
6656: (12)                      overlay=1, rotate=0, keep_proportion=1, oc=0, width=0, height=0,
6657: (12)                      xref=0, alpha=-1, _imgname=None, digests=None
6658: (12)                      ):
6659: (8)                   maskbuf = mupdf.FzBuffer()
6660: (8)                   page = self._pdf_page()
6661: (8)                   pdf = page.doc()
6662: (8)                   w = width
6663: (8)                   h = height
```

```
6664: (8)                   img_xref = xref
6665: (8)                   rc_digest = 0
6666: (8)                   do_process_pixmap = 1
6667: (8)                   do_process_stream = 1
6668: (8)                   do_have_imask = 1
6669: (8)                   do_have_image = 1
6670: (8)                   do_have_xref = 1
6671: (8)                   if xref > 0:
6672: (12)                      ref = mupdf.pdf_new_indirect(pdf, xref, 0)
6673: (12)                      w = mupdf.pdf_to_int( mupdf.pdf_dict_geta( ref, PDF_NAME('Width'),
PDF_NAME('W')))
6674: (12)                      h = mupdf.pdf_to_int( mupdf.pdf_dict_geta( ref,
PDF_NAME('Height'), PDF_NAME('H')))
6675: (12)                      if w + h == 0:
6676: (16)                          raise ValueError( MSG_IS_NO_IMAGE)
6677: (12)                      do_process_pixmap = 0
6678: (12)                      do_process_stream = 0
6679: (12)                      do_have_imask = 0
6680: (12)                      do_have_image = 0
6681: (8)                   else:
6682: (12)                      if stream:
6683: (16)                          imgbuf = JM_BufferFromBytes(stream)
6684: (16)                          do_process_pixmap = 0
6685: (12)                      else:
6686: (16)                          if filename:
6687: (20)                              imgbuf = mupdf.fz_read_file(filename)
6688: (20)                              do_process_pixmap = 0
6689: (8)                   if do_process_pixmap:
6690: (12)                      arg_pix = pixmap.this
6691: (12)                      w = arg_pix.w()
6692: (12)                      h = arg_pix.h()
6693: (12)                      digest = mupdf.fz_md5_pixmap2(arg_pix)
6694: (12)                      md5_py = digest
6695: (12)                      temp = digests.get(md5_py, None)
6696: (12)                      if temp is not None:
6697: (16)                          img_xref = temp
6698: (16)                          ref = mupdf.pdf_new_indirect(page.doc(), img_xref, 0)
6699: (16)                          do_process_stream = 0
6700: (16)                          do_have_imask = 0
6701: (16)                          do_have_image = 0
6702: (12)                      else:
6703: (16)                          if arg_pix.alpha() == 0:
6704: (20)                              image = mupdf.fz_new_image_from_pixmap(arg_pix,
mupdf.FzImage())
6705: (16)                          else:
6706: (20)                              pm = mupdf.fz_convert_pixmap(
6707: (28)                                      arg_pix,
6708: (28)                                      mupdf.FzColorspace(),
6709: (28)                                      mupdf.FzColorspace(),
6710: (28)                                      mupdf.FzDefaultColorspaces(None),
6711: (28)                                      mupdf.FzColorParams(),
6712: (28)                                      1,
6713: (28)                                      )
6714: (20)                              pm.alpha = 0
6715: (20)                              pm.colorspace = None
6716: (20)                              mask = mupdf.fz_new_image_from_pixmap(pm, mupdf.FzImage())
6717: (20)                              image = mupdf.fz_new_image_from_pixmap(arg_pix, mask)
6718: (16)                          do_process_stream = 0
6719: (16)                          do_have_imask = 0
6720: (8)                   if do_process_stream:
6721: (12)                      state = mupdf.FzMd5()
6722: (12)                      if mupdf_cppyy:
6723: (16)                          mupdf.fz_md5_update_buffer( state, imgbuf)
6724: (12)                      else:
6725: (16)                          mupdf.fz_md5_update(state, imgbuf.m_internal.data,
imgbuf.m_internal.len)
6726: (12)                      if imask:
6727: (16)                          maskbuf = JM_BufferFromBytes(imask)
6728: (16)                          if mupdf_cppyy:
```

```
6729: (20)                                mupdf.fz_md5_update_buffer( state, maskbuf)
6730: (16)                            else:
6731: (20)                                mupdf.fz_md5_update(state, maskbuf.m_internal.data,
maskbuf.m_internal.len)
6732: (12)                        digest = mupdf.fz_md5_final2(state)
6733: (12)                        md5_py = bytes(digest)
6734: (12)                        temp = digests.get(md5_py, None)
6735: (12)                        if temp is not None:
6736: (16)                            img_xref = temp
6737: (16)                            ref = mupdf.pdf_new_indirect(page.doc(), img_xref, 0)
6738: (16)                            w = mupdf.pdf_to_int( mupdf.pdf_dict_geta( ref,
PDF_NAME('Width'), PDF_NAME('W')))
6739: (16)                            h = mupdf.pdf_to_int( mupdf.pdf_dict_geta( ref,
PDF_NAME('Height'), PDF_NAME('H')))
6740: (16)                            do_have_imask = 0
6741: (16)                            do_have_image = 0
6742: (12)                        else:
6743: (16)                            image = mupdf.fz_new_image_from_buffer(imgbuf)
6744: (16)                            w = image.w()
6745: (16)                            h = image.h()
6746: (16)                            if not imask:
6747: (20)                                do_have_imask = 0
6748: (8)                    if do_have_imask:
6749: (12)                        if mupdf_version_tuple >= (1, 24):
6750: (16)                            cbuf1 = mupdf.fz_compressed_image_buffer( image)
6751: (16)                            if not cbuf1.m_internal:
6752: (20)                                raise ValueError( "uncompressed image cannot have mask")
6753: (16)                            bpc = image.bpc()
6754: (16)                            colorspace = image.colorspace()
6755: (16)                            xres, yres = mupdf.fz_image_resolution(image)
6756: (16)                            mask = mupdf.fz_new_image_from_buffer(maskbuf)
6757: (16)                            image = mupdf.fz_new_image_from_compressed_buffer2(
6758: (24)                                    w,
6759: (24)                                    h,
6760: (24)                                    bpc,
6761: (24)                                    colorspace,
6762: (24)                                    xres,
6763: (24)                                    yres,
6764: (24)                                    1,  # interpolate
6765: (24)                                    0,  # imagemask,
6766: (24)                                    list(), # decode
6767: (24)                                    list(), # colorkey
6768: (24)                                    cbuf1,
6769: (24)                                    mask,
6770: (24)                                    )
6771: (12)                        else:
6772: (16)                            cbuf1 = mupdf.ll_fz_compressed_image_buffer( image.m_internal)
6773: (16)                            if not cbuf1:
6774: (20)                                raise ValueError( "uncompressed image cannot have mask")
6775: (16)                            bpc = image.bpc()
6776: (16)                            colorspace = image.colorspace()
6777: (16)                            xres, yres = mupdf.fz_image_resolution(image)
6778: (16)                            mask = mupdf.fz_new_image_from_buffer(maskbuf)
6779: (16)                            zimg = extra.fz_new_image_from_compressed_buffer(
6780: (24)                                    w,
6781: (24)                                    h,
6782: (24)                                    bpc,
6783: (24)                                    colorspace.m_internal,
6784: (24)                                    xres,
6785: (24)                                    yres,
6786: (24)                                    1,  # interpolate
6787: (24)                                    0,  # imagemask,
6788: (24)                                    cbuf1,
6789: (24)                                    mask.m_internal,
6790: (24)                                    )
6791: (16)                            zimg = mupdf.FzImage(zimg)
6792: (16)                            image.m_internal = None
6793: (16)                            image = zimg
6794: (8)                    if do_have_image:
```

```
6795: (12)                      ref = mupdf.pdf_add_image(pdf, image)
6796: (12)                      if oc:
6797: (16)                          JM_add_oc_object(pdf, ref, oc)
6798: (12)                      img_xref = mupdf.pdf_to_num(ref)
6799: (12)                      digests[md5_py] = img_xref
6800: (12)                      rc_digest = 1
6801: (8)                   if do_have_xref:
6802: (12)                      resources = mupdf.pdf_dict_get_inheritable(page.obj(),
PDF_NAME('Resources'))
6803: (12)                      if not resources.m_internal:
6804: (16)                          resources = mupdf.pdf_dict_put_dict(page.obj(),
PDF_NAME('Resources'), 2)
6805: (12)                      xobject = mupdf.pdf_dict_get(resources, PDF_NAME('XObject'))
6806: (12)                      if not xobject.m_internal:
6807: (16)                          xobject = mupdf.pdf_dict_put_dict(resources,
PDF_NAME('XObject'), 2)
6808: (12)                      mat = calc_image_matrix(w, h, clip, rotate, keep_proportion)
6809: (12)                      mupdf.pdf_dict_puts(xobject, _imgname, ref)
6810: (12)                      nres = mupdf.fz_new_buffer(50)
6811: (12)                      s = f"\nq\n{_format_g((mat.a, mat.b, mat.c, mat.d, mat.e, mat.f))}
cm\n/{_imgname} Do\nQ\n"
6812: (12)                      mupdf.fz_append_string(nres, s)
6813: (12)                      JM_insert_contents(pdf, page.obj(), nres, overlay)
6814: (8)                   if rc_digest:
6815: (12)                      return img_xref, digests
6816: (8)                   else:
6817: (12)                      return img_xref, None
6818: (4)               def _insertFont(self, fontname, bfname, fontfile, fontbuffer, set_simple,
idx, wmode, serif, encoding, ordering):
6819: (8)                   page = self._pdf_page()
6820: (8)                   pdf = page.doc()
6821: (8)                   value = JM_insert_font(pdf, bfname, fontfile,fontbuffer, set_simple,
idx, wmode, serif, encoding, ordering)
6822: (8)                   resources = mupdf.pdf_dict_get_inheritable( page.obj(),
PDF_NAME('Resources'))
6823: (8)                   fonts = mupdf.pdf_dict_get(resources, PDF_NAME('Font'))
6824: (8)                   if not fonts.m_internal:    # page has no fonts yet
6825: (12)                      fonts = mupdf.pdf_new_dict(pdf, 5)
6826: (12)                      mupdf.pdf_dict_putl(page.obj(), fonts, PDF_NAME('Resources'),
PDF_NAME('Font'))
6827: (8)                   _, xref = JM_INT_ITEM(value, 0)
6828: (8)                   if not xref:
6829: (12)                      raise RuntimeError( "cannot insert font")
6830: (8)                   font_obj = mupdf.pdf_new_indirect(pdf, xref, 0)
6831: (8)                   mupdf.pdf_dict_puts(fonts, fontname, font_obj)
6832: (8)                   return value
6833: (4)               def _load_annot(self, name, xref):
6834: (8)                   page = self._pdf_page()
6835: (8)                   if xref == 0:
6836: (12)                      annot = JM_get_annot_by_name(page, name)
6837: (8)                   else:
6838: (12)                      annot = JM_get_annot_by_xref(page, xref)
6839: (8)                   if annot.m_internal:
6840: (12)                      return Annot(annot)
6841: (4)               def _makePixmap(self, doc, ctm, cs, alpha=0, annots=1, clip=None):
6842: (8)                   pix = JM_pixmap_from_page(doc, self.this, ctm, cs, alpha, annots,
clip)
6843: (8)                   return Pixmap(pix)
6844: (4)               def _other_box(self, boxtype):
6845: (8)                   rect = mupdf.FzRect( mupdf.FzRect.Fixed_INFINITE)
6846: (8)                   page = _as_pdf_page(self.this, required=False)
6847: (8)                   if page.m_internal:
6848: (12)                      obj = mupdf.pdf_dict_gets( page.obj(), boxtype)
6849: (12)                      if mupdf.pdf_is_array(obj):
6850: (16)                          rect = mupdf.pdf_to_rect(obj)
6851: (8)                   if mupdf.fz_is_infinite_rect( rect):
6852: (12)                      return
6853: (8)                   return JM_py_from_rect(rect)
6854: (4)               def _pdf_page(self, required=True):
```

```
6855: (8)                   return _as_pdf_page(self.this, required=required)
6856: (4)               def _reset_annot_refs(self):
6857: (8)                   """Invalidate / delete all annots of this page."""
6858: (8)                   self._annot_refs.clear()
6859: (4)               def _set_opacity(self, gstate=None, CA=1, ca=1, blendmode=None):
6860: (8)                   if CA >= 1 and ca >= 1 and blendmode is None:
6861: (12)                      return
6862: (8)                   tCA = int(round(max(CA , 0) * 100))
6863: (8)                   if tCA >= 100:
6864: (12)                      tCA = 99
6865: (8)                   tca = int(round(max(ca, 0) * 100))
6866: (8)                   if tca >= 100:
6867: (12)                      tca = 99
6868: (8)                   gstate = "fitzca%02i%02i" % (tCA, tca)
6869: (8)                   if not gstate:
6870: (12)                      return
6871: (8)                   page = _as_pdf_page(self.this)
6872: (8)                   resources = mupdf.pdf_dict_get(page.obj(), PDF_NAME('Resources'))
6873: (8)                   if not resources.m_internal:
6874: (12)                      resources = mupdf.pdf_dict_put_dict(page.obj(),
PDF_NAME('Resources'), 2)
6875: (8)                   extg = mupdf.pdf_dict_get(resources, PDF_NAME('ExtGState'))
6876: (8)                   if not extg.m_internal:
6877: (12)                      extg = mupdf.pdf_dict_put_dict(resources, PDF_NAME('ExtGState'),
2)
6878: (8)                   n = mupdf.pdf_dict_len(extg)
6879: (8)                   for i in range(n):
6880: (12)                      o1 = mupdf.pdf_dict_get_key(extg, i)
6881: (12)                      name = mupdf.pdf_to_name(o1)
6882: (12)                      if name == gstate:
6883: (16)                          return gstate
6884: (8)                   opa = mupdf.pdf_new_dict(page.doc(), 3)
6885: (8)                   mupdf.pdf_dict_put_real(opa, PDF_NAME('CA'), CA)
6886: (8)                   mupdf.pdf_dict_put_real(opa, PDF_NAME('ca'), ca)
6887: (8)                   mupdf.pdf_dict_puts(extg, gstate, opa)
6888: (8)                   return gstate
6889: (4)               def _set_pagebox(self, boxtype, rect):
6890: (8)                   doc = self.parent
6891: (8)                   if doc is None:
6892: (12)                      raise ValueError("orphaned object: parent is None")
6893: (8)                   if not doc.is_pdf:
6894: (12)                      raise ValueError("is no PDF")
6895: (8)                   valid_boxes = ("CropBox", "BleedBox", "TrimBox", "ArtBox")
6896: (8)                   if boxtype not in valid_boxes:
6897: (12)                      raise ValueError("bad boxtype")
6898: (8)                   rect = Rect(rect)
6899: (8)                   mb = self.mediabox
6900: (8)                   rect = Rect(rect[0], mb.y1 - rect[3], rect[2], mb.y1 - rect[1])
6901: (8)                   if not (mb.x0 <= rect.x0 < rect.x1 <= mb.x1 and mb.y0 <= rect.y0 <
rect.y1 <= mb.y1):
6902: (12)                      raise ValueError(f"{boxtype} not in MediaBox")
6903: (8)                   doc.xref_set_key(self.xref, boxtype, f"[{_format_g(tuple(rect))}]")
6904: (4)               def _set_resource_property(self, name, xref):
6905: (8)                   page = self._pdf_page()
6906: (8)                   JM_set_resource_property(page.obj(), name, xref)
6907: (4)               def _show_pdf_page(self, fz_srcpage, overlay=1, matrix=None, xref=0, oc=0,
clip=None, graftmap=None, _imgname=None):
6908: (8)                   cropbox = JM_rect_from_py(clip)
6909: (8)                   mat = JM_matrix_from_py(matrix)
6910: (8)                   rc_xref = xref
6911: (8)                   tpage = _as_pdf_page(self.this)
6912: (8)                   tpageref = tpage.obj()
6913: (8)                   pdfout = tpage.doc()    # target PDF
6914: (8)                   ENSURE_OPERATION(pdfout)
6915: (8)                   xobj1 = JM_xobject_from_page(pdfout, fz_srcpage, xref, graftmap.this)
6916: (8)                   if not rc_xref:
6917: (12)                      rc_xref = mupdf.pdf_to_num(xobj1)
6918: (8)                   subres1 = mupdf.pdf_new_dict(pdfout, 5)
6919: (8)                   mupdf.pdf_dict_puts(subres1, "fullpage", xobj1)
```

```
6920: (8)                   subres = mupdf.pdf_new_dict(pdfout, 5)
6921: (8)                   mupdf.pdf_dict_put(subres, PDF_NAME('XObject'), subres1)
6922: (8)                   res = mupdf.fz_new_buffer(20)
6923: (8)                   mupdf.fz_append_string(res, "/fullpage Do")
6924: (8)                   xobj2 = mupdf.pdf_new_xobject(pdfout, cropbox, mat, subres, res)
6925: (8)                   if oc > 0:
6926: (12)                      JM_add_oc_object(pdfout, mupdf.pdf_resolve_indirect(xobj2), oc)
6927: (8)                   resources = mupdf.pdf_dict_get_inheritable(tpageref,
PDF_NAME('Resources'))
6928: (8)                   subres = mupdf.pdf_dict_get(resources, PDF_NAME('XObject'))
6929: (8)                   if not subres.m_internal:
6930: (12)                      subres = mupdf.pdf_dict_put_dict(resources, PDF_NAME('XObject'),
5)
6931: (8)                   mupdf.pdf_dict_puts(subres, _imgname, xobj2)
6932: (8)                   nres = mupdf.fz_new_buffer(50) # buffer for Do-command
6933: (8)                   mupdf.fz_append_string(nres, " q /")   # Do-command
6934: (8)                   mupdf.fz_append_string(nres, _imgname)
6935: (8)                   mupdf.fz_append_string(nres, " Do Q ")
6936: (8)                   JM_insert_contents(pdfout, tpageref, nres, overlay)
6937: (8)                   return rc_xref
6938: (4)               def add_caret_annot(self, point: point_like) -> Annot:
6939: (8)                   """Add a 'Caret' annotation."""
6940: (8)                   old_rotation = annot_preprocess(self)
6941: (8)                   try:
6942: (12)                      annot = self._add_caret_annot(point)
6943: (8)                   finally:
6944: (12)                      if old_rotation != 0:
6945: (16)                          self.set_rotation(old_rotation)
6946: (8)                   annot = Annot( annot)
6947: (8)                   annot_postprocess(self, annot)
6948: (8)                   assert hasattr( annot, 'parent')
6949: (8)                   return annot
6950: (4)               def add_circle_annot(self, rect: rect_like) -> Annot:
6951: (8)                   """Add a 'Circle' (ellipse, oval) annotation."""
6952: (8)                   old_rotation = annot_preprocess(self)
6953: (8)                   try:
6954: (12)                      annot = self._add_square_or_circle(rect, mupdf.PDF_ANNOT_CIRCLE)
6955: (8)                   finally:
6956: (12)                      if old_rotation != 0:
6957: (16)                          self.set_rotation(old_rotation)
6958: (8)                   annot_postprocess(self, annot)
6959: (8)                   return annot
6960: (4)               def add_file_annot(
6961: (12)                      self,
6962: (12)                      point: point_like,
6963: (12)                      buffer_: typing.ByteString,
6964: (12)                      filename: str,
6965: (12)                      ufilename: OptStr =None,
6966: (12)                      desc: OptStr =None,
6967: (12)                      icon: OptStr =None
6968: (12)                      ) -> Annot:
6969: (8)                   """Add a 'FileAttachment' annotation."""
6970: (8)                   old_rotation = annot_preprocess(self)
6971: (8)                   try:
6972: (12)                      annot = self._add_file_annot(point,
6973: (20)                              buffer_,
6974: (20)                              filename,
6975: (20)                              ufilename=ufilename,
6976: (20)                              desc=desc,
6977: (20)                              icon=icon,
6978: (20)                              )
6979: (8)                   finally:
6980: (12)                      if old_rotation != 0:
6981: (16)                          self.set_rotation(old_rotation)
6982: (8)                   annot_postprocess(self, annot)
6983: (8)                   return annot
6984: (4)               def add_freetext_annot(
6985: (12)                      self,
6986: (12)                      rect: rect_like,
```

```
6987: (12)                      text: str,
6988: (12)                      fontsize: float =11,
6989: (12)                      fontname: OptStr =None,
6990: (12)                      border_color: OptSeq =None,
6991: (12)                      text_color: OptSeq =None,
6992: (12)                      fill_color: OptSeq =None,
6993: (12)                      align: int =0,
6994: (12)                      rotate: int =0
6995: (12)                      ) -> Annot:
6996: (8)                   """Add a 'FreeText' annotation."""
6997: (8)                   old_rotation = annot_preprocess(self)
6998: (8)                   try:
6999: (12)                      annot = self._add_freetext_annot(
7000: (20)                              rect,
7001: (20)                              text,
7002: (20)                              fontsize=fontsize,
7003: (20)                              fontname=fontname,
7004: (20)                              border_color=border_color,
7005: (20)                              text_color=text_color,
7006: (20)                              fill_color=fill_color,
7007: (20)                              align=align,
7008: (20)                              rotate=rotate,
7009: (20)                              )
7010: (8)                   finally:
7011: (12)                      if old_rotation != 0:
7012: (16)                          self.set_rotation(old_rotation)
7013: (8)                   annot_postprocess(self, annot)
7014: (8)                   return annot
7015: (4)               def add_highlight_annot(self, quads=None, start=None,
7016: (26)                                      stop=None, clip=None) -> Annot:
7017: (8)                   """Add a 'Highlight' annotation."""
7018: (8)                   if quads is None:
7019: (12)                      q = get_highlight_selection(self, start=start, stop=stop,
clip=clip)
7020: (8)                   else:
7021: (12)                      q = CheckMarkerArg(quads)
7022: (8)                   ret = self._add_text_marker(q, mupdf.PDF_ANNOT_HIGHLIGHT)
7023: (8)                   return ret
7024: (4)               def add_ink_annot(self, handwriting: list) -> Annot:
7025: (8)                   """Add a 'Ink' ('handwriting') annotation.
7026: (8)                   The argument must be a list of lists of point_likes.
7027: (8)                   """
7028: (8)                   old_rotation = annot_preprocess(self)
7029: (8)                   try:
7030: (12)                      annot = self._add_ink_annot(handwriting)
7031: (8)                   finally:
7032: (12)                      if old_rotation != 0:
7033: (16)                          self.set_rotation(old_rotation)
7034: (8)                   annot_postprocess(self, annot)
7035: (8)                   return annot
7036: (4)               def add_line_annot(self, p1: point_like, p2: point_like) -> Annot:
7037: (8)                   """Add a 'Line' annotation."""
7038: (8)                   old_rotation = annot_preprocess(self)
7039: (8)                   try:
7040: (12)                      annot = self._add_line_annot(p1, p2)
7041: (8)                   finally:
7042: (12)                      if old_rotation != 0:
7043: (16)                          self.set_rotation(old_rotation)
7044: (8)                   annot_postprocess(self, annot)
7045: (8)                   return annot
7046: (4)               def add_polygon_annot(self, points: list) -> Annot:
7047: (8)                   """Add a 'Polygon' annotation."""
7048: (8)                   old_rotation = annot_preprocess(self)
7049: (8)                   try:
7050: (12)                      annot = self._add_multiline(points, mupdf.PDF_ANNOT_POLYGON)
7051: (8)                   finally:
7052: (12)                      if old_rotation != 0:
7053: (16)                          self.set_rotation(old_rotation)
7054: (8)                   annot_postprocess(self, annot)
```

```
(8)                 return annot
(4)             def add_polyline_annot(self, points: list) -> Annot:
(8)                 """Add a 'PolyLine' annotation."""
(8)                 old_rotation = annot_preprocess(self)
(8)                 try:
(12)                    annot = self._add_multiline(points, mupdf.PDF_ANNOT_POLY_LINE)
(8)                 finally:
(12)                    if old_rotation != 0:
(16)                        self.set_rotation(old_rotation)
(8)                 annot_postprocess(self, annot)
(8)                 return annot
(4)             def add_rect_annot(self, rect: rect_like) -> Annot:
(8)                 """Add a 'Square' (rectangle) annotation."""
(8)                 old_rotation = annot_preprocess(self)
(8)                 try:
(12)                    annot = self._add_square_or_circle(rect, mupdf.PDF_ANNOT_SQUARE)
(8)                 finally:
(12)                    if old_rotation != 0:
(16)                        self.set_rotation(old_rotation)
(8)                 annot_postprocess(self, annot)
(8)                 return annot
(4)             def add_redact_annot(
(12)                    self,
(12)                    quad,
(12)                    text: OptStr =None,
(12)                    fontname: OptStr =None,
(12)                    fontsize: float =11,
(12)                    align: int =0,
(12)                    fill: OptSeq =None,
(12)                    text_color: OptSeq =None,
(12)                    cross_out: bool =True,
(12)                    ) -> Annot:
(8)                 """Add a 'Redact' annotation."""
(8)                 da_str = None
(8)                 if text and not set(string.whitespace).issuperset(text):
(12)                    CheckColor(fill)
(12)                    CheckColor(text_color)
(12)                    if not fontname:
(16)                        fontname = "Helv"
(12)                    if not fontsize:
(16)                        fontsize = 11
(12)                    if not text_color:
(16)                        text_color = (0, 0, 0)
(12)                    if hasattr(text_color, "__float__"):
(16)                        text_color = (text_color, text_color, text_color)
(12)                    if len(text_color) > 3:
(16)                        text_color = text_color[:3]
(12)                    fmt = "{:g} {:g} {:g} rg /{f:s} {s:g} Tf"
(12)                    da_str = fmt.format(*text_color, f=fontname, s=fontsize)
(12)                    if fill is None:
(16)                        fill = (1, 1, 1)
(12)                    if fill:
(16)                        if hasattr(fill, "__float__"):
(20)                            fill = (fill, fill, fill)
(16)                        if len(fill) > 3:
(20)                            fill = fill[:3]
(8)                 old_rotation = annot_preprocess(self)
(8)                 try:
(12)                    annot = self._add_redact_annot(quad, text=text, da_str=da_str,
(23)                                   align=align, fill=fill)
(8)                 finally:
(12)                    if old_rotation != 0:
(16)                        self.set_rotation(old_rotation)
(8)                 annot_postprocess(self, annot)
(8)                 if cross_out:
(12)                    ap_tab = annot._getAP().splitlines()[:-1]  # get the 4 commands
only
(12)                    _, LL, LR, UR, UL = ap_tab
(12)                    ap_tab.append(LR)
```

```
7123: (12)                      ap_tab.append(LL)
7124: (12)                      ap_tab.append(UR)
7125: (12)                      ap_tab.append(LL)
7126: (12)                      ap_tab.append(UL)
7127: (12)                      ap_tab.append(b"S")
7128: (12)                      ap = b"\n".join(ap_tab)
7129: (12)                      annot._setAP(ap, 0)
7130: (8)                   return annot
7131: (4)               def add_squiggly_annot(
7132: (12)                      self,
7133: (12)                      quads=None,
7134: (12)                      start=None,
7135: (12)                      stop=None,
7136: (12)                      clip=None,
7137: (12)                      ) -> Annot:
7138: (8)                   """Add a 'Squiggly' annotation."""
7139: (8)                   if quads is None:
7140: (12)                      q = get_highlight_selection(self, start=start, stop=stop,
clip=clip)
7141: (8)                   else:
7142: (12)                      q = CheckMarkerArg(quads)
7143: (8)                   return self._add_text_marker(q, mupdf.PDF_ANNOT_SQUIGGLY)
7144: (4)               def add_stamp_annot(self, rect: rect_like, stamp: int =0) -> Annot:
7145: (8)                   """Add a ('rubber') 'Stamp' annotation."""
7146: (8)                   old_rotation = annot_preprocess(self)
7147: (8)                   try:
7148: (12)                      annot = self._add_stamp_annot(rect, stamp)
7149: (8)                   finally:
7150: (12)                      if old_rotation != 0:
7151: (16)                          self.set_rotation(old_rotation)
7152: (8)                   annot_postprocess(self, annot)
7153: (8)                   return annot
7154: (4)               def add_strikeout_annot(self, quads=None, start=None, stop=None,
clip=None) -> Annot:
7155: (8)                   """Add a 'StrikeOut' annotation."""
7156: (8)                   if quads is None:
7157: (12)                      q = get_highlight_selection(self, start=start, stop=stop,
clip=clip)
7158: (8)                   else:
7159: (12)                      q = CheckMarkerArg(quads)
7160: (8)                   return self._add_text_marker(q, mupdf.PDF_ANNOT_STRIKE_OUT)
7161: (4)               def add_text_annot(self, point: point_like, text: str, icon: str ="Note")
-> Annot:
7162: (8)                   """Add a 'Text' (sticky note) annotation."""
7163: (8)                   old_rotation = annot_preprocess(self)
7164: (8)                   try:
7165: (12)                      annot = self._add_text_annot(point, text, icon=icon)
7166: (8)                   finally:
7167: (12)                      if old_rotation != 0:
7168: (16)                          self.set_rotation(old_rotation)
7169: (8)                   annot_postprocess(self, annot)
7170: (8)                   return annot
7171: (4)               def add_underline_annot(self, quads=None, start=None, stop=None,
clip=None) -> Annot:
7172: (8)                   """Add a 'Underline' annotation."""
7173: (8)                   if quads is None:
7174: (12)                      q = get_highlight_selection(self, start=start, stop=stop,
clip=clip)
7175: (8)                   else:
7176: (12)                      q = CheckMarkerArg(quads)
7177: (8)                   return self._add_text_marker(q, mupdf.PDF_ANNOT_UNDERLINE)
7178: (4)               def add_widget(self, widget: Widget) -> Annot:
7179: (8)                   """Add a 'Widget' (form field)."""
7180: (8)                   CheckParent(self)
7181: (8)                   doc = self.parent
7182: (8)                   if not doc.is_pdf:
7183: (12)                      raise ValueError("is no PDF")
7184: (8)                   widget._validate()
7185: (8)                   annot = self._addWidget(widget.field_type, widget.field_name)
```

```
7186: (8)                    if not annot:
7187: (12)                       return None
7188: (8)                    annot.thisown = True
7189: (8)                    annot.parent = weakref.proxy(self) # owning page object
7190: (8)                    self._annot_refs[id(annot)] = annot
7191: (8)                    widget.parent = annot.parent
7192: (8)                    widget._annot = annot
7193: (8)                    widget.update()
7194: (8)                    return annot
7195: (4)                def annot_names(self):
7196: (8)                    '''
7197: (8)                    page get list of annot names
7198: (8)                    '''
7199: (8)                    """List of names of annotations, fields and links."""
7200: (8)                    CheckParent(self)
7201: (8)                    page = self._pdf_page(required=False)
7202: (8)                    if not page.m_internal:
7203: (12)                       return []
7204: (8)                    return JM_get_annot_id_list(page)
7205: (4)                def annot_xrefs(self):
7206: (8)                    '''
7207: (8)                    List of xref numbers of annotations, fields and links.
7208: (8)                    '''
7209: (8)                    return JM_get_annot_xref_list2(self)
7210: (4)                def annots(self, types=None):
7211: (8)                    """ Generator over the annotations of a page.
7212: (8)                    Args:
7213: (12)                       types: (list) annotation types to subselect from. If none,
7214: (19)                              all annotations are returned. E.g. types=[PDF_ANNOT_LINE]
7215: (19)                              will only yield line annotations.
7216: (8)                    """
7217: (8)                    skip_types = (mupdf.PDF_ANNOT_LINK, mupdf.PDF_ANNOT_POPUP,
mupdf.PDF_ANNOT_WIDGET)
7218: (8)                    if not hasattr(types, "__getitem__"):
7219: (12)                       annot_xrefs = [a[0] for a in self.annot_xrefs() if a[1] not in
skip_types]
7220: (8)                    else:
7221: (12)                       annot_xrefs = [a[0] for a in self.annot_xrefs() if a[1] in types
and a[1] not in skip_types]
7222: (8)                    for xref in annot_xrefs:
7223: (12)                       annot = self.load_annot(xref)
7224: (12)                       annot._yielded=True
7225: (12)                       yield annot
7226: (4)                @property
7227: (4)                def artbox(self):
7228: (8)                    """The ArtBox"""
7229: (8)                    rect = self._other_box("ArtBox")
7230: (8)                    if rect is None:
7231: (12)                       return self.cropbox
7232: (8)                    mb = self.mediabox
7233: (8)                    return Rect(rect[0], mb.y1 - rect[3], rect[2], mb.y1 - rect[1])
7234: (4)                @property
7235: (4)                def bleedbox(self):
7236: (8)                    """The BleedBox"""
7237: (8)                    rect = self._other_box("BleedBox")
7238: (8)                    if rect is None:
7239: (12)                       return self.cropbox
7240: (8)                    mb = self.mediabox
7241: (8)                    return Rect(rect[0], mb.y1 - rect[3], rect[2], mb.y1 - rect[1])
7242: (4)                def bound(self):
7243: (8)                    """Get page rectangle."""
7244: (8)                    CheckParent(self)
7245: (8)                    page = _as_fz_page(self.this)
7246: (8)                    val = mupdf.fz_bound_page(page)
7247: (8)                    val = Rect(val)
7248: (8)                    if val.is_infinite and self.parent.is_pdf:
7249: (12)                       cb = self.cropbox
7250: (12)                       w, h = cb.width, cb.height
7251: (12)                       if self.rotation not in (0, 180):
```

```
(16)                        w, h = h, w
(12)                    val = Rect(0, 0, w, h)
(12)                    msg = TOOLS.mupdf_warnings(reset=False).splitlines()[-1]
(12)                    message(msg)
(8)                 return val
(4)             def clean_contents(self, sanitize=1):
(8)                 if not sanitize and not self.is_wrapped:
(12)                    self.wrap_contents()
(8)                 page = _as_pdf_page( self.this, required=False)
(8)                 if not page.m_internal:
(12)                    return
(8)                 filter_ = _make_PdfFilterOptions(recurse=1, sanitize=sanitize)
(8)                 mupdf.pdf_filter_page_contents( page.doc(), page, filter_)
(4)             @property
(4)             def cropbox(self):
(8)                 """The CropBox."""
(8)                 CheckParent(self)
(8)                 page = self._pdf_page(required=False)
(8)                 if not page.m_internal:
(12)                    val = mupdf.fz_bound_page(self.this)
(8)                 else:
(12)                    val = JM_cropbox(page.obj())
(8)                 val = Rect(val)
(8)                 return val
(4)             @property
(4)             def cropbox_position(self):
(8)                 return self.cropbox.tl
(4)             def delete_annot(self, annot):
(8)                 """Delete annot and return next one."""
(8)                 CheckParent(self)
(8)                 CheckParent(annot)
(8)                 page = self._pdf_page()
(8)                 while 1:
(12)                    irt_annot = JM_find_annot_irt(annot.this)
(12)                    if not irt_annot:    # no more there
(16)                        break
(12)                    mupdf.pdf_delete_annot(page, irt_annot.this)
(8)                 nextannot = mupdf.pdf_next_annot(annot.this)   # store next
(8)                 mupdf.pdf_delete_annot(page, annot.this)
(8)                 val = Annot(nextannot)
(8)                 if val:
(12)                    val.thisown = True
(12)                    val.parent = weakref.proxy(self) # owning page object
(12)                    val.parent._annot_refs[id(val)] = val
(8)                 annot._erase()
(8)                 return val
(4)             def delete_link(self, linkdict):
(8)                 """Delete a Link."""
(8)                 CheckParent(self)
(8)                 if not isinstance( linkdict, dict):
(12)                    return  # have no dictionary
(8)                 def finished():
(12)                    if linkdict["xref"] == 0: return
(12)                    try:
(16)                        linkid = linkdict["id"]
(16)                        linkobj = self._annot_refs[linkid]
(16)                        linkobj._erase()
(12)                    except Exception:
(16)                        if g_exceptions_verbose > 1:    exception_info()
(16)                        pass
(8)                 page = _as_pdf_page(self.this, required=False)
(8)                 if not page.m_internal:
(12)                    return finished()   # have no PDF
(8)                 xref = linkdict[dictkey_xref]
(8)                 if xref < 1:
(12)                    return finished()   # invalid xref
(8)                 annots = mupdf.pdf_dict_get( page.obj(), PDF_NAME('Annots'))
(8)                 if not annots.m_internal:
(12)                    return finished()   # have no annotations
```

```
7321: (8)                   len_ = mupdf.pdf_array_len( annots)
7322: (8)                   if len_ == 0:
7323: (12)                      return finished()
7324: (8)                   oxref = 0
7325: (8)                   for i in range( len_):
7326: (12)                      oxref = mupdf.pdf_to_num( mupdf.pdf_array_get( annots, i))
7327: (12)                      if xref == oxref:
7328: (16)                          break   # found xref in annotations
7329: (8)                   if xref != oxref:
7330: (12)                      return finished()   # xref not in annotations
7331: (8)                   mupdf.pdf_array_delete( annots, i) # delete entry in annotations
7332: (8)                   mupdf.pdf_delete_object( page.doc(), xref) # delete link object
7333: (8)                   mupdf.pdf_dict_put( page.obj(), PDF_NAME('Annots'), annots)
7334: (8)                   JM_refresh_links( page)
7335: (8)                   return finished()
7336: (4)               @property
7337: (4)               def derotation_matrix(self) -> Matrix:
7338: (8)                   """Reflects page de-rotation."""
7339: (8)                   if g_use_extra:
7340: (12)                      return Matrix(extra.Page_derotate_matrix( self.this))
7341: (8)                   pdfpage = self._pdf_page(required=False)
7342: (8)                   if not pdfpage.m_internal:
7343: (12)                      return Matrix(mupdf.FzRect(mupdf.FzRect.UNIT))
7344: (8)                   return Matrix(JM_derotate_page_matrix(pdfpage))
7345: (4)               def extend_textpage(self, tpage, flags=0, matrix=None):
7346: (8)                   page = self.this
7347: (8)                   tp = tpage.this
7348: (8)                   assert isinstance( tp, mupdf.FzStextPage)
7349: (8)                   options = mupdf.FzStextOptions()
7350: (8)                   options.flags = flags
7351: (8)                   ctm = JM_matrix_from_py(matrix)
7352: (8)                   dev = mupdf.FzDevice(tp, options)
7353: (8)                   mupdf.fz_run_page( page, dev, ctm, mupdf.FzCookie())
7354: (8)                   mupdf.fz_close_device( dev)
7355: (4)               @property
7356: (4)               def first_annot(self):
7357: (8)                   """First annotation."""
7358: (8)                   CheckParent(self)
7359: (8)                   page = self._pdf_page(required=False)
7360: (8)                   if not page.m_internal:
7361: (12)                      return
7362: (8)                   annot = mupdf.pdf_first_annot(page)
7363: (8)                   if not annot.m_internal:
7364: (12)                      return
7365: (8)                   val = Annot(annot)
7366: (8)                   val.thisown = True
7367: (8)                   val.parent = weakref.proxy(self) # owning page object
7368: (8)                   self._annot_refs[id(val)] = val
7369: (8)                   return val
7370: (4)               @property
7371: (4)               def first_link(self):
7372: (8)                   '''
7373: (8)                   First link on page
7374: (8)                   '''
7375: (8)                   return self.load_links()
7376: (4)               @property
7377: (4)               def first_widget(self):
7378: (8)                   """First widget/field."""
7379: (8)                   CheckParent(self)
7380: (8)                   annot = 0
7381: (8)                   page = self._pdf_page(required=False)
7382: (8)                   if not page.m_internal:
7383: (12)                      return
7384: (8)                   annot = mupdf.pdf_first_widget(page)
7385: (8)                   if not annot.m_internal:
7386: (12)                      return
7387: (8)                   val = Annot(annot)
7388: (8)                   val.thisown = True
7389: (8)                   val.parent = weakref.proxy(self) # owning page object
```

```
7390: (8)                    self._annot_refs[id(val)] = val
7391: (8)                    widget = Widget()
7392: (8)                    TOOLS._fill_widget(val, widget)
7393: (8)                    val = widget
7394: (8)                    return val
7395: (4)                def get_bboxlog(self, layers=None):
7396: (8)                    CheckParent(self)
7397: (8)                    old_rotation = self.rotation
7398: (8)                    if old_rotation != 0:
7399: (12)                       self.set_rotation(0)
7400: (8)                    page = self.this
7401: (8)                    rc = []
7402: (8)                    inc_layers = True if layers else False
7403: (8)                    dev = JM_new_bbox_device( rc, inc_layers)
7404: (8)                    mupdf.fz_run_page( page, dev, mupdf.FzMatrix(), mupdf.FzCookie())
7405: (8)                    mupdf.fz_close_device( dev)
7406: (8)                    if old_rotation != 0:
7407: (12)                       self.set_rotation(old_rotation)
7408: (8)                    return rc
7409: (4)                def get_cdrawings(self, extended=None, callback=None, method=None):
7410: (8)                    """Extract vector graphics ("line art") from the page."""
7411: (8)                    CheckParent(self)
7412: (8)                    old_rotation = self.rotation
7413: (8)                    if old_rotation != 0:
7414: (12)                       self.set_rotation(0)
7415: (8)                    page = self.this
7416: (8)                    if isinstance(page, mupdf.PdfPage):
7417: (12)                       page = mupdf.FzPage(page)
7418: (8)                    assert isinstance(page, mupdf.FzPage), f'{self.this=}'
7419: (8)                    clips = True if extended else False
7420: (8)                    prect = mupdf.fz_bound_page(page)
7421: (8)                    if g_use_extra:
7422: (12)                       rc = extra.get_cdrawings(page, extended, callback, method)
7423: (8)                    else:
7424: (12)                       rc = list()
7425: (12)                       if callable(callback) or method is not None:
7426: (16)                           dev = JM_new_lineart_device_Device(callback, clips, method)
7427: (12)                       else:
7428: (16)                           dev = JM_new_lineart_device_Device(rc, clips, method)
7429: (12)                       dev.ptm = mupdf.FzMatrix(1, 0, 0, -1, 0, prect.y1)
7430: (12)                       mupdf.fz_run_page(page, dev, mupdf.FzMatrix(), mupdf.FzCookie())
7431: (12)                       mupdf.fz_close_device(dev)
7432: (8)                    if old_rotation != 0:
7433: (12)                       self.set_rotation(old_rotation)
7434: (8)                    if callable(callback) or method is not None:
7435: (12)                       return
7436: (8)                    return rc
7437: (4)                def get_contents(self):
7438: (8)                    """Get xrefs of /Contents objects."""
7439: (8)                    CheckParent(self)
7440: (8)                    ret = []
7441: (8)                    page = _as_pdf_page(self.this)
7442: (8)                    obj = page.obj()
7443: (8)                    contents = mupdf.pdf_dict_get(obj, mupdf.PDF_ENUM_NAME_Contents)
7444: (8)                    if mupdf.pdf_is_array(contents):
7445: (12)                       n = mupdf.pdf_array_len(contents)
7446: (12)                       for i in range(n):
7447: (16)                           icont = mupdf.pdf_array_get(contents, i)
7448: (16)                           xref = mupdf.pdf_to_num(icont)
7449: (16)                           ret.append(xref)
7450: (8)                    elif contents.m_internal:
7451: (12)                       xref = mupdf.pdf_to_num(contents)
7452: (12)                       ret.append( xref)
7453: (8)                    return ret
7454: (4)                def get_displaylist(self, annots=1):
7455: (8)                    '''
7456: (8)                    Make a DisplayList from the page for Pixmap generation.
7457: (8)                    Include (default) or exclude annotations.
7458: (8)                    '''
```

```
7459: (8)                   CheckParent(self)
7460: (8)                   if annots:
7461: (12)                      dl = mupdf.fz_new_display_list_from_page(self.this)
7462: (8)                   else:
7463: (12)                      dl = mupdf.fz_new_display_list_from_page_contents(self.this)
7464: (8)                   return DisplayList(dl)
7465: (4)               def get_drawings(self, extended: bool=False) -> list:
7466: (8)                   """Retrieve vector graphics. The extended version includes clips.
7467: (8)                   Note:
7468: (8)                   For greater comfort, this method converts point-likes, rect-likes,
quad-likes
7469: (8)                   of the C version to respective Point / Rect / Quad objects.
7470: (8)                   It also adds default items that are missing in original path types.
7471: (8)                   """
7472: (8)                   allkeys = (
7473: (16)                          'closePath',
7474: (16)                          'fill',
7475: (16)                          'color',
7476: (16)                          'width',
7477: (16)                          'lineCap',
7478: (16)                          'lineJoin',
7479: (16)                          'dashes',
7480: (16)                          'stroke_opacity',
7481: (16)                          'fill_opacity',
7482: (16)                          'even_odd',
7483: (16)                          )
7484: (8)                   val = self.get_cdrawings(extended=extended)
7485: (8)                   for i in range(len(val)):
7486: (12)                      npath = val[i]
7487: (12)                      if not npath["type"].startswith("clip"):
7488: (16)                          npath["rect"] = Rect(npath["rect"])
7489: (12)                      else:
7490: (16)                          npath["scissor"] = Rect(npath["scissor"])
7491: (12)                      if npath["type"]!="group":
7492: (16)                          items = npath["items"]
7493: (16)                          newitems = []
7494: (16)                          for item in items:
7495: (20)                              cmd = item[0]
7496: (20)                              rest = item[1:]
7497: (20)                              if  cmd == "re":
7498: (24)                                  item = ("re", Rect(rest[0]).normalize(), rest[1])
7499: (20)                              elif cmd == "qu":
7500: (24)                                  item = ("qu", Quad(rest[0]))
7501: (20)                              else:
7502: (24)                                  item = tuple([cmd] + [Point(i) for i in rest])
7503: (20)                              newitems.append(item)
7504: (16)                          npath["items"] = newitems
7505: (12)                      if npath['type'] in ('f', 's'):
7506: (16)                          for k in allkeys:
7507: (20)                              npath[k] = npath.get(k)
7508: (12)                      val[i] = npath
7509: (8)                   return val
7510: (8)                   class Drawpath(object):
7511: (12)                      """Reflects a path dictionary from get_cdrawings()."""
7512: (12)                      def __init__(self, **args):
7513: (16)                          self.__dict__.update(args)
7514: (8)                   class Drawpathlist(object):
7515: (12)                      """List of Path objects representing get_cdrawings() output."""
7516: (12)                      def __getitem__(self, item):
7517: (16)                          return self.paths.__getitem__(item)
7518: (12)                      def __init__(self):
7519: (16)                          self.paths = []
7520: (16)                          self.path_count = 0
7521: (16)                          self.group_count = 0
7522: (16)                          self.clip_count = 0
7523: (16)                          self.fill_count = 0
7524: (16)                          self.stroke_count = 0
7525: (16)                          self.fillstroke_count = 0
7526: (12)                      def __len__(self):
```

```
7527: (16)                          return self.paths.__len__()
7528: (12)                      def append(self, path):
7529: (16)                          self.paths.append(path)
7530: (16)                          self.path_count += 1
7531: (16)                          if path.type == "clip":
7532: (20)                              self.clip_count += 1
7533: (16)                          elif path.type == "group":
7534: (20)                              self.group_count += 1
7535: (16)                          elif path.type == "f":
7536: (20)                              self.fill_count += 1
7537: (16)                          elif path.type == "s":
7538: (20)                              self.stroke_count += 1
7539: (16)                          elif path.type == "fs":
7540: (20)                              self.fillstroke_count += 1
7541: (12)                      def clip_parents(self, i):
7542: (16)                          """Return list of parent clip paths.
7543: (16)                          Args:
7544: (20)                              i: (int) return parents of this path.
7545: (16)                          Returns:
7546: (20)                              List of the clip parents."""
7547: (16)                          if i >= self.path_count:
7548: (20)                              raise IndexError("bad path index")
7549: (16)                          while i < 0:
7550: (20)                              i += self.path_count
7551: (16)                          lvl = self.paths[i].level
7552: (16)                          clips = list(  # clip paths before identified one
7553: (20)                              reversed(
7554: (24)                                  [
7555: (28)                                      p
7556: (28)                                      for p in self.paths[:i]
7557: (28)                                      if p.type == "clip" and p.level < lvl
7558: (24)                                  ]
7559: (20)                              )
7560: (16)                          )
7561: (16)                          if clips == []:  # none found: empty list
7562: (20)                              return []
7563: (16)                          nclips = [clips[0]]  # init return list
7564: (16)                          for p in clips[1:]:
7565: (20)                              if p.level >= nclips[-1].level:
7566: (24)                                  continue  # only accept smaller clip levels
7567: (20)                              nclips.append(p)
7568: (16)                          return nclips
7569: (12)                      def group_parents(self, i):
7570: (16)                          """Return list of parent group paths.
7571: (16)                          Args:
7572: (20)                              i: (int) return parents of this path.
7573: (16)                          Returns:
7574: (20)                              List of the group parents."""
7575: (16)                          if i >= self.path_count:
7576: (20)                              raise IndexError("bad path index")
7577: (16)                          while i < 0:
7578: (20)                              i += self.path_count
7579: (16)                          lvl = self.paths[i].level
7580: (16)                          groups = list(  # group paths before identified one
7581: (20)                              reversed(
7582: (24)                                  [
7583: (28)                                      p
7584: (28)                                      for p in self.paths[:i]
7585: (28)                                      if p.type == "group" and p.level < lvl
7586: (24)                                  ]
7587: (20)                              )
7588: (16)                          )
7589: (16)                          if groups == []:  # none found: empty list
7590: (20)                              return []
7591: (16)                          ngroups = [groups[0]]  # init return list
7592: (16)                          for p in groups[1:]:
7593: (20)                              if p.level >= ngroups[-1].level:
7594: (24)                                  continue  # only accept smaller group levels
7595: (20)                              ngroups.append(p)
```

```
7596: (16)                          return ngroups
7597: (8)                   def get_lineart(self) -> object:
7598: (12)                      """Get page drawings paths.
7599: (12)                      Note:
7600: (12)                      For greater comfort, this method converts point-like, rect-like,
quad-like
7601: (12)                      tuples of the C version to respective Point / Rect / Quad objects.
7602: (12)                      Also adds default items that are missing in original path types.
7603: (12)                      In contrast to get_drawings(), this output is an object.
7604: (12)                      """
7605: (12)                      val = self.get_cdrawings(extended=True)
7606: (12)                      paths = self.Drawpathlist()
7607: (12)                      for path in val:
7608: (16)                          npath = self.Drawpath(**path)
7609: (16)                          if npath.type != "clip":
7610: (20)                              npath.rect = Rect(path["rect"])
7611: (16)                          else:
7612: (20)                              npath.scissor = Rect(path["scissor"])
7613: (16)                          if npath.type != "group":
7614: (20)                              items = path["items"]
7615: (20)                              newitems = []
7616: (20)                              for item in items:
7617: (24)                                  cmd = item[0]
7618: (24)                                  rest = item[1:]
7619: (24)                                  if  cmd == "re":
7620: (28)                                      item = ("re", Rect(rest[0]).normalize(), rest[1])
7621: (24)                                  elif cmd == "qu":
7622: (28)                                      item = ("qu", Quad(rest[0]))
7623: (24)                                  else:
7624: (28)                                      item = tuple([cmd] + [Point(i) for i in rest])
7625: (24)                                  newitems.append(item)
7626: (20)                              npath.items = newitems
7627: (16)                          if npath.type == "f":
7628: (20)                              npath.stroke_opacity = None
7629: (20)                              npath.dashes = None
7630: (20)                              npath.line_join = None
7631: (20)                              npath.line_cap = None
7632: (20)                              npath.color = None
7633: (20)                              npath.width = None
7634: (16)                          paths.append(npath)
7635: (12)                      val = None
7636: (12)                      return paths
7637: (4)               def remove_rotation(self):
7638: (8)                   """Set page rotation to 0 while maintaining visual appearance."""
7639: (8)                   rot = self.rotation  # normalized rotation value
7640: (8)                   if rot == 0:
7641: (12)                      return  Identity # nothing to do
7642: (8)                   mb = self.mediabox  # current mediabox
7643: (8)                   if rot == 90:
7644: (12)                      mat0 = Matrix(1, 0, 0, 1, mb.y1 - mb.x1 - mb.x0 - mb.y0, 0)
7645: (8)                   elif rot == 270:
7646: (12)                      mat0 = Matrix(1, 0, 0, 1, 0, mb.x1 - mb.y1 - mb.y0 - mb.x0)
7647: (8)                   else:  # rot = 180
7648: (12)                      mat0 = Matrix(1, 0, 0, 1, -2 * mb.x0, -2 * mb.y0)
7649: (8)                   mat = mat0 * self.derotation_matrix
7650: (8)                   cmd = _format_g(tuple(mat)) + ' cm '
7651: (8)                   cmd = cmd.encode('utf8')
7652: (8)                   _ = TOOLS._insert_contents(self, cmd, False)  # prepend to page
contents
7653: (8)                   if rot in (90, 270):
7654: (12)                      x0, y0, x1, y1 = mb
7655: (12)                      mb.x0 = y0
7656: (12)                      mb.y0 = x0
7657: (12)                      mb.x1 = y1
7658: (12)                      mb.y1 = x1
7659: (12)                      self.set_mediabox(mb)
7660: (8)                   self.set_rotation(0)
7661: (8)                   rot = ~mat  # inverse of the derotation matrix
7662: (8)                   for annot in self.annots():  # modify rectangles of annotations
```

```
7663: (12)                        r = annot.rect * rot
7664: (12)                        annot.set_rect(r)
7665: (8)                    for link in self.get_links():  # modify 'from' rectangles of links
7666: (12)                        r = link["from"] * rot
7667: (12)                        self.delete_link(link)
7668: (12)                        link["from"] = r
7669: (12)                        try:  # invalid links remain deleted
7670: (16)                            self.insert_link(link)
7671: (12)                        except Exception:
7672: (16)                            pass
7673: (8)                    for widget in self.widgets():  # modify field rectangles
7674: (12)                        r = widget.rect * rot
7675: (12)                        widget.rect = r
7676: (12)                        widget.update()
7677: (8)                    return rot  # the inverse of the generated derotation matrix
7678: (4)                def cluster_drawings(
7679: (8)                    self, clip=None, drawings=None, x_tolerance: float = 3, y_tolerance:
float = 3
7680: (4)                ) -> list:
7681: (8)                    """Join rectangles of neighboring vector graphic items.
7682: (8)                    Args:
7683: (12)                        clip: optional rect-like to restrict the page area to consider.
7684: (12)                        drawings: (optional) output of a previous "get_drawings()".
7685: (12)                        x_tolerance: horizontal neighborhood threshold.
7686: (12)                        y_tolerance: vertical neighborhood threshold.
7687: (8)                    Notes:
7688: (12)                        Vector graphics (also called line-art or drawings) usually consist
7689: (12)                        of independent items like rectangles, lines or curves to jointly
7690: (12)                        form table grid lines or bar, line, pie charts and similar.
7691: (12)                        This method identifies rectangles wrapping these disparate items.
7692: (8)                    Returns:
7693: (12)                        A list of Rect items, each wrapping line-art items that are close
7694: (12)                        enough to be considered forming a common vector graphic.
7695: (12)                        Only "significant" rectangles will be returned, i.e. having both,
7696: (12)                        width and height larger than the tolerance values.
7697: (8)                    """
7698: (8)                    CheckParent(self)
7699: (8)                    parea = self.rect  # the default clipping area
7700: (8)                    if clip is not None:
7701: (12)                        parea = Rect(clip)
7702: (8)                    delta_x = x_tolerance  # shorter local name
7703: (8)                    delta_y = y_tolerance  # shorter local name
7704: (8)                    if drawings is None:  # if we cannot re-use a previous output
7705: (12)                        drawings = self.get_drawings()
7706: (8)                    def are_neighbors(r1, r2):
7707: (12)                        """Detect whether r1, r2 are "neighbors".
7708: (12)                        Items r1, r2 are called neighbors if the minimum distance between
7709: (12)                        their points is less-equal delta.
7710: (12)                        Both parameters must be (potentially invalid) rectangles.
7711: (12)                        """
7712: (12)                        rr1_x0, rr1_x1 = (r1.x0, r1.x1) if r1.x1 > r1.x0 else (r1.x1,
r1.x0)
7713: (12)                        rr1_y0, rr1_y1 = (r1.y0, r1.y1) if r1.y1 > r1.y0 else (r1.y1,
r1.y0)
7714: (12)                        rr2_x0, rr2_x1 = (r2.x0, r2.x1) if r2.x1 > r2.x0 else (r2.x1,
r2.x0)
7715: (12)                        rr2_y0, rr2_y1 = (r2.y0, r2.y1) if r2.y1 > r2.y0 else (r2.y1,
r2.y0)
7716: (12)                        if (
7717: (16)                            0
7718: (16)                            or rr1_x1 < rr2_x0 - delta_x
7719: (16)                            or rr1_x0 > rr2_x1 + delta_x
7720: (16)                            or rr1_y1 < rr2_y0 - delta_y
7721: (16)                            or rr1_y0 > rr2_y1 + delta_y
7722: (12)                        ):
7723: (16)                            return False
7724: (12)                        else:
7725: (16)                            return True
7726: (8)                    paths = [
```

```
7727: (12)                        p
7728: (12)                        for p in drawings
7729: (12)                        if 1
7730: (12)                        and p["rect"].x0 >= parea.x0
7731: (12)                        and p["rect"].x1 <= parea.x1
7732: (12)                        and p["rect"].y0 >= parea.y0
7733: (12)                        and p["rect"].y1 <= parea.y1
7734: (8)                     ]
7735: (8)                     prects = sorted([p["rect"] for p in paths], key=lambda r: (r.y1,
r.x0))
7736: (8)                     new_rects = []  # the final list of the joined rectangles
7737: (8)                     while prects:  # the algorithm will empty this list
7738: (12)                        r = +prects[0]  # copy of first rectangle
7739: (12)                        repeat = True
7740: (12)                        while repeat:
7741: (16)                            repeat = False
7742: (16)                            for i in range(len(prects) - 1, 0, -1):  # from back to front
7743: (20)                                if are_neighbors(prects[i], r):
7744: (24)                                    r |= prects[i].tl  # include in first rect
7745: (24)                                    r |= prects[i].br  # include in first rect
7746: (24)                                    del prects[i]  # delete this rect
7747: (24)                                    repeat = True
7748: (12)                        new_rects.append(r)
7749: (12)                        del prects[0]
7750: (12)                        prects = sorted(set(prects), key=lambda r: (r.y1, r.x0))
7751: (8)                     new_rects = sorted(set(new_rects), key=lambda r: (r.y1, r.x0))
7752: (8)                     return [r for r in new_rects if r.width > delta_x and r.height >
delta_y]
7753: (4)                 def get_fonts(self, full=False):
7754: (8)                     """List of fonts defined in the page object."""
7755: (8)                     CheckParent(self)
7756: (8)                     return self.parent.get_page_fonts(self.number, full=full)
7757: (4)                 def get_image_bbox(self, name, transform=0):
7758: (8)                     """Get rectangle occupied by image 'name'.
7759: (8)                     'name' is either an item of the image list, or the referencing
7760: (8)                     name string - elem[7] of the resp. item.
7761: (8)                     Option 'transform' also returns the image transformation matrix.
7762: (8)                     """
7763: (8)                     CheckParent(self)
7764: (8)                     doc = self.parent
7765: (8)                     if doc.is_closed or doc.is_encrypted:
7766: (12)                        raise ValueError('document closed or encrypted')
7767: (8)                     inf_rect = Rect(1, 1, -1, -1)
7768: (8)                     null_mat = Matrix()
7769: (8)                     if transform:
7770: (12)                        rc = (inf_rect, null_mat)
7771: (8)                     else:
7772: (12)                        rc = inf_rect
7773: (8)                     if type(name) in (list, tuple):
7774: (12)                        if not type(name[-1]) is int:
7775: (16)                            raise ValueError('need item of full page image list')
7776: (12)                        item = name
7777: (8)                     else:
7778: (12)                        imglist = [i for i in doc.get_page_images(self.number, True) if
name == i[7]]
7779: (12)                        if len(imglist) == 1:
7780: (16)                            item = imglist[0]
7781: (12)                        elif imglist == []:
7782: (16)                            raise ValueError('bad image name')
7783: (12)                        else:
7784: (16)                            raise ValueError("found multiple images named '%s'." % name)
7785: (8)                     xref = item[-1]
7786: (8)                     if xref != 0 or transform is True:
7787: (12)                        try:
7788: (16)                            return self.get_image_rects(item, transform=transform)[0]
7789: (12)                        except Exception:
7790: (16)                            exception_info()
7791: (16)                            return inf_rect
7792: (8)                     pdf_page = self._pdf_page()
```

```
7793: (8)                   val = JM_image_reporter(pdf_page)
7794: (8)                   if not bool(val):
7795: (12)                      return rc
7796: (8)                   for v in val:
7797: (12)                      if v[0] != item[-3]:
7798: (16)                          continue
7799: (12)                      q = Quad(v[1])
7800: (12)                      bbox = q.rect
7801: (12)                      if transform == 0:
7802: (16)                          rc = bbox
7803: (16)                          break
7804: (12)                      hm = Matrix(util_hor_matrix(q.ll, q.lr))
7805: (12)                      h = abs(q.ll - q.ul)
7806: (12)                      w = abs(q.ur - q.ul)
7807: (12)                      m0 = Matrix(1 / w, 0, 0, 1 / h, 0, 0)
7808: (12)                      m = ~(hm * m0)
7809: (12)                      rc = (bbox, m)
7810: (12)                      break
7811: (8)                   val = rc
7812: (8)                   return val
7813: (4)               def get_images(self, full=False):
7814: (8)                   """List of images defined in the page object."""
7815: (8)                   CheckParent(self)
7816: (8)                   return self.parent.get_page_images(self.number, full=full)
7817: (4)               def get_oc_items(self) -> list:
7818: (8)                   """Get OCGs and OCMDs used in the page's contents.
7819: (8)                   Returns:
7820: (12)                      List of items (name, xref, type), where type is one of "ocg" /
"ocmd",
7821: (12)                      and name is the property name.
7822: (8)                   """
7823: (8)                   rc = []
7824: (8)                   for pname, xref in self._get_resource_properties():
7825: (12)                      text = self.parent.xref_object(xref, compressed=True)
7826: (12)                      if "/Type/OCG" in text:
7827: (16)                          octype = "ocg"
7828: (12)                      elif "/Type/OCMD" in text:
7829: (16)                          octype = "ocmd"
7830: (12)                      else:
7831: (16)                          continue
7832: (12)                      rc.append((pname, xref, octype))
7833: (8)                   return rc
7834: (4)               def get_svg_image(self, matrix=None, text_as_path=1):
7835: (8)                   """Make SVG image from page."""
7836: (8)                   CheckParent(self)
7837: (8)                   mediabox = mupdf.fz_bound_page(self.this)
7838: (8)                   ctm = JM_matrix_from_py(matrix)
7839: (8)                   tbounds = mediabox
7840: (8)                   text_option = mupdf.FZ_SVG_TEXT_AS_PATH if text_as_path == 1 else
mupdf.FZ_SVG_TEXT_AS_TEXT
7841: (8)                   tbounds = mupdf.fz_transform_rect(tbounds, ctm)
7842: (8)                   res = mupdf.fz_new_buffer(1024)
7843: (8)                   out = mupdf.FzOutput(res)
7844: (8)                   dev = mupdf.fz_new_svg_device(
7845: (16)                          out,
7846: (16)                          tbounds.x1-tbounds.x0,  # width
7847: (16)                          tbounds.y1-tbounds.y0,  # height
7848: (16)                          text_option,
7849: (16)                          1,
7850: (16)                          )
7851: (8)                   mupdf.fz_run_page(self.this, dev, ctm, mupdf.FzCookie())
7852: (8)                   mupdf.fz_close_device(dev)
7853: (8)                   out.fz_close_output()
7854: (8)                   text = JM_EscapeStrFromBuffer(res)
7855: (8)                   return text
7856: (4)               def get_textbox(
7857: (12)                      page: Page,
7858: (12)                      rect: rect_like,
7859: (12)                      textpage=None,  #: TextPage = None,
```

```
7860: (12)                      ) -> str:
7861: (8)                   tp = textpage
7862: (8)                   if tp is None:
7863: (12)                      tp = page.get_textpage()
7864: (8)                   elif getattr(tp, "parent") != page:
7865: (12)                      raise ValueError("not a textpage of this page")
7866: (8)                   rc = tp.extractTextbox(rect)
7867: (8)                   if textpage is None:
7868: (12)                      del tp
7869: (8)                   return rc
7870: (4)               def get_textpage(self, clip: rect_like = None, flags: int = 0,
matrix=None) -> "TextPage":
7871: (8)                   CheckParent(self)
7872: (8)                   if matrix is None:
7873: (12)                      matrix = Matrix(1, 1)
7874: (8)                   old_rotation = self.rotation
7875: (8)                   if old_rotation != 0:
7876: (12)                      self.set_rotation(0)
7877: (8)                   try:
7878: (12)                      textpage = self._get_textpage(clip, flags=flags, matrix=matrix)
7879: (8)                   finally:
7880: (12)                      if old_rotation != 0:
7881: (16)                          self.set_rotation(old_rotation)
7882: (8)                   textpage = TextPage(textpage)
7883: (8)                   textpage.parent = weakref.proxy(self)
7884: (8)                   return textpage
7885: (4)               def get_texttrace(self):
7886: (8)                   CheckParent(self)
7887: (8)                   old_rotation = self.rotation
7888: (8)                   if old_rotation != 0:
7889: (12)                      self.set_rotation(0)
7890: (8)                   page = self.this
7891: (8)                   rc = []
7892: (8)                   if g_use_extra:
7893: (12)                      dev = extra.JM_new_texttrace_device(rc)
7894: (8)                   else:
7895: (12)                      dev = JM_new_texttrace_device(rc)
7896: (8)                   prect = mupdf.fz_bound_page(page)
7897: (8)                   dev.ptm = mupdf.FzMatrix(1, 0, 0, -1, 0, prect.y1)
7898: (8)                   mupdf.fz_run_page(page, dev, mupdf.FzMatrix(), mupdf.FzCookie())
7899: (8)                   mupdf.fz_close_device(dev)
7900: (8)                   if old_rotation != 0:
7901: (12)                      self.set_rotation(old_rotation)
7902: (8)                   return rc
7903: (4)               def get_xobjects(self):
7904: (8)                   """List of xobjects defined in the page object."""
7905: (8)                   CheckParent(self)
7906: (8)                   return self.parent.get_page_xobjects(self.number)
7907: (4)               def insert_font(self, fontname="helv", fontfile=None, fontbuffer=None,
7908: (19)                              set_simple=False, wmode=0, encoding=0):
7909: (8)                   doc = self.parent
7910: (8)                   if doc is None:
7911: (12)                      raise ValueError("orphaned object: parent is None")
7912: (8)                   idx = 0
7913: (8)                   if fontname.startswith("/"):
7914: (12)                      fontname = fontname[1:]
7915: (8)                   inv_chars = INVALID_NAME_CHARS.intersection(fontname)
7916: (8)                   if inv_chars != set():
7917: (12)                      raise ValueError(f"bad fontname chars {inv_chars}")
7918: (8)                   font = CheckFont(self, fontname)
7919: (8)                   if font is not None:                    # font already in font list of
page
7920: (12)                      xref = font[0]                      # this is the xref
7921: (12)                      if CheckFontInfo(doc, xref):        # also in our document font
list?
7922: (16)                          return xref                     # yes: we are done
7923: (12)                      doc.get_char_widths(xref)
7924: (12)                      return xref
7925: (8)                   bfname = Base14_fontdict.get(fontname.lower(), None) # BaseFont if
```

```
Base-14 font
7926: (8)                   serif = 0
7927: (8)                   CJK_number = -1
7928: (8)                   CJK_list_n = ["china-t", "china-s", "japan", "korea"]
7929: (8)                   CJK_list_s = ["china-ts", "china-ss", "japan-s", "korea-s"]
7930: (8)                   try:
7931: (12)                      CJK_number = CJK_list_n.index(fontname)
7932: (12)                      serif = 0
7933: (8)                   except Exception:
7934: (12)                      if g_exceptions_verbose > 1:    exception_info()
7935: (12)                      pass
7936: (8)                   if CJK_number < 0:
7937: (12)                      try:
7938: (16)                          CJK_number = CJK_list_s.index(fontname)
7939: (16)                          serif = 1
7940: (12)                      except Exception:
7941: (16)                          if g_exceptions_verbose > 1:    exception_info()
7942: (16)                          pass
7943: (8)                   if fontname.lower() in fitz_fontdescriptors.keys():
7944: (12)                      import pymupdf_fonts
7945: (12)                      fontbuffer = pymupdf_fonts.myfont(fontname)  # make a copy
7946: (12)                      del pymupdf_fonts
7947: (8)                   if fontfile is not None:
7948: (12)                      if type(fontfile) is str:
7949: (16)                          fontfile_str = fontfile
7950: (12)                      elif hasattr(fontfile, "absolute"):
7951: (16)                          fontfile_str = str(fontfile)
7952: (12)                      elif hasattr(fontfile, "name"):
7953: (16)                          fontfile_str = fontfile.name
7954: (12)                      else:
7955: (16)                          raise ValueError("bad fontfile")
7956: (8)                   else:
7957: (12)                      fontfile_str = None
7958: (8)                   val = self._insertFont(fontname, bfname, fontfile_str, fontbuffer,
set_simple, idx,
7959: (31)                                         wmode, serif, encoding, CJK_number)
7960: (8)                   if not val:                   # did not work, error return
7961: (12)                      return val
7962: (8)                   xref = val[0]                 # xref of installed font
7963: (8)                   fontdict = val[1]
7964: (8)                   if CheckFontInfo(doc, xref):  # check again: document already has this
font
7965: (12)                      return xref               # we are done
7966: (8)                   doc.get_char_widths(xref, fontdict=fontdict)
7967: (8)                   return xref
7968: (4)               @property
7969: (4)               def is_wrapped(self):
7970: (8)                   """Check if /Contents is in a balanced graphics state."""
7971: (8)                   return self._count_q_balance() == (0, 0)
7972: (4)               @property
7973: (4)               def language(self):
7974: (8)                   """Page language."""
7975: (8)                   pdfpage = _as_pdf_page(self.this, required=False)
7976: (8)                   if not pdfpage.m_internal:
7977: (12)                      return
7978: (8)                   lang = mupdf.pdf_dict_get_inheritable(pdfpage.obj(), PDF_NAME('Lang'))
7979: (8)                   if not lang.m_internal:
7980: (12)                      return
7981: (8)                   return mupdf.pdf_to_str_buf(lang)
7982: (4)               def links(self, kinds=None):
7983: (8)                   """ Generator over the links of a page.
7984: (8)                   Args:
7985: (12)                      kinds: (list) link kinds to subselect from. If none,
7986: (19)                             all links are returned. E.g. kinds=[LINK_URI]
7987: (19)                             will only yield URI links.
7988: (8)                   """
7989: (8)                   all_links = self.get_links()
7990: (8)                   for link in all_links:
7991: (12)                      if kinds is None or link["kind"] in kinds:
```

```
(16)                            yield (link)
(4)                 def load_annot(self, ident: typing.Union[str, int]) -> Annot:
(8)                     """Load an annot by name (/NM key) or xref.
(8)                     Args:
(12)                        ident: identifier, either name (str) or xref (int).
(8)                     """
(8)                     CheckParent(self)
(8)                     if type(ident) is str:
(12)                        xref = 0
(12)                        name = ident
(8)                     elif type(ident) is int:
(12)                        xref = ident
(12)                        name = None
(8)                     else:
(12)                        raise ValueError("identifier must be a string or integer")
(8)                     val = self._load_annot(name, xref)
(8)                     if not val:
(12)                        return val
(8)                     val.thisown = True
(8)                     val.parent = weakref.proxy(self)
(8)                     self._annot_refs[id(val)] = val
(8)                     return val
(4)                 def load_links(self):
(8)                     """Get first Link."""
(8)                     CheckParent(self)
(8)                     val = mupdf.fz_load_links( self.this)
(8)                     if not val.m_internal:
(12)                        return
(8)                     val = Link( val)
(8)                     val.thisown = True
(8)                     val.parent = weakref.proxy(self) # owning page object
(8)                     self._annot_refs[id(val)] = val
(8)                     val.xref = 0
(8)                     val.id = ""
(8)                     if self.parent.is_pdf:
(12)                        xrefs = self.annot_xrefs()
(12)                        xrefs = [x for x in xrefs if x[1] == mupdf.PDF_ANNOT_LINK]
(12)                        if xrefs:
(16)                            link_id = xrefs[0]
(16)                            val.xref = link_id[0]
(16)                            val.id = link_id[2]
(8)                     else:
(12)                        val.xref = 0
(12)                        val.id = ""
(8)                     return val
(4)                 def load_widget( self, xref):
(8)                     """Load a widget by its xref."""
(8)                     CheckParent(self)
(8)                     page = _as_pdf_page(self.this)
(8)                     annot = JM_get_widget_by_xref( page, xref)
(8)                     val = annot
(8)                     if not val:
(12)                        return val
(8)                     val.thisown = True
(8)                     val.parent = weakref.proxy(self)
(8)                     self._annot_refs[id(val)] = val
(8)                     widget = Widget()
(8)                     TOOLS._fill_widget(val, widget)
(8)                     val = widget
(8)                     return val
(4)                 @property
(4)                 def mediabox(self):
(8)                     """The MediaBox."""
(8)                     CheckParent(self)
(8)                     page = self._pdf_page(required=False)
(8)                     if not page.m_internal:
(12)                        rect = mupdf.fz_bound_page( self.this)
(8)                     else:
(12)                        rect = JM_mediabox( page.obj())
```

```
8061: (8)                   return Rect(rect)
8062: (4)               @property
8063: (4)               def mediabox_size(self):
8064: (8)                   return Point(self.mediabox.x1, self.mediabox.y1)
8065: (4)               def read_contents(self):
8066: (8)                   """All /Contents streams concatenated to one bytes object."""
8067: (8)                   return TOOLS._get_all_contents(self)
8068: (4)               def refresh(self):
8069: (8)                   """Refresh page after link/annot/widget updates."""
8070: (8)                   CheckParent(self)
8071: (8)                   doc = self.parent
8072: (8)                   page = doc.reload_page(self)
8073: (8)                   self.this = page
8074: (4)               @property
8075: (4)               def rotation(self):
8076: (8)                   """Page rotation."""
8077: (8)                   CheckParent(self)
8078: (8)                   page = _as_pdf_page(self.this, required=0)
8079: (8)                   if not page.m_internal:
8080: (12)                      return 0
8081: (8)                   return JM_page_rotation(page)
8082: (4)               @property
8083: (4)               def rotation_matrix(self) -> Matrix:
8084: (8)                   """Reflects page rotation."""
8085: (8)                   return Matrix(TOOLS._rotate_matrix(self))
8086: (4)               def run(self, dw, m):
8087: (8)                   """Run page through a device.
8088: (8)                   dw: DeviceWrapper
8089: (8)                   """
8090: (8)                   CheckParent(self)
8091: (8)                   mupdf.fz_run_page(self.this, dw.device, JM_matrix_from_py(m),
mupdf.FzCookie())
8092: (4)               def set_artbox(self, rect):
8093: (8)                   """Set the ArtBox."""
8094: (8)                   return self._set_pagebox("ArtBox", rect)
8095: (4)               def set_bleedbox(self, rect):
8096: (8)                   """Set the BleedBox."""
8097: (8)                   return self._set_pagebox("BleedBox", rect)
8098: (4)               def set_contents(self, xref):
8099: (8)                   """Set object at 'xref' as the page's /Contents."""
8100: (8)                   CheckParent(self)
8101: (8)                   doc = self.parent
8102: (8)                   if doc.is_closed:
8103: (12)                      raise ValueError("document closed")
8104: (8)                   if not doc.is_pdf:
8105: (12)                      raise ValueError("is no PDF")
8106: (8)                   if xref not in range(1, doc.xref_length()):
8107: (12)                      raise ValueError("bad xref")
8108: (8)                   if not doc.xref_is_stream(xref):
8109: (12)                      raise ValueError("xref is no stream")
8110: (8)                   doc.xref_set_key(self.xref, "Contents", "%i 0 R" % xref)
8111: (4)               def set_cropbox(self, rect):
8112: (8)                   """Set the CropBox. Will also change Page.rect."""
8113: (8)                   return self._set_pagebox("CropBox", rect)
8114: (4)               def set_language(self, language=None):
8115: (8)                   """Set PDF page default language."""
8116: (8)                   CheckParent(self)
8117: (8)                   pdfpage = _as_pdf_page(self.this)
8118: (8)                   if not language:
8119: (12)                      mupdf.pdf_dict_del(pdfpage.obj(), PDF_NAME('Lang'))
8120: (8)                   else:
8121: (12)                      lang = mupdf.fz_text_language_from_string(language)
8122: (12)                      assert hasattr(mupdf, 'fz_string_from_text_language2')
8123: (12)                      mupdf.pdf_dict_put_text_string(
8124: (20)                              pdfpage.obj,
8125: (20)                              PDF_NAME('Lang'),
8126: (20)                              mupdf.fz_string_from_text_language2(lang)
8127: (20)                              )
8128: (4)               def set_mediabox(self, rect):
```

```
8129: (8)                    """Set the MediaBox."""
8130: (8)                    CheckParent(self)
8131: (8)                    page = self._pdf_page()
8132: (8)                    mediabox = JM_rect_from_py(rect)
8133: (8)                    if (mupdf.fz_is_empty_rect(mediabox)
8134: (16)                           or mupdf.fz_is_infinite_rect(mediabox)
8135: (16)                           ):
8136: (12)                       raise ValueError( MSG_BAD_RECT)
8137: (8)                    mupdf.pdf_dict_put_rect( page.obj(), PDF_NAME('MediaBox'), mediabox)
8138: (8)                    mupdf.pdf_dict_del( page.obj(), PDF_NAME('CropBox'))
8139: (8)                    mupdf.pdf_dict_del( page.obj(), PDF_NAME('ArtBox'))
8140: (8)                    mupdf.pdf_dict_del( page.obj(), PDF_NAME('BleedBox'))
8141: (8)                    mupdf.pdf_dict_del( page.obj(), PDF_NAME('TrimBox'))
8142: (4)                def set_rotation(self, rotation):
8143: (8)                    """Set page rotation."""
8144: (8)                    CheckParent(self)
8145: (8)                    page = _as_pdf_page(self.this)
8146: (8)                    rot = JM_norm_rotation(rotation)
8147: (8)                    mupdf.pdf_dict_put_int( page.obj(), PDF_NAME('Rotate'), rot)
8148: (4)                def set_trimbox(self, rect):
8149: (8)                    """Set the TrimBox."""
8150: (8)                    return self._set_pagebox("TrimBox", rect)
8151: (4)                @property
8152: (4)                def transformation_matrix(self):
8153: (8)                    """Page transformation matrix."""
8154: (8)                    CheckParent(self)
8155: (8)                    ctm = mupdf.FzMatrix()
8156: (8)                    page = self._pdf_page(required=False)
8157: (8)                    if not page.m_internal:
8158: (12)                       return JM_py_from_matrix(ctm)
8159: (8)                    mediabox = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)    # fixme: original
code passed mediabox=NULL.
8160: (8)                    mupdf.pdf_page_transform(page, mediabox, ctm)
8161: (8)                    val = JM_py_from_matrix(ctm)
8162: (8)                    if self.rotation % 360 == 0:
8163: (12)                       val = Matrix(val)
8164: (8)                    else:
8165: (12)                       val = Matrix(1, 0, 0, -1, 0, self.cropbox.height)
8166: (8)                    return val
8167: (4)                @property
8168: (4)                def trimbox(self):
8169: (8)                    """The TrimBox"""
8170: (8)                    rect = self._other_box("TrimBox")
8171: (8)                    if rect is None:
8172: (12)                       return self.cropbox
8173: (8)                    mb = self.mediabox
8174: (8)                    return Rect(rect[0], mb.y1 - rect[3], rect[2], mb.y1 - rect[1])
8175: (4)                def widgets(self, types=None):
8176: (8)                    """ Generator over the widgets of a page.
8177: (8)                    Args:
8178: (12)                       types: (list) field types to subselect from. If none,
8179: (20)                              all fields are returned. E.g. types=[PDF_WIDGET_TYPE_TEXT]
8180: (20)                              will only yield text fields.
8181: (8)                    """
8182: (8)                    widget_xrefs = [a[0] for a in self.annot_xrefs() if a[1] ==
mupdf.PDF_ANNOT_WIDGET]
8183: (8)                    for xref in widget_xrefs:
8184: (12)                       widget = self.load_widget(xref)
8185: (12)                       if types is None or widget.field_type in types:
8186: (16)                           yield (widget)
8187: (4)                def wrap_contents(self):
8188: (8)                    """Ensure page is in a balanced graphics state."""
8189: (8)                    push, pop = self._count_q_balance()  # count missing "q"/"Q" commands
8190: (8)                    if push > 0:  # prepend required push commands
8191: (12)                       prepend = b"q\n" * push
8192: (12)                       TOOLS._insert_contents(self, prepend, False)
8193: (8)                    if pop > 0:  # append required pop commands
8194: (12)                       append = b"\nQ" * pop + b"\n"
8195: (12)                       TOOLS._insert_contents(self, append, True)
```

```
8196: (4)               @property
8197: (4)               def xref(self):
8198: (8)                   """PDF xref number of page."""
8199: (8)                   CheckParent(self)
8200: (8)                   return self.parent.page_xref(self.number)
8201: (4)               rect = property(bound, doc="page rectangle")
8202: (0)           class Pixmap:
8203: (4)               def __init__(self, *args):
8204: (8)                   """
8205: (8)                   Pixmap(colorspace, irect, alpha) - empty pixmap.
8206: (8)                   Pixmap(colorspace, src) - copy changing colorspace.
8207: (8)                   Pixmap(src, width, height,[clip]) - scaled copy, float dimensions.
8208: (8)                   Pixmap(src, alpha=1) - copy and add or drop alpha channel.
8209: (8)                   Pixmap(filename) - from an image in a file.
8210: (8)                   Pixmap(image) - from an image in memory (bytes).
8211: (8)                   Pixmap(colorspace, width, height, samples, alpha) - from samples data.
8212: (8)                   Pixmap(PDFdoc, xref) - from an image at xref in a PDF document.
8213: (8)                   """
8214: (8)                   if 0:
8215: (12)                      pass
8216: (8)                   elif args_match(args,
8217: (16)                          (Colorspace, mupdf.FzColorspace),
8218: (16)                          (mupdf.FzRect, mupdf.FzIrect, IRect, Rect, tuple)
8219: (16)                          ):
8220: (12)                      cs, rect = args
8221: (12)                      alpha = 0
8222: (12)                      pm = mupdf.fz_new_pixmap_with_bbox(cs, JM_irect_from_py(rect),
mupdf.FzSeparations(0), alpha)
8223: (12)                      self.this = pm
8224: (8)                   elif args_match(args,
8225: (16)                          (Colorspace, mupdf.FzColorspace),
8226: (16)                          (mupdf.FzRect, mupdf.FzIrect, IRect, Rect, tuple),
8227: (16)                          (int, bool)
8228: (16)                          ):
8229: (12)                      cs, rect, alpha = args
8230: (12)                      pm = mupdf.fz_new_pixmap_with_bbox(cs, JM_irect_from_py(rect),
mupdf.FzSeparations(0), alpha)
8231: (12)                      self.this = pm
8232: (8)                   elif args_match(args, (Colorspace, mupdf.FzColorspace, type(None)),
(Pixmap, mupdf.FzPixmap)):
8233: (12)                      cs, spix = args
8234: (12)                      if isinstance(cs, Colorspace):
8235: (16)                          cs = cs.this
8236: (12)                      elif cs is None:
8237: (16)                          cs = mupdf.FzColorspace(None)
8238: (12)                      if isinstance(spix, Pixmap):
8239: (16)                          spix = spix.this
8240: (12)                      if not mupdf.fz_pixmap_colorspace(spix).m_internal:
8241: (16)                          raise ValueError( "source colorspace must not be None")
8242: (12)                      if cs.m_internal:
8243: (16)                          self.this = mupdf.fz_convert_pixmap(
8244: (24)                                  spix,
8245: (24)                                  cs,
8246: (24)                                  mupdf.FzColorspace(),
8247: (24)                                  mupdf.FzDefaultColorspaces(None),
8248: (24)                                  mupdf.FzColorParams(),
8249: (24)                                  1
8250: (24)                                  )
8251: (12)                      else:
8252: (16)                          self.this = mupdf.fz_new_pixmap_from_alpha_channel( spix)
8253: (16)                          if not self.this.m_internal:
8254: (20)                              raise RuntimeError( MSG_PIX_NOALPHA)
8255: (8)                   elif args_match(args, (Pixmap, mupdf.FzPixmap), (Pixmap,
mupdf.FzPixmap)):
8256: (12)                      spix, mpix = args
8257: (12)                      if isinstance(spix, Pixmap):
8258: (16)                          spix = spix.this
8259: (12)                      if isinstance(mpix, Pixmap):
8260: (16)                          mpix = mpix.this
```

```
8261: (12)                        spm = spix
8262: (12)                        mpm = mpix
8263: (12)                        if not spix.m_internal: # intercept NULL for spix: make alpha only
pix
8264: (16)                            dst = mupdf.fz_new_pixmap_from_alpha_channel(mpm)
8265: (16)                            if not dst.m_internal:
8266: (20)                                raise RuntimeError( MSG_PIX_NOALPHA)
8267: (12)                        else:
8268: (16)                            dst = mupdf.fz_new_pixmap_from_color_and_mask(spm, mpm)
8269: (12)                        self.this = dst
8270: (8)                     elif (args_match(args, (Pixmap, mupdf.FzPixmap), (float, int), (float,
int), None) or
8271: (13)                         args_match(args, (Pixmap, mupdf.FzPixmap), (float, int), (float,
int))):
8272: (12)                        if mupdf_version_tuple < (1, 23, 8):
8273: (16)                            assert 0, f'Cannot handle {args=} because fz_scale_pixmap()
and fz_scale_pixmap_cached() are not declared in MuPDF headers'
8274: (12)                        if len(args) == 3:
8275: (16)                            spix, w, h = args
8276: (16)                            bbox = mupdf.FzIrect(mupdf.fz_infinite_irect)
8277: (12)                        else:
8278: (16)                            spix, w, h, clip = args
8279: (16)                            bbox = JM_irect_from_py(clip)
8280: (12)                        spix, w, h, clip = args
8281: (12)                        src_pix = spix.this if isinstance(spix, Pixmap) else spix
8282: (12)                        bbox = JM_irect_from_py(clip)
8283: (12)                        if not mupdf.fz_is_infinite_irect(bbox):
8284: (16)                            pm = mupdf.fz_scale_pixmap(src_pix, src_pix.x(), src_pix.y(),
w, h, bbox)
8285: (12)                        else:
8286: (16)                            pm = mupdf.fz_scale_pixmap(src_pix, src_pix.x(), src_pix.y(),
w, h, mupdf.FzIrect(mupdf.fz_infinite_irect))
8287: (12)                        self.this = pm
8288: (8)                     elif args_match(args, str, (Pixmap, mupdf.FzPixmap)) and args[0] ==
'raw':
8289: (12)                        _, pm = args
8290: (12)                        if isinstance(pm, Pixmap):
8291: (16)                            pm = pm.this
8292: (12)                        self.this = pm
8293: (8)                     elif args_match(args, (Pixmap, mupdf.FzPixmap), (int, None)):
8294: (12)                        spix = args[0]
8295: (12)                        alpha = args[1] if len(args) == 2 else 1
8296: (12)                        src_pix = spix.this if isinstance(spix, Pixmap) else spix
8297: (12)                        if not _INRANGE(alpha, 0, 1):
8298: (16)                            raise ValueError( "bad alpha value")
8299: (12)                        cs = mupdf.fz_pixmap_colorspace(src_pix)
8300: (12)                        if not cs.m_internal and not alpha:
8301: (16)                            raise ValueError( "cannot drop alpha for 'NULL' colorspace")
8302: (12)                        seps = mupdf.FzSeparations()
8303: (12)                        n = mupdf.fz_pixmap_colorants(src_pix)
8304: (12)                        w = mupdf.fz_pixmap_width(src_pix)
8305: (12)                        h = mupdf.fz_pixmap_height(src_pix)
8306: (12)                        pm = mupdf.fz_new_pixmap(cs, w, h, seps, alpha)
8307: (12)                        pm.m_internal.x = src_pix.m_internal.x
8308: (12)                        pm.m_internal.y = src_pix.m_internal.y
8309: (12)                        pm.m_internal.xres = src_pix.m_internal.xres
8310: (12)                        pm.m_internal.yres = src_pix.m_internal.yres
8311: (12)                        if 1:
8312: (16)                            mupdf.ll_fz_pixmap_copy( pm.m_internal, src_pix.m_internal, n)
8313: (12)                        elif 1:
8314: (16)                            src_view = mupdf.fz_pixmap_samples_memoryview( src_pix)
8315: (16)                            pm_view = mupdf.fz_pixmap_samples_memoryview( pm)
8316: (16)                            if src_pix.alpha() == pm.alpha():   # identical samples
8317: (20)                                size = w * h * (n + alpha)
8318: (20)                                pm_view[ 0 : size] = src_view[ 0 : size]
8319: (16)                            else:
8320: (20)                                tptr = 0
8321: (20)                                sptr = 0
8322: (20)                                pm_stride = pm.stride()
```

```
8323: (20)                                pm_n = pm.n()
8324: (20)                                pm_alpha = pm.alpha()
8325: (20)                                src_stride = src_pix.stride()
8326: (20)                                src_n = src_pix.n()
8327: (20)                                for y in range( h):
8328: (24)                                    for x in range( w):
8329: (28)                                        pm_i = pm_stride * y + pm_n * x
8330: (28)                                        src_i = src_stride * y + src_n * x
8331: (28)                                        pm_view[ pm_i : pm_i + n] = src_view[ src_i :
src_i + n]
8332: (28)                                        if pm_alpha:
8333: (32)                                            pm_view[ pm_i + n] = 255
8334: (12)                        else:
8335: (16)                            if src_pix.alpha() == pm.alpha():   # identical samples
8336: (20)                                for i in range(w * h * (n + alpha)):
8337: (24)                                    mupdf.fz_samples_set(pm, i,
mupdf.fz_samples_get(src_pix, i))
8338: (16)                            else:
8339: (20)                                tptr = 0
8340: (20)                                sptr = 0
8341: (20)                                src_pix_alpha = src_pix.alpha()
8342: (20)                                for i in range(w * h):
8343: (24)                                    for j in range(n):
8344: (28)                                        mupdf.fz_samples_set(pm, tptr + j,
mupdf.fz_samples_get(src_pix, sptr + j))
8345: (24)                                    tptr += n
8346: (24)                                    if pm.alpha():
8347: (28)                                        mupdf.fz_samples_set(pm, tptr, 255)
8348: (28)                                        tptr += 1
8349: (24)                                    sptr += n + src_pix_alpha
8350: (12)                        self.this = pm
8351: (8)                     elif args_match(args, (mupdf.FzColorspace, Colorspace), int, int,
None, (int, bool)):
8352: (12)                        cs, w, h, samples, alpha = args
8353: (12)                        if isinstance(cs, Colorspace):
8354: (16)                            cs = cs.this
8355: (16)                            assert isinstance(cs, mupdf.FzColorspace)
8356: (12)                        n = mupdf.fz_colorspace_n(cs)
8357: (12)                        stride = (n + alpha) * w
8358: (12)                        seps = mupdf.FzSeparations()
8359: (12)                        pm = mupdf.fz_new_pixmap(cs, w, h, seps, alpha)
8360: (12)                        if isinstance( samples, (bytes, bytearray)):
8361: (16)                            samples2 = mupdf.python_buffer_data(samples)
8362: (16)                            size = len(samples)
8363: (12)                        else:
8364: (16)                            res = JM_BufferFromBytes(samples)
8365: (16)                            if not res.m_internal:
8366: (20)                                raise ValueError( "bad samples data")
8367: (16)                            size, c = mupdf.fz_buffer_storage(res)
8368: (16)                            samples2 = mupdf.python_buffer_data(samples) # raw swig proxy
for `const unsigned char*`.
8369: (12)                        if stride * h != size:
8370: (16)                            raise ValueError( f"bad samples length {w=} {h=} {alpha=} {n=}
{stride=} {size=}")
8371: (12)                        mupdf.ll_fz_pixmap_copy_raw( pm.m_internal, samples2)
8372: (12)                        self.this = pm
8373: (8)                     elif args_match(args, None):
8374: (12)                        imagedata, = args
8375: (12)                        name = 'name'
8376: (12)                        if hasattr(imagedata, "resolve"):
8377: (16)                            fname = imagedata.__str__()
8378: (16)                            if fname:
8379: (20)                                img = mupdf.fz_new_image_from_file(fname)
8380: (12)                        elif hasattr(imagedata, name):
8381: (16)                            fname = imagedata.name
8382: (16)                            if fname:
8383: (20)                                img = mupdf.fz_new_image_from_file(fname)
8384: (12)                        elif isinstance(imagedata, str):
8385: (16)                            img = mupdf.fz_new_image_from_file(imagedata)
```

```
8386: (12)                      else:
8387: (16)                          res = JM_BufferFromBytes(imagedata)
8388: (16)                          if not res.m_internal or not res.m_internal.len:
8389: (20)                              raise ValueError( "bad image data")
8390: (16)                          img = mupdf.fz_new_image_from_buffer(res)
8391: (12)                      pm, w, h = mupdf.fz_get_pixmap_from_image(
8392: (20)                              img,
8393: (20)                              mupdf.FzIrect(FZ_MIN_INF_RECT, FZ_MIN_INF_RECT,
FZ_MAX_INF_RECT, FZ_MAX_INF_RECT),
8394: (20)                              mupdf.FzMatrix( img.w(), 0, 0, img.h(), 0, 0),
8395: (20)                              )
8396: (12)                      xres, yres = mupdf.fz_image_resolution(img)
8397: (12)                      pm.m_internal.xres = xres
8398: (12)                      pm.m_internal.yres = yres
8399: (12)                      self.this = pm
8400: (8)                   elif args_match(args, (Document, mupdf.FzDocument), int):
8401: (12)                      doc, xref = args
8402: (12)                      pdf = _as_pdf_document(doc)
8403: (12)                      xreflen = mupdf.pdf_xref_len(pdf)
8404: (12)                      if not _INRANGE(xref, 1, xreflen-1):
8405: (16)                          raise ValueError( MSG_BAD_XREF)
8406: (12)                      ref = mupdf.pdf_new_indirect(pdf, xref, 0)
8407: (12)                      type_ = mupdf.pdf_dict_get(ref, PDF_NAME('Subtype'))
8408: (12)                      if (not mupdf.pdf_name_eq(type_, PDF_NAME('Image'))
8409: (20)                              and not mupdf.pdf_name_eq(type_, PDF_NAME('Alpha'))
8410: (20)                              and not mupdf.pdf_name_eq(type_, PDF_NAME('Luminosity'))
8411: (20)                              ):
8412: (16)                          raise ValueError( MSG_IS_NO_IMAGE)
8413: (12)                      img = mupdf.pdf_load_image(pdf, ref)
8414: (12)                      pix, w, h = mupdf.fz_get_pixmap_from_image(
8415: (20)                              img,
8416: (20)                              mupdf.FzIrect(FZ_MIN_INF_RECT, FZ_MIN_INF_RECT,
FZ_MAX_INF_RECT, FZ_MAX_INF_RECT),
8417: (20)                              mupdf.FzMatrix(img.w(), 0, 0, img.h(), 0, 0),
8418: (20)                              )
8419: (12)                      self.this = pix
8420: (8)                   else:
8421: (12)                      text = 'Unrecognised args for constructing Pixmap:\n'
8422: (12)                      for arg in args:
8423: (16)                          text += f'    {type(arg)}: {arg}\n'
8424: (12)                      raise Exception( text)
8425: (8)                   self._memory_view = None
8426: (4)               def __len__(self):
8427: (8)                   return self.size
8428: (4)               def __repr__(self):
8429: (8)                   if not type(self) is Pixmap: return
8430: (8)                   if self.colorspace:
8431: (12)                      return "Pixmap(%s, %s, %s)" %
(self.colorspace.this.m_internal.name, self.irect, self.alpha)
8432: (8)                   else:
8433: (12)                      return "Pixmap(%s, %s, %s)" % ('None', self.irect, self.alpha)
8434: (4)               def _tobytes(self, format_, jpg_quality):
8435: (8)                   '''
8436: (8)                   Pixmap._tobytes
8437: (8)                   '''
8438: (8)                   pm = self.this
8439: (8)                   size = mupdf.fz_pixmap_stride(pm) * pm.h()
8440: (8)                   res = mupdf.fz_new_buffer(size)
8441: (8)                   out = mupdf.FzOutput(res)
8442: (8)                   if   format_ == 1:  mupdf.fz_write_pixmap_as_png(out, pm)
8443: (8)                   elif format_ == 2:  mupdf.fz_write_pixmap_as_pnm(out, pm)
8444: (8)                   elif format_ == 3:  mupdf.fz_write_pixmap_as_pam(out, pm)
8445: (8)                   elif format_ == 5:  mupdf.fz_write_pixmap_as_psd(out, pm)
8446: (8)                   elif format_ == 6:  mupdf.fz_write_pixmap_as_ps(out, pm)
8447: (8)                   elif format_ == 7:
8448: (12)                      if mupdf_version_tuple < (1, 24):
8449: (16)                          mupdf.fz_write_pixmap_as_jpeg(out, pm, jpg_quality)
8450: (12)                      else:
8451: (16)                          mupdf.fz_write_pixmap_as_jpeg(out, pm, jpg_quality, 0)
```

```
8452: (8)                    else:
8453: (12)                       mupdf.fz_write_pixmap_as_png(out, pm)
8454: (8)                    out.fz_close_output()
8455: (8)                    barray = JM_BinFromBuffer(res)
8456: (8)                    return barray
8457: (4)                def _writeIMG(self, filename, format_, jpg_quality):
8458: (8)                    pm = self.this
8459: (8)                    if   format_ == 1:  mupdf.fz_save_pixmap_as_png(pm, filename)
8460: (8)                    elif format_ == 2:  mupdf.fz_save_pixmap_as_pnm(pm, filename)
8461: (8)                    elif format_ == 3:  mupdf.fz_save_pixmap_as_pam(pm, filename)
8462: (8)                    elif format_ == 5:  mupdf.fz_save_pixmap_as_psd(pm, filename)
8463: (8)                    elif format_ == 6:  mupdf.fz_save_pixmap_as_ps(pm, filename)
8464: (8)                    elif format_ == 7:  mupdf.fz_save_pixmap_as_jpeg(pm, filename,
jpg_quality)
8465: (8)                    else:               mupdf.fz_save_pixmap_as_png(pm, filename)
8466: (4)                @property
8467: (4)                def alpha(self):
8468: (8)                    """Indicates presence of alpha channel."""
8469: (8)                    return mupdf.fz_pixmap_alpha(self.this)
8470: (4)                def clear_with(self, value=None, bbox=None):
8471: (8)                    """Fill all color components with same value."""
8472: (8)                    if value is None:
8473: (12)                       mupdf.fz_clear_pixmap(self.this)
8474: (8)                    elif bbox is None:
8475: (12)                       mupdf.fz_clear_pixmap_with_value(self.this, value)
8476: (8)                    else:
8477: (12)                       JM_clear_pixmap_rect_with_value(self.this, value,
JM_irect_from_py(bbox))
8478: (4)                def color_count(self, colors=0, clip=None):
8479: (8)                    '''
8480: (8)                    Return count of each color.
8481: (8)                    '''
8482: (8)                    pm = self.this
8483: (8)                    rc = JM_color_count( pm, clip)
8484: (8)                    if not rc:
8485: (12)                       raise RuntimeError( MSG_COLOR_COUNT_FAILED)
8486: (8)                    if not colors:
8487: (12)                       return len( rc)
8488: (8)                    return rc
8489: (4)                def color_topusage(self, clip=None):
8490: (8)                    """Return most frequent color and its usage ratio."""
8491: (8)                    allpixels = 0
8492: (8)                    cnt = 0
8493: (8)                    if clip is not None and self.irect in Rect(clip):
8494: (12)                       clip = self.irect
8495: (8)                    for pixel, count in self.color_count(colors=True,clip=clip).items():
8496: (12)                       allpixels += count
8497: (12)                       if count > cnt:
8498: (16)                           cnt = count
8499: (16)                           maxpixel = pixel
8500: (8)                    if not allpixels:
8501: (12)                       return (1, bytes([255] * self.n))
8502: (8)                    return (cnt / allpixels, maxpixel)
8503: (4)                @property
8504: (4)                def colorspace(self):
8505: (8)                    """Pixmap Colorspace."""
8506: (8)                    return Colorspace(mupdf.fz_pixmap_colorspace(self.this))
8507: (4)                def copy(self, src, bbox):
8508: (8)                    """Copy bbox from another Pixmap."""
8509: (8)                    pm = self.this
8510: (8)                    src_pix = src.this
8511: (8)                    if not mupdf.fz_pixmap_colorspace(src_pix):
8512: (12)                       raise ValueError( "cannot copy pixmap with NULL colorspace")
8513: (8)                    if pm.alpha() != src_pix.alpha():
8514: (12)                       raise ValueError( "source and target alpha must be equal")
8515: (8)                    mupdf.fz_copy_pixmap_rect(pm, src_pix, JM_irect_from_py(bbox),
mupdf.FzDefaultColorspaces(None))
8516: (4)                @property
8517: (4)                def digest(self):
```

```
8518: (8)                    """MD5 digest of pixmap (bytes)."""
8519: (8)                    ret = mupdf.fz_md5_pixmap2(self.this)
8520: (8)                    return bytes(ret)
8521: (4)                def gamma_with(self, gamma):
8522: (8)                    """Apply correction with some float.
8523: (8)                    gamma=1 is a no-op."""
8524: (8)                    if not mupdf.fz_pixmap_colorspace( self.this):
8525: (12)                       message_warning("colorspace invalid for function")
8526: (12)                       return
8527: (8)                    mupdf.fz_gamma_pixmap( self.this, gamma)
8528: (4)                @property
8529: (4)                def h(self):
8530: (8)                    """The height."""
8531: (8)                    return mupdf.fz_pixmap_height(self.this)
8532: (4)                def invert_irect(self, bbox=None):
8533: (8)                    """Invert the colors inside a bbox."""
8534: (8)                    pm = self.this
8535: (8)                    if not mupdf.fz_pixmap_colorspace(pm).m_internal:
8536: (12)                       message_warning("ignored for stencil pixmap")
8537: (12)                       return False
8538: (8)                    r = JM_irect_from_py(bbox)
8539: (8)                    if mupdf.fz_is_infinite_irect(r):
8540: (12)                       r = mupdf.fz_pixmap_bbox( pm)
8541: (8)                    return bool(JM_invert_pixmap_rect( pm, r))
8542: (4)                @property
8543: (4)                def irect(self):
8544: (8)                    """Pixmap bbox - an IRect object."""
8545: (8)                    val = mupdf.fz_pixmap_bbox(self.this)
8546: (8)                    return JM_py_from_irect( val)
8547: (4)                @property
8548: (4)                def is_monochrome(self):
8549: (8)                    """Check if pixmap is monochrome."""
8550: (8)                    return mupdf.fz_is_pixmap_monochrome( self.this)
8551: (4)                @property
8552: (4)                def is_unicolor(self):
8553: (8)                    '''
8554: (8)                    Check if pixmap has only one color.
8555: (8)                    '''
8556: (8)                    pm = self.this
8557: (8)                    n = pm.n()
8558: (8)                    count = pm.w() * pm.h() * n
8559: (8)                    def _pixmap_read_samples(pm, offset, n):
8560: (12)                       ret = list()
8561: (12)                       for i in range(n):
8562: (16)                           ret.append(mupdf.fz_samples_get(pm, offset+i))
8563: (12)                       return ret
8564: (8)                    sample0 = _pixmap_read_samples( pm, 0, n)
8565: (8)                    for offset in range( n, count, n):
8566: (12)                       sample = _pixmap_read_samples( pm, offset, n)
8567: (12)                       if sample != sample0:
8568: (16)                           return False
8569: (8)                    return True
8570: (4)                @property
8571: (4)                def n(self):
8572: (8)                    """The size of one pixel."""
8573: (8)                    if g_use_extra:
8574: (12)                       def n2(self):
8575: (16)                           return extra.pixmap_n(self.this)
8576: (12)                       self.__class__.n = property(n2)
8577: (12)                       return self.n
8578: (8)                    return mupdf.fz_pixmap_components(self.this)
8579: (4)                def pdfocr_save(self, filename, compress=1, language=None, tessdata=None):
8580: (8)                    '''
8581: (8)                    Save pixmap as an OCR-ed PDF page.
8582: (8)                    '''
8583: (8)                    if not TESSDATA_PREFIX and not tessdata:
8584: (12)                       raise RuntimeError('No OCR support: TESSDATA_PREFIX not set')
8585: (8)                    opts = mupdf.FzPdfocrOptions()
8586: (8)                    opts.compress = compress
```

```
8587: (8)                    if language:
8588: (12)                       opts.language_set2( language)
8589: (8)                    if tessdata:
8590: (12)                       opts.datadir_set2( tessdata)
8591: (8)                    pix = self.this
8592: (8)                    if isinstance(filename, str):
8593: (12)                       mupdf.fz_save_pixmap_as_pdfocr( pix, filename, 0, opts)
8594: (8)                    else:
8595: (12)                       out = JM_new_output_fileptr( filename)
8596: (12)                       mupdf.fz_write_pixmap_as_pdfocr( out, pix, opts)
8597: (12)                       out.fz_close_output()
8598: (4)                def pdfocr_tobytes(self, compress=True, language="eng", tessdata=None):
8599: (8)                    """Save pixmap as an OCR-ed PDF page.
8600: (8)                    Args:
8601: (12)                       compress: (bool) compress, default 1 (True).
8602: (12)                       language: (str) language(s) occurring on page, default "eng"
(English),
8603: (20)                               multiples like "eng+ger" for English and German.
8604: (12)                       tessdata: (str) folder name of Tesseract's language support. Must
be
8605: (20)                               given if environment variable TESSDATA_PREFIX is not set.
8606: (8)                    Notes:
8607: (12)                       On failure, make sure Tesseract is installed and you have set the
8608: (12)                       environment variable "TESSDATA_PREFIX" to the folder containing
your
8609: (12)                       Tesseract's language support data.
8610: (8)                    """
8611: (8)                    if not TESSDATA_PREFIX and not tessdata:
8612: (12)                       raise RuntimeError('No OCR support: TESSDATA_PREFIX not set')
8613: (8)                    from io import BytesIO
8614: (8)                    bio = BytesIO()
8615: (8)                    self.pdfocr_save(bio, compress=compress, language=language,
tessdata=tessdata)
8616: (8)                    return bio.getvalue()
8617: (4)                def pil_save(self, *args, **kwargs):
8618: (8)                    """Write to image file using Pillow.
8619: (8)                    Args are passed to Pillow's Image.save method, see their
documentation.
8620: (8)                    Use instead of save when other output formats are desired.
8621: (8)                    """
8622: (8)                    try:
8623: (12)                       from PIL import Image
8624: (8)                    except ImportError:
8625: (12)                       message("PIL/Pillow not installed")
8626: (12)                       raise
8627: (8)                    cspace = self.colorspace
8628: (8)                    if cspace is None:
8629: (12)                       mode = "L"
8630: (8)                    elif cspace.n == 1:
8631: (12)                       mode = "L" if self.alpha == 0 else "LA"
8632: (8)                    elif cspace.n == 3:
8633: (12)                       mode = "RGB" if self.alpha == 0 else "RGBA"
8634: (8)                    else:
8635: (12)                       mode = "CMYK"
8636: (8)                    img = Image.frombytes(mode, (self.width, self.height), self.samples)
8637: (8)                    if "dpi" not in kwargs.keys():
8638: (12)                       kwargs["dpi"] = (self.xres, self.yres)
8639: (8)                    img.save(*args, **kwargs)
8640: (4)                def pil_tobytes(self, *args, **kwargs):
8641: (8)                    """Convert to binary image stream using pillow.
8642: (8)                    Args are passed to Pillow's Image.save method, see their
documentation.
8643: (8)                    Use instead of 'tobytes' when other output formats are needed.
8644: (8)                    """
8645: (8)                    from io import BytesIO
8646: (8)                    bytes_out = BytesIO()
8647: (8)                    self.pil_save(bytes_out, *args, **kwargs)
8648: (8)                    return bytes_out.getvalue()
8649: (4)                def pixel(self, x, y):
```

```
8650: (8)                    """Get color tuple of pixel (x, y).
8651: (8)                    Last item is the alpha if Pixmap.alpha is true."""
8652: (8)                    if g_use_extra:
8653: (12)                       return extra.pixmap_pixel(self.this.m_internal, x, y)
8654: (8)                    if (0
8655: (16)                           or x < 0
8656: (16)                           or x >= self.this.m_internal.w
8657: (16)                           or y < 0
8658: (16)                           or y >= self.this.m_internal.h
8659: (16)                           ):
8660: (12)                       RAISEPY(MSG_PIXEL_OUTSIDE, PyExc_ValueError)
8661: (8)                    n = self.this.m_internal.n
8662: (8)                    stride = self.this.m_internal.stride
8663: (8)                    i = stride * y + n * x
8664: (8)                    ret = tuple( self.samples_mv[ i: i+n])
8665: (8)                    return ret
8666: (4)                @property
8667: (4)                def samples(self)->bytes:
8668: (8)                    mv = self.samples_mv
8669: (8)                    return bytes( mv)
8670: (4)                @property
8671: (4)                def samples_mv(self):
8672: (8)                    '''
8673: (8)                    Pixmap samples memoryview.
8674: (8)                    '''
8675: (8)                    return mupdf.fz_pixmap_samples_memoryview(self.this)
8676: (4)                @property
8677: (4)                def samples_ptr(self):
8678: (8)                    return mupdf.fz_pixmap_samples_int(self.this)
8679: (4)                def save(self, filename, output=None, jpg_quality=95):
8680: (8)                    """Output as image in format determined by filename extension.
8681: (8)                    Args:
8682: (12)                       output: (str) only use to overrule filename extension. Default is
PNG.
8683: (20)                                  Others are JPEG, JPG, PNM, PGM, PPM, PBM, PAM, PSD, PS.
8684: (8)                    """
8685: (8)                    valid_formats = {
8686: (16)                           "png": 1,
8687: (16)                           "pnm": 2,
8688: (16)                           "pgm": 2,
8689: (16)                           "ppm": 2,
8690: (16)                           "pbm": 2,
8691: (16)                           "pam": 3,
8692: (16)                           "psd": 5,
8693: (16)                           "ps": 6,
8694: (16)                           "jpg": 7,
8695: (16)                           "jpeg": 7,
8696: (16)                           }
8697: (8)                    if type(filename) is str:
8698: (12)                       pass
8699: (8)                    elif hasattr(filename, "absolute"):
8700: (12)                       filename = str(filename)
8701: (8)                    elif hasattr(filename, "name"):
8702: (12)                       filename = filename.name
8703: (8)                    if output is None:
8704: (12)                       _, ext = os.path.splitext(filename)
8705: (12)                       output = ext[1:]
8706: (8)                    idx = valid_formats.get(output.lower(), None)
8707: (8)                    if idx is None:
8708: (12)                       raise ValueError(f"Image format {output} not in
{tuple(valid_formats.keys())}")
8709: (8)                    if self.alpha and idx in (2, 6, 7):
8710: (12)                       raise ValueError("'%s' cannot have alpha" % output)
8711: (8)                    if self.colorspace and self.colorspace.n > 3 and idx in (1, 2, 4):
8712: (12)                       raise ValueError(f"unsupported colorspace for '{output}'")
8713: (8)                    if idx == 7:
8714: (12)                       self.set_dpi(self.xres, self.yres)
8715: (8)                    return self._writeIMG(filename, idx, jpg_quality)
8716: (4)                def set_alpha(self, alphavalues=None, premultiply=1, opaque=None,
```

```
matte=None):
8717: (8)                    """Set alpha channel to values contained in a byte array.
8718: (8)                    If omitted, set alphas to 255.
8719: (8)                    Args:
8720: (12)                       alphavalues: (bytes) with length (width * height) or 'None'.
8721: (12)                       premultiply: (bool, True) premultiply colors with alpha values.
8722: (12)                       opaque: (tuple, length colorspace.n) this color receives opacity
0.
8723: (12)                       matte: (tuple, length colorspace.n)) preblending background color.
8724: (8)                    """
8725: (8)                    pix = self.this
8726: (8)                    alpha = 0
8727: (8)                    m = 0
8728: (8)                    if pix.alpha() == 0:
8729: (12)                       raise ValueError( MSG_PIX_NOALPHA)
8730: (8)                    n = mupdf.fz_pixmap_colorants(pix)
8731: (8)                    w = mupdf.fz_pixmap_width(pix)
8732: (8)                    h = mupdf.fz_pixmap_height(pix)
8733: (8)                    balen = w * h * (n+1)
8734: (8)                    colors = [0, 0, 0, 0]   # make this color opaque
8735: (8)                    bgcolor = [0, 0, 0, 0]  # preblending background color
8736: (8)                    zero_out = 0
8737: (8)                    bground = 0
8738: (8)                    if opaque and isinstance(opaque, (list, tuple)) and len(opaque) == n:
8739: (12)                       for i in range(n):
8740: (16)                           colors[i] = opaque[i]
8741: (12)                       zero_out = 1
8742: (8)                    if matte and isinstance( matte, (tuple, list)) and len(matte) == n:
8743: (12)                       for i in range(n):
8744: (16)                           bgcolor[i] = matte[i]
8745: (12)                       bground = 1
8746: (8)                    data = bytes()
8747: (8)                    data_len = 0
8748: (8)                    if alphavalues:
8749: (12)                       if isinstance(alphavalues, (bytes, bytearray)):
8750: (16)                           data = alphavalues
8751: (16)                           data_len = len(alphavalues)
8752: (12)                       else:
8753: (16)                           assert 0, f'unexpected type for alphavalues:
{type(alphavalues)}'
8754: (12)                       if data_len < w * h:
8755: (16)                           raise ValueError( "bad alpha values")
8756: (8)                    if 1:
8757: (12)                       mupdf.Pixmap_set_alpha_helper(
8758: (20)                               balen,
8759: (20)                               n,
8760: (20)                               data_len,
8761: (20)                               zero_out,
8762: (20)                               mupdf.python_buffer_data( data),
8763: (20)                               pix.m_internal,
8764: (20)                               premultiply,
8765: (20)                               bground,
8766: (20)                               colors,
8767: (20)                               bgcolor,
8768: (20)                               )
8769: (8)                    else:
8770: (12)                       i = k = j = 0
8771: (12)                       data_fix = 255
8772: (12)                       while i < balen:
8773: (16)                           alpha = data[k]
8774: (16)                           if zero_out:
8775: (20)                               for j in range(i, i+n):
8776: (24)                                   if mupdf.fz_samples_get(pix, j) != colors[j - i]:
8777: (28)                                       data_fix = 255
8778: (28)                                       break
8779: (24)                                   else:
8780: (28)                                       data_fix = 0
8781: (16)                           if data_len:
8782: (20)                               def fz_mul255( a, b):
```

```
8783: (24)                                  x = a * b + 128
8784: (24)                                  x += x // 256
8785: (24)                                  return x // 256
8786: (20)                              if data_fix == 0:
8787: (24)                                  mupdf.fz_samples_set(pix, i+n, 0)
8788: (20)                              else:
8789: (24)                                  mupdf.fz_samples_set(pix, i+n, alpha)
8790: (20)                              if premultiply and not bground:
8791: (24)                                  for j in range(i, i+n):
8792: (28)                                      mupdf.fz_samples_set(pix, j, fz_mul255(
mupdf.fz_samples_get(pix, j), alpha))
8793: (20)                              elif bground:
8794: (24)                                  for j in range( i, i+n):
8795: (28)                                      m = bgcolor[j - i]
8796: (28)                                      mupdf.fz_samples_set(pix, j, fz_mul255(
mupdf.fz_samples_get(pix, j) - m, alpha))
8797: (16)                          else:
8798: (20)                              mupdf.fz_samples_set(pix, i+n, data_fix)
8799: (16)                          i += n+1
8800: (16)                          k += 1
8801: (4)               def tobytes(self, output="png", jpg_quality=95):
8802: (8)                   '''
8803: (8)                   Convert to binary image stream of desired type.
8804: (8)                   '''
8805: (8)                   valid_formats = {
8806: (16)                          "png": 1,
8807: (16)                          "pnm": 2,
8808: (16)                          "pgm": 2,
8809: (16)                          "ppm": 2,
8810: (16)                          "pbm": 2,
8811: (16)                          "pam": 3,
8812: (16)                          "tga": 4,
8813: (16)                          "tpic": 4,
8814: (16)                          "psd": 5,
8815: (16)                          "ps": 6,
8816: (16)                          'jpg': 7,
8817: (16)                          'jpeg': 7,
8818: (16)                          }
8819: (8)                   idx = valid_formats.get(output.lower(), None)
8820: (8)                   if idx is None:
8821: (12)                      raise ValueError(f"Image format {output} not in
{tuple(valid_formats.keys())}")
8822: (8)                   if self.alpha and idx in (2, 6, 7):
8823: (12)                      raise ValueError("'{output}' cannot have alpha")
8824: (8)                   if self.colorspace and self.colorspace.n > 3 and idx in (1, 2, 4):
8825: (12)                      raise ValueError(f"unsupported colorspace for '{output}'")
8826: (8)                   if idx == 7:
8827: (12)                      self.set_dpi(self.xres, self.yres)
8828: (8)                   barray = self._tobytes(idx, jpg_quality)
8829: (8)                   return barray
8830: (4)               def set_dpi(self, xres, yres):
8831: (8)                   """Set resolution in both dimensions."""
8832: (8)                   pm = self.this
8833: (8)                   pm.m_internal.xres = xres
8834: (8)                   pm.m_internal.yres = yres
8835: (4)               def set_origin(self, x, y):
8836: (8)                   """Set top-left coordinates."""
8837: (8)                   pm = self.this
8838: (8)                   pm.m_internal.x = x
8839: (8)                   pm.m_internal.y = y
8840: (4)               def set_pixel(self, x, y, color):
8841: (8)                   """Set color of pixel (x, y)."""
8842: (8)                   if g_use_extra:
8843: (12)                      return extra.set_pixel(self.this.m_internal, x, y, color)
8844: (8)                   pm = self.this
8845: (8)                   if not _INRANGE(x, 0, pm.w() - 1) or not _INRANGE(y, 0, pm.h() - 1):
8846: (12)                      raise ValueError( MSG_PIXEL_OUTSIDE)
8847: (8)                   n = pm.n()
8848: (8)                   for j in range(n):
```

```
8849: (12)                      i = color[j]
8850: (12)                      if not _INRANGE(i, 0, 255):
8851: (16)                          raise ValueError( MSG_BAD_COLOR_SEQ)
8852: (8)                   stride = mupdf.fz_pixmap_stride( pm)
8853: (8)                   i = stride * y + n * x
8854: (8)                   if 0:
8855: (12)                      if not self._memory_view:
8856: (16)                          self._memory_view = self.samples_mv
8857: (12)                      for j in range(n):
8858: (16)                          self._memory_view[i + j] = color[j]
8859: (8)                   else:
8860: (12)                      for j in range(n):
8861: (16)                          pm.fz_samples_set(i + j, color[j])
8862: (4)               def set_rect(self, bbox, color):
8863: (8)                   """Set color of all pixels in bbox."""
8864: (8)                   pm = self.this
8865: (8)                   n = pm.n()
8866: (8)                   c = []
8867: (8)                   for j in range(n):
8868: (12)                      i = color[j]
8869: (12)                      if not _INRANGE(i, 0, 255):
8870: (16)                          raise ValueError( MSG_BAD_COLOR_SEQ)
8871: (12)                      c.append(i)
8872: (8)                   bbox = JM_irect_from_py(bbox)
8873: (8)                   i = JM_fill_pixmap_rect_with_color(pm, c, bbox)
8874: (8)                   rc = bool(i)
8875: (8)                   return rc
8876: (4)               def shrink(self, factor):
8877: (8)                   """Divide width and height by 2**factor.
8878: (8)                   E.g. factor=1 shrinks to 25% of original size (in place)."""
8879: (8)                   if factor < 1:
8880: (12)                      message_warning("ignoring shrink factor < 1")
8881: (12)                      return
8882: (8)                   mupdf.fz_subsample_pixmap( self.this, factor)
8883: (8)                   self._memory_view = None
8884: (4)               @property
8885: (4)               def size(self):
8886: (8)                   """Pixmap size."""
8887: (8)                   if mupdf_version_tuple >= (1, 23, 8):
8888: (12)                      return  mupdf.fz_pixmap_size( self.this)
8889: (8)                   pm = self.this
8890: (8)                   return pm.n() * pm.w() * pm.h()
8891: (4)               @property
8892: (4)               def stride(self):
8893: (8)                   """Length of one image line (width * n)."""
8894: (8)                   return self.this.stride()
8895: (4)               def tint_with(self, black, white):
8896: (8)                   """Tint colors with modifiers for black and white."""
8897: (8)                   if not self.colorspace or self.colorspace.n > 3:
8898: (12)                      message("warning: colorspace invalid for function")
8899: (12)                      return
8900: (8)                   return mupdf.fz_tint_pixmap( self.this, black, white)
8901: (4)               @property
8902: (4)               def w(self):
8903: (8)                   """The width."""
8904: (8)                   return mupdf.fz_pixmap_width(self.this)
8905: (4)               def warp(self, quad, width, height):
8906: (8)                   """Return pixmap from a warped quad."""
8907: (8)                   if not quad.is_convex: raise ValueError("quad must be convex")
8908: (8)                   q = JM_quad_from_py(quad)
8909: (8)                   points = [ q.ul, q.ur, q.lr, q.ll]
8910: (8)                   dst = mupdf.fz_warp_pixmap( self.this, points, width, height)
8911: (8)                   return Pixmap( dst)
8912: (4)               @property
8913: (4)               def x(self):
8914: (8)                   """x component of Pixmap origin."""
8915: (8)                   return mupdf.fz_pixmap_x(self.this)
8916: (4)               @property
8917: (4)               def xres(self):
```

```
8918: (8)                    """Resolution in x direction."""
8919: (8)                    return self.this.xres()
8920: (4)                @property
8921: (4)                def y(self):
8922: (8)                    """y component of Pixmap origin."""
8923: (8)                    return mupdf.fz_pixmap_y(self.this)
8924: (4)                @property
8925: (4)                def yres(self):
8926: (8)                    """Resolution in y direction."""
8927: (8)                    return self.this.yres()
8928: (4)                width  = w
8929: (4)                height = h
8930: (0)            del Point
8931: (0)            class Point:
8932: (4)                def __abs__(self):
8933: (8)                    return math.sqrt(self.x * self.x + self.y * self.y)
8934: (4)                def __add__(self, p):
8935: (8)                    if hasattr(p, "__float__"):
8936: (12)                       return Point(self.x + p, self.y + p)
8937: (8)                    if len(p) != 2:
8938: (12)                       raise ValueError("Point: bad seq len")
8939: (8)                    return Point(self.x + p[0], self.y + p[1])
8940: (4)                def __bool__(self):
8941: (8)                    return not (max(self) == min(self) == 0)
8942: (4)                def __eq__(self, p):
8943: (8)                    if not hasattr(p, "__len__"):
8944: (12)                       return False
8945: (8)                    return len(p) == 2 and bool(self - p) is False
8946: (4)                def __getitem__(self, i):
8947: (8)                    return (self.x, self.y)[i]
8948: (4)                def __hash__(self):
8949: (8)                    return hash(tuple(self))
8950: (4)                def __init__(self, *args, x=None, y=None):
8951: (8)                    '''
8952: (8)                    Point() - all zeros
8953: (8)                    Point(x, y)
8954: (8)                    Point(Point) - new copy
8955: (8)                    Point(sequence) - from 'sequence'
8956: (8)                    Explicit keyword args x, y override earlier settings if not None.
8957: (8)                    '''
8958: (8)                    if not args:
8959: (12)                       self.x = 0.0
8960: (12)                       self.y = 0.0
8961: (8)                    elif len(args) > 2:
8962: (12)                       raise ValueError("Point: bad seq len")
8963: (8)                    elif len(args) == 2:
8964: (12)                       self.x = float(args[0])
8965: (12)                       self.y = float(args[1])
8966: (8)                    elif len(args) == 1:
8967: (12)                       l = args[0]
8968: (12)                       if isinstance(l, (mupdf.FzPoint, mupdf.fz_point)):
8969: (16)                           self.x = l.x
8970: (16)                           self.y = l.y
8971: (12)                       else:
8972: (16)                           if hasattr(l, "__getitem__") is False:
8973: (20)                               raise ValueError("Point: bad args")
8974: (16)                           if len(l) != 2:
8975: (20)                               raise ValueError("Point: bad seq len")
8976: (16)                           self.x = float(l[0])
8977: (16)                           self.y = float(l[1])
8978: (8)                    else:
8979: (12)                       raise ValueError("Point: bad seq len")
8980: (8)                    if x is not None:   self.x = x
8981: (8)                    if y is not None:   self.y = y
8982: (4)                def __len__(self):
8983: (8)                    return 2
8984: (4)                def __mul__(self, m):
8985: (8)                    if hasattr(m, "__float__"):
8986: (12)                       return Point(self.x * m, self.y * m)
```

```
8987: (8)                   p = Point(self)
8988: (8)                   return p.transform(m)
8989: (4)               def __neg__(self):
8990: (8)                   return Point(-self.x, -self.y)
8991: (4)               def __nonzero__(self):
8992: (8)                   return not (max(self) == min(self) == 0)
8993: (4)               def __pos__(self):
8994: (8)                   return Point(self)
8995: (4)               def __repr__(self):
8996: (8)                   return "Point" + str(tuple(self))
8997: (4)               def __setitem__(self, i, v):
8998: (8)                   v = float(v)
8999: (8)                   if   i == 0: self.x = v
9000: (8)                   elif i == 1: self.y = v
9001: (8)                   else:
9002: (12)                      raise IndexError("index out of range")
9003: (8)                   return None
9004: (4)               def __sub__(self, p):
9005: (8)                   if hasattr(p, "__float__"):
9006: (12)                      return Point(self.x - p, self.y - p)
9007: (8)                   if len(p) != 2:
9008: (12)                      raise ValueError("Point: bad seq len")
9009: (8)                   return Point(self.x - p[0], self.y - p[1])
9010: (4)               def __truediv__(self, m):
9011: (8)                   if hasattr(m, "__float__"):
9012: (12)                      return Point(self.x * 1./m, self.y * 1./m)
9013: (8)                   m1 = util_invert_matrix(m)[1]
9014: (8)                   if not m1:
9015: (12)                      raise ZeroDivisionError("matrix not invertible")
9016: (8)                   p = Point(self)
9017: (8)                   return p.transform(m1)
9018: (4)               @property
9019: (4)               def abs_unit(self):
9020: (8)                   """Unit vector with positive coordinates."""
9021: (8)                   s = self.x * self.x + self.y * self.y
9022: (8)                   if s < EPSILON:
9023: (12)                      return Point(0,0)
9024: (8)                   s = math.sqrt(s)
9025: (8)                   return Point(abs(self.x) / s, abs(self.y) / s)
9026: (4)               def distance_to(self, *args):
9027: (8)                   """Return distance to rectangle or another point."""
9028: (8)                   if not len(args) > 0:
9029: (12)                      raise ValueError("at least one parameter must be given")
9030: (8)                   x = args[0]
9031: (8)                   if len(x) == 2:
9032: (12)                      x = Point(x)
9033: (8)                   elif len(x) == 4:
9034: (12)                      x = Rect(x)
9035: (8)                   else:
9036: (12)                      raise ValueError("arg1 must be point-like or rect-like")
9037: (8)                   if len(args) > 1:
9038: (12)                      unit = args[1]
9039: (8)                   else:
9040: (12)                      unit = "px"
9041: (8)                   u = {"px": (1.,1.), "in": (1.,72.), "cm": (2.54, 72.),
9042: (13)                        "mm": (25.4, 72.)}
9043: (8)                   f = u[unit][0] / u[unit][1]
9044: (8)                   if type(x) is Point:
9045: (12)                      return abs(self - x) * f
9046: (8)                   r = Rect(x.top_left, x.top_left)
9047: (8)                   r = r | x.bottom_right
9048: (8)                   if self in r:
9049: (12)                      return 0.0
9050: (8)                   if self.x > r.x1:
9051: (12)                      if self.y >= r.y1:
9052: (16)                          return self.distance_to(r.bottom_right, unit)
9053: (12)                      elif self.y <= r.y0:
9054: (16)                          return self.distance_to(r.top_right, unit)
9055: (12)                      else:
```

```
9056: (16)                          return (self.x - r.x1) * f
9057: (8)                   elif r.x0 <= self.x <= r.x1:
9058: (12)                      if self.y >= r.y1:
9059: (16)                          return (self.y - r.y1) * f
9060: (12)                      else:
9061: (16)                          return (r.y0 - self.y) * f
9062: (8)                   else:
9063: (12)                      if self.y >= r.y1:
9064: (16)                          return self.distance_to(r.bottom_left, unit)
9065: (12)                      elif self.y <= r.y0:
9066: (16)                          return self.distance_to(r.top_left, unit)
9067: (12)                      else:
9068: (16)                          return (r.x0 - self.x) * f
9069: (4)               def transform(self, m):
9070: (8)                   """Replace point by its transformation with matrix-like m."""
9071: (8)                   if len(m) != 6:
9072: (12)                      raise ValueError("Matrix: bad seq len")
9073: (8)                   self.x, self.y = util_transform_point(self, m)
9074: (8)                   return self
9075: (4)               @property
9076: (4)               def unit(self):
9077: (8)                   """Unit vector of the point."""
9078: (8)                   s = self.x * self.x + self.y * self.y
9079: (8)                   if s < EPSILON:
9080: (12)                      return Point(0,0)
9081: (8)                   s = math.sqrt(s)
9082: (8)                   return Point(self.x / s, self.y / s)
9083: (4)               __div__ = __truediv__
9084: (4)               norm = __abs__
9085: (0)           class Quad:
9086: (4)               def __abs__(self):
9087: (8)                   if self.is_empty:
9088: (12)                      return 0.0
9089: (8)                   return abs(self.ul - self.ur) * abs(self.ul - self.ll)
9090: (4)               def __add__(self, q):
9091: (8)                   if hasattr(q, "__float__"):
9092: (12)                      return Quad(self.ul + q, self.ur + q, self.ll + q, self.lr + q)
9093: (8)                   if len(q) != 4:
9094: (12)                      raise ValueError("Quad: bad seq len")
9095: (8)                   return Quad(self.ul + q[0], self.ur + q[1], self.ll + q[2], self.lr +
q[3])
9096: (4)               def __bool__(self):
9097: (8)                   return not self.is_empty
9098: (4)               def __contains__(self, x):
9099: (8)                   try:
9100: (12)                      l = x.__len__()
9101: (8)                   except Exception:
9102: (12)                      if g_exceptions_verbose > 1:    exception_info()
9103: (12)                      return False
9104: (8)                   if l == 2:
9105: (12)                      return util_point_in_quad(x, self)
9106: (8)                   if l != 4:
9107: (12)                      return False
9108: (8)                   if CheckRect(x):
9109: (12)                      if Rect(x).is_empty:
9110: (16)                          return True
9111: (12)                      return util_point_in_quad(x[:2], self) and
util_point_in_quad(x[2:], self)
9112: (8)                   if CheckQuad(x):
9113: (12)                      for i in range(4):
9114: (16)                          if not util_point_in_quad(x[i], self):
9115: (20)                              return False
9116: (12)                      return True
9117: (8)                   return False
9118: (4)               def __eq__(self, quad):
9119: (8)                   if not hasattr(quad, "__len__"):
9120: (12)                      return False
9121: (8)                   return len(quad) == 4 and (
9122: (12)                      self.ul == quad[0] and
```

```
9123: (12)                      self.ur == quad[1] and
9124: (12)                      self.ll == quad[2] and
9125: (12)                      self.lr == quad[3]
9126: (8)                   )
9127: (4)               def __getitem__(self, i):
9128: (8)                   return (self.ul, self.ur, self.ll, self.lr)[i]
9129: (4)               def __hash__(self):
9130: (8)                   return hash(tuple(self))
9131: (4)               def __init__(self, *args, ul=None, ur=None, ll=None, lr=None):
9132: (8)                   '''
9133: (8)                   Quad() - all zero points
9134: (8)                   Quad(ul, ur, ll, lr)
9135: (8)                   Quad(quad) - new copy
9136: (8)                   Quad(sequence) - from 'sequence'
9137: (8)                   Explicit keyword args ul, ur, ll, lr override earlier settings if not
9138: (8)                   None.
9139: (8)                   '''
9140: (8)                   if not args:
9141: (12)                      self.ul = self.ur = self.ll = self.lr = Point()
9142: (8)                   elif len(args) > 4:
9143: (12)                      raise ValueError("Quad: bad seq len")
9144: (8)                   elif len(args) == 4:
9145: (12)                      self.ul, self.ur, self.ll, self.lr = map(Point, args)
9146: (8)                   elif len(args) == 1:
9147: (12)                      l = args[0]
9148: (12)                      if isinstance(l, mupdf.FzQuad):
9149: (16)                          self.this = l
9150: (16)                          self.ul, self.ur, self.ll, self.lr = Point(l.ul), Point(l.ur),
Point(l.ll), Point(l.lr)
9151: (12)                      elif hasattr(l, "__getitem__") is False:
9152: (16)                          raise ValueError("Quad: bad args")
9153: (12)                      elif len(l) != 4:
9154: (16)                          raise ValueError("Quad: bad seq len")
9155: (12)                      else:
9156: (16)                          self.ul, self.ur, self.ll, self.lr = map(Point, l)
9157: (8)                   else:
9158: (12)                      raise ValueError("Quad: bad args")
9159: (8)                   if ul is not None:  self.ul = Point(ul)
9160: (8)                   if ur is not None:  self.ur = Point(ur)
9161: (8)                   if ll is not None:  self.ll = Point(ll)
9162: (8)                   if lr is not None:  self.lr = Point(lr)
9163: (4)               def __len__(self):
9164: (8)                   return 4
9165: (4)               def __mul__(self, m):
9166: (8)                   q = Quad(self)
9167: (8)                   q = q.transform(m)
9168: (8)                   return q
9169: (4)               def __neg__(self):
9170: (8)                   return Quad(-self.ul, -self.ur, -self.ll, -self.lr)
9171: (4)               def __nonzero__(self):
9172: (8)                   return not self.is_empty
9173: (4)               def __pos__(self):
9174: (8)                   return Quad(self)
9175: (4)               def __repr__(self):
9176: (8)                   return "Quad" + str(tuple(self))
9177: (4)               def __setitem__(self, i, v):
9178: (8)                   if   i == 0: self.ul = Point(v)
9179: (8)                   elif i == 1: self.ur = Point(v)
9180: (8)                   elif i == 2: self.ll = Point(v)
9181: (8)                   elif i == 3: self.lr = Point(v)
9182: (8)                   else:
9183: (12)                      raise IndexError("index out of range")
9184: (8)                   return None
9185: (4)               def __sub__(self, q):
9186: (8)                   if hasattr(q, "__float__"):
9187: (12)                      return Quad(self.ul - q, self.ur - q, self.ll - q, self.lr - q)
9188: (8)                   if len(q) != 4:
9189: (12)                      raise ValueError("Quad: bad seq len")
9190: (8)                   return Quad(self.ul - q[0], self.ur - q[1], self.ll - q[2], self.lr -
```

```
q[3])
    def __truediv__(self, m):
        if hasattr(m, "__float__"):
            im = 1. / m
        else:
            im = util_invert_matrix(m)[1]
            if not im:
                raise ZeroDivisionError("Matrix not invertible")
        q = Quad(self)
        q = q.transform(im)
        return q
    @property
    def is_convex(self):
        """Check if quad is convex and not degenerate.
        Notes:
            Check that for the two diagonals, the other two corners are not
            on the same side of the diagonal.
        Returns:
            True or False.
        """
        m = planish_line(self.ul, self.lr)  # puts this diagonal on x-axis
        p1 = self.ll * m  # transform the
        p2 = self.ur * m  # other two points
        if p1.y * p2.y > 0:
            return False
        m = planish_line(self.ll, self.ur)  # puts other diagonal on x-axis
        p1 = self.lr * m  # transform the
        p2 = self.ul * m  # remaining points
        if p1.y * p2.y > 0:
            return False
        return True
    @property
    def is_empty(self):
        """Check whether all quad corners are on the same line.
        This is the case if width or height is zero.
        """
        return self.width < EPSILON or self.height < EPSILON
    @property
    def is_infinite(self):
        """Check whether this is the infinite quad."""
        return self.rect.is_infinite
    @property
    def is_rectangular(self):
        """Check if quad is rectangular.
        Notes:
            Some rotation matrix can thus transform it into a rectangle.
            This is equivalent to three corners enclose 90 degrees.
        Returns:
            True or False.
        """
        sine = util_sine_between(self.ul, self.ur, self.lr)
        if abs(sine - 1) > EPSILON:  # the sine of the angle
            return False
        sine = util_sine_between(self.ur, self.lr, self.ll)
        if abs(sine - 1) > EPSILON:
            return False
        sine = util_sine_between(self.lr, self.ll, self.ul)
        if abs(sine - 1) > EPSILON:
            return False
        return True
    def morph(self, p, m):
        """Morph the quad with matrix-like 'm' and point-like 'p'.
        Return a new quad."""
        if self.is_infinite:
            return INFINITE_QUAD()
        delta = Matrix(1, 1).pretranslate(p.x, p.y)
        q = self * ~delta * m * delta
        return q
    @property
```

```
9259: (4)                def rect(self):
9260: (8)                    r = Rect()
9261: (8)                    r.x0 = min(self.ul.x, self.ur.x, self.lr.x, self.ll.x)
9262: (8)                    r.y0 = min(self.ul.y, self.ur.y, self.lr.y, self.ll.y)
9263: (8)                    r.x1 = max(self.ul.x, self.ur.x, self.lr.x, self.ll.x)
9264: (8)                    r.y1 = max(self.ul.y, self.ur.y, self.lr.y, self.ll.y)
9265: (8)                    return r
9266: (4)                def transform(self, m):
9267: (8)                    """Replace quad by its transformation with matrix m."""
9268: (8)                    if hasattr(m, "__float__"):
9269: (12)                       pass
9270: (8)                    elif len(m) != 6:
9271: (12)                       raise ValueError("Matrix: bad seq len")
9272: (8)                    self.ul *= m
9273: (8)                    self.ur *= m
9274: (8)                    self.ll *= m
9275: (8)                    self.lr *= m
9276: (8)                    return self
9277: (4)                __div__ = __truediv__
9278: (4)                width  = property(lambda self: max(abs(self.ul - self.ur), abs(self.ll -
self.lr)))
9279: (4)                height = property(lambda self: max(abs(self.ul - self.ll), abs(self.ur -
self.lr)))
9280: (0)            class Rect:
9281: (4)                def __abs__(self):
9282: (8)                    if self.is_empty or self.is_infinite:
9283: (12)                       return 0.0
9284: (8)                    return (self.x1 - self.x0) * (self.y1 - self.y0)
9285: (4)                def __add__(self, p):
9286: (8)                    if hasattr(p, "__float__"):
9287: (12)                       return Rect(self.x0 + p, self.y0 + p, self.x1 + p, self.y1 + p)
9288: (8)                    if len(p) != 4:
9289: (12)                       raise ValueError("Rect: bad seq len")
9290: (8)                    return Rect(self.x0 + p[0], self.y0 + p[1], self.x1 + p[2], self.y1 +
p[3])
9291: (4)                def __and__(self, x):
9292: (8)                    if not hasattr(x, "__len__"):
9293: (12)                       raise ValueError("bad operand 2")
9294: (8)                    r1 = Rect(x)
9295: (8)                    r = Rect(self)
9296: (8)                    return r.intersect(r1)
9297: (4)                def __bool__(self):
9298: (8)                    return not (max(self) == min(self) == 0)
9299: (4)                def __contains__(self, x):
9300: (8)                    if hasattr(x, "__float__"):
9301: (12)                       return x in tuple(self)
9302: (8)                    l = len(x)
9303: (8)                    if l == 2:
9304: (12)                       return util_is_point_in_rect(x, self)
9305: (8)                    if l == 4:
9306: (12)                       r = INFINITE_RECT()
9307: (12)                       try:
9308: (16)                           r = Rect(x)
9309: (12)                       except Exception:
9310: (16)                           if g_exceptions_verbose > 1:    exception_info()
9311: (16)                           r = Quad(x).rect
9312: (12)                       return (self.x0 <= r.x0 <= r.x1 <= self.x1 and
9313: (20)                               self.y0 <= r.y0 <= r.y1 <= self.y1)
9314: (8)                    return False
9315: (4)                def __eq__(self, rect):
9316: (8)                    if not hasattr(rect, "__len__"):
9317: (12)                       return False
9318: (8)                    return len(rect) == 4 and bool(self - rect) is False
9319: (4)                def __getitem__(self, i):
9320: (8)                    return (self.x0, self.y0, self.x1, self.y1)[i]
9321: (4)                def __hash__(self):
9322: (8)                    return hash(tuple(self))
9323: (4)                def __init__(self, *args, p0=None, p1=None, x0=None, y0=None, x1=None,
y1=None):
```

```
 9324: (8)                    """
 9325: (8)                    Rect() - all zeros
 9326: (8)                    Rect(x0, y0, x1, y1)
 9327: (8)                    Rect(top-left, x1, y1)
 9328: (8)                    Rect(x0, y0, bottom-right)
 9329: (8)                    Rect(top-left, bottom-right)
 9330: (8)                    Rect(Rect or IRect) - new copy
 9331: (8)                    Rect(sequence) - from 'sequence'
 9332: (8)                    Explicit keyword args p0, p1, x0, y0, x1, y1 override earlier settings
 9333: (8)                    if not None.
 9334: (8)                    """
 9335: (8)                    x0, y0, x1, y1 = util_make_rect( *args, p0=p0, p1=p1, x0=x0, y0=y0,
 x1=x1, y1=y1)
 9336: (8)                    self.x0 = float( x0)
 9337: (8)                    self.y0 = float( y0)
 9338: (8)                    self.x1 = float( x1)
 9339: (8)                    self.y1 = float( y1)
 9340: (4)                def __len__(self):
 9341: (8)                    return 4
 9342: (4)                def __mul__(self, m):
 9343: (8)                    if hasattr(m, "__float__"):
 9344: (12)                       return Rect(self.x0 * m, self.y0 * m, self.x1 * m, self.y1 * m)
 9345: (8)                    r = Rect(self)
 9346: (8)                    r = r.transform(m)
 9347: (8)                    return r
 9348: (4)                def __neg__(self):
 9349: (8)                    return Rect(-self.x0, -self.y0, -self.x1, -self.y1)
 9350: (4)                def __nonzero__(self):
 9351: (8)                    return not (max(self) == min(self) == 0)
 9352: (4)                def __or__(self, x):
 9353: (8)                    if not hasattr(x, "__len__"):
 9354: (12)                       raise ValueError("bad operand 2")
 9355: (8)                    r = Rect(self)
 9356: (8)                    if len(x) == 2:
 9357: (12)                       return r.include_point(x)
 9358: (8)                    if len(x) == 4:
 9359: (12)                       return r.include_rect(x)
 9360: (8)                    raise ValueError("bad operand 2")
 9361: (4)                def __pos__(self):
 9362: (8)                    return Rect(self)
 9363: (4)                def __repr__(self):
 9364: (8)                    return "Rect" + str(tuple(self))
 9365: (4)                def __setitem__(self, i, v):
 9366: (8)                    v = float(v)
 9367: (8)                    if   i == 0: self.x0 = v
 9368: (8)                    elif i == 1: self.y0 = v
 9369: (8)                    elif i == 2: self.x1 = v
 9370: (8)                    elif i == 3: self.y1 = v
 9371: (8)                    else:
 9372: (12)                       raise IndexError("index out of range")
 9373: (8)                    return None
 9374: (4)                def __sub__(self, p):
 9375: (8)                    if hasattr(p, "__float__"):
 9376: (12)                       return Rect(self.x0 - p, self.y0 - p, self.x1 - p, self.y1 - p)
 9377: (8)                    if len(p) != 4:
 9378: (12)                       raise ValueError("Rect: bad seq len")
 9379: (8)                    return Rect(self.x0 - p[0], self.y0 - p[1], self.x1 - p[2], self.y1 -
 p[3])
 9380: (4)                def __truediv__(self, m):
 9381: (8)                    if hasattr(m, "__float__"):
 9382: (12)                       return Rect(self.x0 * 1./m, self.y0 * 1./m, self.x1 * 1./m,
 self.y1 * 1./m)
 9383: (8)                    im = util_invert_matrix(m)[1]
 9384: (8)                    if not im:
 9385: (12)                       raise ZeroDivisionError(f"Matrix not invertible: {m}")
 9386: (8)                    r = Rect(self)
 9387: (8)                    r = r.transform(im)
 9388: (8)                    return r
 9389: (4)                @property
```

```
9390: (4)                def bottom_left(self):
9391: (8)                    """Bottom-left corner."""
9392: (8)                    return Point(self.x0, self.y1)
9393: (4)                @property
9394: (4)                def bottom_right(self):
9395: (8)                    """Bottom-right corner."""
9396: (8)                    return Point(self.x1, self.y1)
9397: (4)                def contains(self, x):
9398: (8)                    """Check if containing point-like or rect-like x."""
9399: (8)                    return self.__contains__(x)
9400: (4)                @property
9401: (4)                def height(self):
9402: (8)                    return max(0, self.y1 - self.y0)
9403: (4)                def include_point(self, p):
9404: (8)                    """Extend to include point-like p."""
9405: (8)                    if len(p) != 2:
9406: (12)                       raise ValueError("Point: bad seq len")
9407: (8)                    self.x0, self.y0, self.x1, self.y1 = util_include_point_in_rect(self,
p)
9408: (8)                    return self
9409: (4)                def include_rect(self, r):
9410: (8)                    """Extend to include rect-like r."""
9411: (8)                    if len(r) != 4:
9412: (12)                       raise ValueError("Rect: bad seq len")
9413: (8)                    r = Rect(r)
9414: (8)                    if r.is_infinite or self.is_infinite:
9415: (12)                       self.x0, self.y0, self.x1, self.y1 = FZ_MIN_INF_RECT,
FZ_MIN_INF_RECT, FZ_MAX_INF_RECT, FZ_MAX_INF_RECT
9416: (8)                    elif r.is_empty:
9417: (12)                       return self
9418: (8)                    elif self.is_empty:
9419: (12)                       self.x0, self.y0, self.x1, self.y1 = r.x0, r.y0, r.x1, r.y1
9420: (8)                    else:
9421: (12)                       self.x0, self.y0, self.x1, self.y1 = util_union_rect(self, r)
9422: (8)                    return self
9423: (4)                def intersect(self, r):
9424: (8)                    """Restrict to common rect with rect-like r."""
9425: (8)                    if not len(r) == 4:
9426: (12)                       raise ValueError("Rect: bad seq len")
9427: (8)                    r = Rect(r)
9428: (8)                    if r.is_infinite:
9429: (12)                       return self
9430: (8)                    elif self.is_infinite:
9431: (12)                       self.x0, self.y0, self.x1, self.y1 = r.x0, r.y0, r.x1, r.y1
9432: (8)                    elif r.is_empty:
9433: (12)                       self.x0, self.y0, self.x1, self.y1 = r.x0, r.y0, r.x1, r.y1
9434: (8)                    elif self.is_empty:
9435: (12)                       return self
9436: (8)                    else:
9437: (12)                       self.x0, self.y0, self.x1, self.y1 = util_intersect_rect(self, r)
9438: (8)                    return self
9439: (4)                def intersects(self, x):
9440: (8)                    """Check if intersection with rectangle x is not empty."""
9441: (8)                    r1 = Rect(x)
9442: (8)                    if self.is_empty or self.is_infinite or r1.is_empty or r1.is_infinite:
9443: (12)                       return False
9444: (8)                    r = Rect(self)
9445: (8)                    if r.intersect(r1).is_empty:
9446: (12)                       return False
9447: (8)                    return True
9448: (4)                @property
9449: (4)                def is_empty(self):
9450: (8)                    """True if rectangle area is empty."""
9451: (8)                    return self.x0 >= self.x1 or self.y0 >= self.y1
9452: (4)                @property
9453: (4)                def is_infinite(self):
9454: (8)                    """True if this is the infinite rectangle."""
9455: (8)                    return self.x0 == self.y0 == FZ_MIN_INF_RECT and self.x1 == self.y1 ==
FZ_MAX_INF_RECT
```

```
9456: (4)                @property
9457: (4)                def is_valid(self):
9458: (8)                    """True if rectangle is valid."""
9459: (8)                    return self.x0 <= self.x1 and self.y0 <= self.y1
9460: (4)                def morph(self, p, m):
9461: (8)                    """Morph with matrix-like m and point-like p.
9462: (8)                    Returns a new quad."""
9463: (8)                    if self.is_infinite:
9464: (12)                       return INFINITE_QUAD()
9465: (8)                    return self.quad.morph(p, m)
9466: (4)                def norm(self):
9467: (8)                    return math.sqrt(sum([c*c for c in self]))
9468: (4)                def normalize(self):
9469: (8)                    """Replace rectangle with its finite version."""
9470: (8)                    if self.x1 < self.x0:
9471: (12)                       self.x0, self.x1 = self.x1, self.x0
9472: (8)                    if self.y1 < self.y0:
9473: (12)                       self.y0, self.y1 = self.y1, self.y0
9474: (8)                    return self
9475: (4)                @property
9476: (4)                def quad(self):
9477: (8)                    """Return Quad version of rectangle."""
9478: (8)                    return Quad(self.tl, self.tr, self.bl, self.br)
9479: (4)                def round(self):
9480: (8)                    """Return the IRect."""
9481: (8)                    return IRect(util_round_rect(self))
9482: (4)                @property
9483: (4)                def top_left(self):
9484: (8)                    """Top-left corner."""
9485: (8)                    return Point(self.x0, self.y0)
9486: (4)                @property
9487: (4)                def top_right(self):
9488: (8)                    """Top-right corner."""
9489: (8)                    return Point(self.x1, self.y0)
9490: (4)                def torect(self, r):
9491: (8)                    """Return matrix that converts to target rect."""
9492: (8)                    r = Rect(r)
9493: (8)                    if self.is_infinite or self.is_empty or r.is_infinite or r.is_empty:
9494: (12)                       raise ValueError("rectangles must be finite and not empty")
9495: (8)                    return (
9496: (12)                       Matrix(1, 0, 0, 1, -self.x0, -self.y0)
9497: (12)                       * Matrix(r.width / self.width, r.height / self.height)
9498: (12)                       * Matrix(1, 0, 0, 1, r.x0, r.y0)
9499: (8)                    )
9500: (4)                def transform(self, m):
9501: (8)                    """Replace with the transformation by matrix-like m."""
9502: (8)                    if not len(m) == 6:
9503: (12)                       raise ValueError("Matrix: bad seq len")
9504: (8)                    self.x0, self.y0, self.x1, self.y1 = util_transform_rect(self, m)
9505: (8)                    return self
9506: (4)                @property
9507: (4)                def width(self):
9508: (8)                    return max(0, self.x1 - self.x0)
9509: (4)                __div__ = __truediv__
9510: (4)                bl = bottom_left
9511: (4)                br = bottom_right
9512: (4)                irect = property(round)
9513: (4)                tl = top_left
9514: (4)                tr = top_right
9515: (0)            class Shape:
9516: (4)                """Create a new shape."""
9517: (4)                def __init__(self, page: Page):
9518: (8)                    CheckParent(page)
9519: (8)                    self.page = page
9520: (8)                    self.doc = page.parent
9521: (8)                    if not self.doc.is_pdf:
9522: (12)                       raise ValueError("not a PDF")
9523: (8)                    self.height = page.mediabox_size.y
9524: (8)                    self.width = page.mediabox_size.x
```

```
(8)                 self.x = page.cropbox_position.x
(8)                 self.y = page.cropbox_position.y
(8)                 self.pctm = page.transformation_matrix  # page transf. matrix
(8)                 self.ipctm = ~self.pctm  # inverted transf. matrix
(8)                 self.draw_cont = ""
(8)                 self.text_cont = ""
(8)                 self.totalcont = ""
(8)                 self.last_point = None
(8)                 self.rect = None
(4)             def commit(self, overlay: bool = True) -> None:
(8)                 """
(8)                 Update the page's /Contents object with Shape data. The argument
(8)                 controls whether data appear in foreground (default) or background.
(8)                 """
(8)                 CheckParent(self.page)  # doc may have died meanwhile
(8)                 self.totalcont += self.text_cont
(8)                 self.totalcont = self.totalcont.encode()
(8)                 if self.totalcont != b"":
(12)                    xref = TOOLS._insert_contents(self.page, b" ", overlay)
(12)                    mupdf.pdf_update_stream(self.doc, xref, self.totalcont)
(8)                 self.last_point = None  # clean up ...
(8)                 self.rect = None  #
(8)                 self.draw_cont = ""  # for potential ...
(8)                 self.text_cont = ""  # ...
(8)                 self.totalcont = ""  # re-use
(8)                 return
(4)             def draw_bezier(
(12)                    self,
(12)                    p1: point_like,
(12)                    p2: point_like,
(12)                    p3: point_like,
(12)                    p4: point_like,
(12)                    ):# -> Point:
(8)                 """Draw a standard cubic Bezier curve."""
(8)                 p1 = Point(p1)
(8)                 p2 = Point(p2)
(8)                 p3 = Point(p3)
(8)                 p4 = Point(p4)
(8)                 if not (self.last_point == p1):
(12)                    args = JM_TUPLE(p1 * self.ipctm)
(12)                    self.draw_cont += f"{_format_g(args)} m\n"
(8)                 args = JM_TUPLE(list(p2 * self.ipctm) + list(p3 * self.ipctm) +
list(p4 * self.ipctm))
(8)                 self.draw_cont += f"{_format_g(args)} c\n"
(8)                 self.updateRect(p1)
(8)                 self.updateRect(p2)
(8)                 self.updateRect(p3)
(8)                 self.updateRect(p4)
(8)                 self.last_point = p4
(8)                 return self.last_point
(4)             def draw_circle(self, center: point_like, radius: float):# -> Point:
(8)                 """Draw a circle given its center and radius."""
(8)                 if not radius > EPSILON:
(12)                    raise ValueError("radius must be positive")
(8)                 center = Point(center)
(8)                 p1 = center - (radius, 0)
(8)                 return self.draw_sector(center, p1, 360, fullSector=False)
(4)             def draw_curve(
(12)                    self,
(12)                    p1: point_like,
(12)                    p2: point_like,
(12)                    p3: point_like,
(12)                    ):# -> Point:
(8)                 """Draw a curve between points using one control point."""
(8)                 kappa = 0.55228474983
(8)                 p1 = Point(p1)
(8)                 p2 = Point(p2)
(8)                 p3 = Point(p3)
(8)                 k1 = p1 + (p2 - p1) * kappa
```

```
9593: (8)                    k2 = p3 + (p2 - p3) * kappa
9594: (8)                    return self.draw_bezier(p1, k1, k2, p3)
9595: (4)                def draw_line(self, p1: point_like, p2: point_like):# -> Point:
9596: (8)                    """Draw a line between two points."""
9597: (8)                    p1 = Point(p1)
9598: (8)                    p2 = Point(p2)
9599: (8)                    if not (self.last_point == p1):
9600: (12)                       self.draw_cont += _format_g(JM_TUPLE(p1 * self.ipctm)) + " m\n"
9601: (12)                       self.last_point = p1
9602: (12)                       self.updateRect(p1)
9603: (8)                    self.draw_cont += _format_g(JM_TUPLE(p2 * self.ipctm)) + " l\n"
9604: (8)                    self.updateRect(p2)
9605: (8)                    self.last_point = p2
9606: (8)                    return self.last_point
9607: (4)                def draw_oval(self, tetra: typing.Union[quad_like, rect_like]):# -> Point:
9608: (8)                    """Draw an ellipse inside a tetrapod."""
9609: (8)                    if len(tetra) != 4:
9610: (12)                       raise ValueError("invalid arg length")
9611: (8)                    if hasattr(tetra[0], "__float__"):
9612: (12)                       q = Rect(tetra).quad
9613: (8)                    else:
9614: (12)                       q = Quad(tetra)
9615: (8)                    mt = q.ul + (q.ur - q.ul) * 0.5
9616: (8)                    mr = q.ur + (q.lr - q.ur) * 0.5
9617: (8)                    mb = q.ll + (q.lr - q.ll) * 0.5
9618: (8)                    ml = q.ul + (q.ll - q.ul) * 0.5
9619: (8)                    if not (self.last_point == ml):
9620: (12)                       self.draw_cont += _format_g(JM_TUPLE(ml * self.ipctm)) + " m\n"
9621: (12)                       self.last_point = ml
9622: (8)                    self.draw_curve(ml, q.ll, mb)
9623: (8)                    self.draw_curve(mb, q.lr, mr)
9624: (8)                    self.draw_curve(mr, q.ur, mt)
9625: (8)                    self.draw_curve(mt, q.ul, ml)
9626: (8)                    self.updateRect(q.rect)
9627: (8)                    self.last_point = ml
9628: (8)                    return self.last_point
9629: (4)                def draw_polyline(self, points: list):# -> Point:
9630: (8)                    """Draw several connected line segments."""
9631: (8)                    for i, p in enumerate(points):
9632: (12)                       if i == 0:
9633: (16)                           if not (self.last_point == Point(p)):
9634: (20)                               self.draw_cont += _format_g(JM_TUPLE(Point(p) *
self.ipctm)) + " m\n"
9635: (20)                               self.last_point = Point(p)
9636: (12)                       else:
9637: (16)                           self.draw_cont += _format_g(JM_TUPLE(Point(p) * self.ipctm)) +
" l\n"
9638: (12)                       self.updateRect(p)
9639: (8)                    self.last_point = Point(points[-1])
9640: (8)                    return self.last_point
9641: (4)                def draw_quad(self, quad: quad_like):# -> Point:
9642: (8)                    """Draw a Quad."""
9643: (8)                    q = Quad(quad)
9644: (8)                    return self.draw_polyline([q.ul, q.ll, q.lr, q.ur, q.ul])
9645: (4)                def draw_rect(self, rect: rect_like):# -> Point:
9646: (8)                    """Draw a rectangle."""
9647: (8)                    r = Rect(rect)
9648: (8)                    args = JM_TUPLE(list(r.bl * self.ipctm) + [r.width, r.height])
9649: (8)                    self.draw_cont += _format_g(args) + " re\n"
9650: (8)                    self.updateRect(r)
9651: (8)                    self.last_point = r.tl
9652: (8)                    return self.last_point
9653: (4)                def draw_sector(
9654: (12)                       self,
9655: (12)                       center: point_like,
9656: (12)                       point: point_like,
9657: (12)                       beta: float,
9658: (12)                       fullSector: bool = True,
9659: (12)                       ):# -> Point:
```

```
9660: (8)                    """Draw a circle sector."""
9661: (8)                    center = Point(center)
9662: (8)                    point = Point(point)
9663: (8)                    def l3(a, b):
9664: (12)                       return _format_g((a, b)) + " m\n"
9665: (8)                    def l4(a, b, c, d, e, f):
9666: (12)                       return _format_g((a, b, c, d, e, f)) + " c\n"
9667: (8)                    def l5(a, b):
9668: (12)                       return _format_g((a, b)) + " l\n"
9669: (8)                    betar = math.radians(-beta)
9670: (8)                    w360 = math.radians(math.copysign(360, betar)) * (-1)
9671: (8)                    w90 = math.radians(math.copysign(90, betar))
9672: (8)                    w45 = w90 / 2
9673: (8)                    while abs(betar) > 2 * math.pi:
9674: (12)                       betar += w360  # bring angle below 360 degrees
9675: (8)                    if not (self.last_point == point):
9676: (12)                       self.draw_cont += l3(JM_TUPLE(point * self.ipctm))
9677: (12)                       self.last_point = point
9678: (8)                    Q = Point(0, 0)  # just make sure it exists
9679: (8)                    C = center
9680: (8)                    P = point
9681: (8)                    S = P - C  # vector 'center' -> 'point'
9682: (8)                    rad = abs(S)  # circle radius
9683: (8)                    if not rad > EPSILON:
9684: (12)                       raise ValueError("radius must be positive")
9685: (8)                    alfa = self.horizontal_angle(center, point)
9686: (8)                    while abs(betar) > abs(w90):  # draw 90 degree arcs
9687: (12)                       q1 = C.x + math.cos(alfa + w90) * rad
9688: (12)                       q2 = C.y + math.sin(alfa + w90) * rad
9689: (12)                       Q = Point(q1, q2)  # the arc's end point
9690: (12)                       r1 = C.x + math.cos(alfa + w45) * rad / math.cos(w45)
9691: (12)                       r2 = C.y + math.sin(alfa + w45) * rad / math.cos(w45)
9692: (12)                       R = Point(r1, r2)  # crossing point of tangents
9693: (12)                       kappah = (1 - math.cos(w45)) * 4 / 3 / abs(R - Q)
9694: (12)                       kappa = kappah * abs(P - Q)
9695: (12)                       cp1 = P + (R - P) * kappa  # control point 1
9696: (12)                       cp2 = Q + (R - Q) * kappa  # control point 2
9697: (12)                       self.draw_cont += l4(JM_TUPLE(
9698: (16)                           list(cp1 * self.ipctm) + list(cp2 * self.ipctm) + list(Q *
self.ipctm)
9699: (12)                       ))
9700: (12)                       betar -= w90  # reduce param angle by 90 deg
9701: (12)                       alfa += w90  # advance start angle by 90 deg
9702: (12)                       P = Q  # advance to arc end point
9703: (8)                    if abs(betar) > 1e-3:  # significant degrees left?
9704: (12)                       beta2 = betar / 2
9705: (12)                       q1 = C.x + math.cos(alfa + betar) * rad
9706: (12)                       q2 = C.y + math.sin(alfa + betar) * rad
9707: (12)                       Q = Point(q1, q2)  # the arc's end point
9708: (12)                       r1 = C.x + math.cos(alfa + beta2) * rad / math.cos(beta2)
9709: (12)                       r2 = C.y + math.sin(alfa + beta2) * rad / math.cos(beta2)
9710: (12)                       R = Point(r1, r2)  # crossing point of tangents
9711: (12)                       kappah = (1 - math.cos(beta2)) * 4 / 3 / abs(R - Q)  # kappa
height
9712: (12)                       kappa = kappah * abs(P - Q) / (1 - math.cos(betar))
9713: (12)                       cp1 = P + (R - P) * kappa  # control point 1
9714: (12)                       cp2 = Q + (R - Q) * kappa  # control point 2
9715: (12)                       self.draw_cont += l4(JM_TUPLE(
9716: (16)                           list(cp1 * self.ipctm) + list(cp2 * self.ipctm) + list(Q *
self.ipctm)
9717: (12)                       ))
9718: (8)                    if fullSector:
9719: (12)                       self.draw_cont += l3(JM_TUPLE(point * self.ipctm))
9720: (12)                       self.draw_cont += l5(JM_TUPLE(center * self.ipctm))
9721: (12)                       self.draw_cont += l5(JM_TUPLE(Q * self.ipctm))
9722: (8)                    self.last_point = Q
9723: (8)                    return self.last_point
9724: (4)                def draw_squiggle(
9725: (12)                       self,
```

```
9726: (12)                      p1: point_like,
9727: (12)                      p2: point_like,
9728: (12)                      breadth=2,
9729: (12)                      ):# -> Point:
9730: (8)                   """Draw a squiggly line from p1 to p2."""
9731: (8)                   p1 = Point(p1)
9732: (8)                   p2 = Point(p2)
9733: (8)                   S = p2 - p1  # vector start - end
9734: (8)                   rad = abs(S)  # distance of points
9735: (8)                   cnt = 4 * int(round(rad / (4 * breadth), 0))  # always take full
phases
9736: (8)                   if cnt < 4:
9737: (12)                      raise ValueError("points too close")
9738: (8)                   mb = rad / cnt  # revised breadth
9739: (8)                   matrix = Matrix(TOOLS._hor_matrix(p1, p2))  # normalize line to x-axis
9740: (8)                   i_mat = ~matrix  # get original position
9741: (8)                   k = 2.4142135623765633  # y of draw_curve helper point
9742: (8)                   points = []  # stores edges
9743: (8)                   for i in range(1, cnt):
9744: (12)                      if i % 4 == 1:  # point "above" connection
9745: (16)                          p = Point(i, -k) * mb
9746: (12)                      elif i % 4 == 3:  # point "below" connection
9747: (16)                          p = Point(i, k) * mb
9748: (12)                      else:  # else on connection line
9749: (16)                          p = Point(i, 0) * mb
9750: (12)                      points.append(p * i_mat)
9751: (8)                   points = [p1] + points + [p2]
9752: (8)                   cnt = len(points)
9753: (8)                   i = 0
9754: (8)                   while i + 2 < cnt:
9755: (12)                      self.draw_curve(points[i], points[i + 1], points[i + 2])
9756: (12)                      i += 2
9757: (8)                   return p2
9758: (4)               def draw_zigzag(
9759: (12)                      self,
9760: (12)                      p1: point_like,
9761: (12)                      p2: point_like,
9762: (12)                      breadth: float = 2,
9763: (12)                      ):# -> Point:
9764: (8)                   """Draw a zig-zagged line from p1 to p2."""
9765: (8)                   p1 = Point(p1)
9766: (8)                   p2 = Point(p2)
9767: (8)                   S = p2 - p1  # vector start - end
9768: (8)                   rad = abs(S)  # distance of points
9769: (8)                   cnt = 4 * int(round(rad / (4 * breadth), 0))  # always take full
phases
9770: (8)                   if cnt < 4:
9771: (12)                      raise ValueError("points too close")
9772: (8)                   mb = rad / cnt  # revised breadth
9773: (8)                   matrix = Matrix(TOOLS._hor_matrix(p1, p2))  # normalize line to x-axis
9774: (8)                   i_mat = ~matrix  # get original position
9775: (8)                   points = []  # stores edges
9776: (8)                   for i in range(1, cnt):
9777: (12)                      if i % 4 == 1:  # point "above" connection
9778: (16)                          p = Point(i, -1) * mb
9779: (12)                      elif i % 4 == 3:  # point "below" connection
9780: (16)                          p = Point(i, 1) * mb
9781: (12)                      else:  # ignore others
9782: (16)                          continue
9783: (12)                      points.append(p * i_mat)
9784: (8)                   self.draw_polyline([p1] + points + [p2])  # add start and end points
9785: (8)                   return p2
9786: (4)               def finish(
9787: (12)                      self,
9788: (12)                      width: float = 1,
9789: (12)                      color: OptSeq = (0,),
9790: (12)                      fill: OptSeq = None,
9791: (12)                      lineCap: int = 0,
9792: (12)                      lineJoin: int = 0,
```

```
9793: (12)                      dashes: OptStr = None,
9794: (12)                      even_odd: bool = False,
9795: (12)                      morph: OptSeq = None,
9796: (12)                      closePath: bool = True,
9797: (12)                      fill_opacity: float = 1,
9798: (12)                      stroke_opacity: float = 1,
9799: (12)                      oc: int = 0,
9800: (12)                      ) -> None:
9801: (8)                   """Finish the current drawing segment.
9802: (8)                   Notes:
9803: (12)                      Apply colors, opacity, dashes, line style and width, or
9804: (12)                      morphing. Also whether to close the path
9805: (12)                      by connecting last to first point.
9806: (8)                   """
9807: (8)                   if self.draw_cont == "":  # treat empty contents as no-op
9808: (12)                      return
9809: (8)                   if width == 0:  # border color makes no sense then
9810: (12)                      color = None
9811: (8)                   elif color is None:  # vice versa
9812: (12)                      width = 0
9813: (8)                   color_str = ColorCode(color, "c")  # ensure proper color string
9814: (8)                   fill_str = ColorCode(fill, "f")  # ensure proper fill string
9815: (8)                   optcont = self.page._get_optional_content(oc)
9816: (8)                   if optcont is not None:
9817: (12)                      self.draw_cont = "/OC /%s BDC\n" % optcont + self.draw_cont
9818: (12)                      emc = "EMC\n"
9819: (8)                   else:
9820: (12)                      emc = ""
9821: (8)                   alpha = self.page._set_opacity(CA=stroke_opacity, ca=fill_opacity)
9822: (8)                   if alpha is not None:
9823: (12)                      self.draw_cont = "/%s gs\n" % alpha + self.draw_cont
9824: (8)                   if width != 1 and width != 0:
9825: (12)                      self.draw_cont += _format_g(width) + " w\n"
9826: (8)                   if lineCap != 0:
9827: (12)                      self.draw_cont = "%i J\n" % lineCap + self.draw_cont
9828: (8)                   if lineJoin != 0:
9829: (12)                      self.draw_cont = "%i j\n" % lineJoin + self.draw_cont
9830: (8)                   if dashes not in (None, "", "[] 0"):
9831: (12)                      self.draw_cont = "%s d\n" % dashes + self.draw_cont
9832: (8)                   if closePath:
9833: (12)                      self.draw_cont += "h\n"
9834: (12)                      self.last_point = None
9835: (8)                   if color is not None:
9836: (12)                      self.draw_cont += color_str
9837: (8)                   if fill is not None:
9838: (12)                      self.draw_cont += fill_str
9839: (12)                      if color is not None:
9840: (16)                          if not even_odd:
9841: (20)                              self.draw_cont += "B\n"
9842: (16)                          else:
9843: (20)                              self.draw_cont += "B*\n"
9844: (12)                      else:
9845: (16)                          if not even_odd:
9846: (20)                              self.draw_cont += "f\n"
9847: (16)                          else:
9848: (20)                              self.draw_cont += "f*\n"
9849: (8)                   else:
9850: (12)                      self.draw_cont += "S\n"
9851: (8)                   self.draw_cont += emc
9852: (8)                   if CheckMorph(morph):
9853: (12)                      m1 = Matrix(
9854: (16)                          1, 0, 0, 1, morph[0].x + self.x, self.height - morph[0].y -
self.y
9855: (12)                      )
9856: (12)                      mat = ~m1 * morph[1] * m1
9857: (12)                      self.draw_cont = _format_g(JM_TUPLE(mat) + self.draw_cont) + "
cm\n"
9858: (8)                   self.totalcont += "\nq\n" + self.draw_cont + "Q\n"
9859: (8)                   self.draw_cont = ""
```

```
9860: (8)                   self.last_point = None
9861: (8)                   return
9862: (4)               @staticmethod
9863: (4)               def horizontal_angle(C, P):
9864: (8)                   """Return the angle to the horizontal for the connection from C to P.
9865: (8)                   This uses the arcus sine function and resolves its inherent ambiguity
by
9866: (8)                   looking up in which quadrant vector S = P - C is located.
9867: (8)                   """
9868: (8)                   S = Point(P - C).unit  # unit vector 'C' -> 'P'
9869: (8)                   alfa = math.asin(abs(S.y))  # absolute angle from horizontal
9870: (8)                   if S.x < 0:  # make arcsin result unique
9871: (12)                      if S.y <= 0:  # bottom-left
9872: (16)                          alfa = -(math.pi - alfa)
9873: (12)                      else:  # top-left
9874: (16)                          alfa = math.pi - alfa
9875: (8)                   else:
9876: (12)                      if S.y >= 0:  # top-right
9877: (16)                          pass
9878: (12)                      else:  # bottom-right
9879: (16)                          alfa = -alfa
9880: (8)                   return alfa
9881: (4)               def insert_text(
9882: (12)                      self,
9883: (12)                      point: point_like,
9884: (12)                      buffer_: typing.Union[str, list],
9885: (12)                      fontsize: float = 11,
9886: (12)                      lineheight: OptFloat = None,
9887: (12)                      fontname: str = "helv",
9888: (12)                      fontfile: OptStr = None,
9889: (12)                      set_simple: bool = 0,
9890: (12)                      encoding: int = 0,
9891: (12)                      color: OptSeq = None,
9892: (12)                      fill: OptSeq = None,
9893: (12)                      render_mode: int = 0,
9894: (12)                      border_width: float = 1,
9895: (12)                      rotate: int = 0,
9896: (12)                      morph: OptSeq = None,
9897: (12)                      stroke_opacity: float = 1,
9898: (12)                      fill_opacity: float = 1,
9899: (12)                      oc: int = 0,
9900: (12)                      ) -> int:
9901: (8)                   if not bool(buffer_):
9902: (12)                      return 0
9903: (8)                   if type(buffer_) not in (list, tuple):
9904: (12)                      text = buffer_.splitlines()
9905: (8)                   else:
9906: (12)                      text = buffer_
9907: (8)                   if not len(text) > 0:
9908: (12)                      return 0
9909: (8)                   point = Point(point)
9910: (8)                   try:
9911: (12)                      maxcode = max([ord(c) for c in " ".join(text)])
9912: (8)                   except Exception:
9913: (12)                      exception_info()
9914: (12)                      return 0
9915: (8)                   fname = fontname
9916: (8)                   if fname.startswith("/"):
9917: (12)                      fname = fname[1:]
9918: (8)                   xref = self.page.insert_font(
9919: (16)                          fontname=fname,
9920: (16)                          fontfile=fontfile,
9921: (16)                          encoding=encoding,
9922: (16)                          set_simple=set_simple,
9923: (16)                          )
9924: (8)                   fontinfo = CheckFontInfo(self.doc, xref)
9925: (8)                   fontdict = fontinfo[1]
9926: (8)                   ordering = fontdict["ordering"]
9927: (8)                   simple = fontdict["simple"]
```

```
9928: (8)                    bfname = fontdict["name"]
9929: (8)                    ascender = fontdict["ascender"]
9930: (8)                    descender = fontdict["descender"]
9931: (8)                    if lineheight:
9932: (12)                       lheight = fontsize * lineheight
9933: (8)                    elif ascender - descender <= 1:
9934: (12)                       lheight = fontsize * 1.2
9935: (8)                    else:
9936: (12)                       lheight = fontsize * (ascender - descender)
9937: (8)                    if maxcode > 255:
9938: (12)                       glyphs = self.doc.get_char_widths(xref, maxcode + 1)
9939: (8)                    else:
9940: (12)                       glyphs = fontdict["glyphs"]
9941: (8)                    tab = []
9942: (8)                    for t in text:
9943: (12)                       if simple and bfname not in ("Symbol", "ZapfDingbats"):
9944: (16)                           g = None
9945: (12)                       else:
9946: (16)                           g = glyphs
9947: (12)                       tab.append(getTJstr(t, g, simple, ordering))
9948: (8)                    text = tab
9949: (8)                    color_str = ColorCode(color, "c")
9950: (8)                    fill_str = ColorCode(fill, "f")
9951: (8)                    if not fill and render_mode == 0:  # ensure fill color when 0 Tr
9952: (12)                       fill = color
9953: (12)                       fill_str = ColorCode(color, "f")
9954: (8)                    morphing = CheckMorph(morph)
9955: (8)                    rot = rotate
9956: (8)                    if rot % 90 != 0:
9957: (12)                       raise ValueError("bad rotate value")
9958: (8)                    while rot < 0:
9959: (12)                       rot += 360
9960: (8)                    rot = rot % 360  # text rotate = 0, 90, 270, 180
9961: (8)                    templ1 = lambda a, b, c, d, e, f, g: f"\nq\n{a}{b}BT\n%{c}1 0 0 1
{_format_g((d, e))} Tm\n/{f} {g} Tf "
9962: (8)                    templ2 = lambda a: f"TJ\n0 -{_format_g(a)} TD\n"
9963: (8)                    cmp90 = "0 1 -1 0 0 0 cm\n"  # rotates 90 deg counter-clockwise
9964: (8)                    cmm90 = "0 -1 1 0 0 0 cm\n"  # rotates 90 deg clockwise
9965: (8)                    cm180 = "-1 0 0 -1 0 0 cm\n"  # rotates by 180 deg.
9966: (8)                    height = self.height
9967: (8)                    width = self.width
9968: (8)                    if morphing:
9969: (12)                       m1 = Matrix(1, 0, 0, 1, morph[0].x + self.x, height - morph[0].y -
self.y)
9970: (12)                       mat = ~m1 * morph[1] * m1
9971: (12)                       cm = _format_g(JM_TUPLE(mat)) + " cm\n"
9972: (8)                    else:
9973: (12)                       cm = ""
9974: (8)                    top = height - point.y - self.y  # start of 1st char
9975: (8)                    left = point.x + self.x  # start of 1. char
9976: (8)                    space = top  # space available
9977: (8)                    if rot == 90:
9978: (12)                       left = height - point.y - self.y
9979: (12)                       top = -point.x - self.x
9980: (12)                       cm += cmp90
9981: (12)                       space = width - abs(top)
9982: (8)                    elif rot == 270:
9983: (12)                       left = -height + point.y + self.y
9984: (12)                       top = point.x + self.x
9985: (12)                       cm += cmm90
9986: (12)                       space = abs(top)
9987: (8)                    elif rot == 180:
9988: (12)                       left = -point.x - self.x
9989: (12)                       top = -height + point.y + self.y
9990: (12)                       cm += cm180
9991: (12)                       space = abs(point.y + self.y)
9992: (8)                    optcont = self.page._get_optional_content(oc)
9993: (8)                    if optcont is not None:
9994: (12)                       bdc = "/OC /%s BDC\n" % optcont
```

```
9995: (12)                      emc = "EMC\n"
9996: (8)                   else:
9997: (12)                      bdc = emc = ""
9998: (8)                   alpha = self.page._set_opacity(CA=stroke_opacity, ca=fill_opacity)
9999: (8)                   if alpha is None:
10000: (12)                     alpha = ""
10001: (8)                  else:
10002: (12)                     alpha = "/%s gs\n" % alpha
10003: (8)                  nres = templ1(bdc, alpha, cm, left, top, fname, fontsize)
10004: (8)                  if render_mode > 0:
10005: (12)                     nres += "%i Tr " % render_mode
10006: (8)                  if border_width != 1:
10007: (12)                     nres += _format_g(border_width) + " w "
10008: (8)                  if color is not None:
10009: (12)                     nres += color_str
10010: (8)                  if fill is not None:
10011: (12)                     nres += fill_str
10012: (8)                  nres += text[0]
10013: (8)                  nlines = 1  # set output line counter
10014: (8)                  if len(text) > 1:
10015: (12)                     nres += templ2(lheight)  # line 1
10016: (8)                  else:
10017: (12)                     nres += 'TJ'
10018: (8)                  for i in range(1, len(text)):
10019: (12)                     if space < lheight:
10020: (16)                         break  # no space left on page
10021: (12)                     if i > 1:
10022: (16)                         nres += "\nT* "
10023: (12)                     nres += text[i] + 'TJ'
10024: (12)                     space -= lheight
10025: (12)                     nlines += 1
10026: (8)                  nres += "\nET\n%sQ\n" % emc
10027: (8)                  self.text_cont += nres
10028: (8)                  return nlines
10029: (4)              def update_rect(self, x):
10030: (8)                  if self.rect is None:
10031: (12)                     if len(x) == 2:
10032: (16)                         self.rect = Rect(x, x)
10033: (12)                     else:
10034: (16)                         self.rect = Rect(x)
10035: (8)                  else:
10036: (12)                     if len(x) == 2:
10037: (16)                         x = Point(x)
10038: (16)                         self.rect.x0 = min(self.rect.x0, x.x)
10039: (16)                         self.rect.y0 = min(self.rect.y0, x.y)
10040: (16)                         self.rect.x1 = max(self.rect.x1, x.x)
10041: (16)                         self.rect.y1 = max(self.rect.y1, x.y)
10042: (12)                     else:
10043: (16)                         x = Rect(x)
10044: (16)                         self.rect.x0 = min(self.rect.x0, x.x0)
10045: (16)                         self.rect.y0 = min(self.rect.y0, x.y0)
10046: (16)                         self.rect.x1 = max(self.rect.x1, x.x1)
10047: (16)                         self.rect.y1 = max(self.rect.y1, x.y1)
10048: (0)          class Story:
10049: (4)              def __init__( self, html='', user_css=None, em=12, archive=None):
10050: (8)                  buffer_ = mupdf.fz_new_buffer_from_copied_data( html.encode('utf-8'))
10051: (8)                  if archive and not isinstance(archive, Archive):
10052: (12)                     archive = Archive(archive)
10053: (8)                  arch = archive.this if archive else mupdf.FzArchive( None)
10054: (8)                  if hasattr(mupdf, 'FzStoryS'):
10055: (12)                     self.this = mupdf.FzStoryS( buffer_, user_css, em, arch)
10056: (8)                  else:
10057: (12)                     self.this = mupdf.FzStory( buffer_, user_css, em, arch)
10058: (4)              def add_header_ids(self):
10059: (8)                  '''
10060: (8)                  Look for `<h1..6>` items in `self` and adds unique `id`
10061: (8)                  attributes if not already present.
10062: (8)                  '''
10063: (8)                  dom = self.body
```

```
10064: (8)                  i = 0
10065: (8)                  x = dom.find(None, None, None)
10066: (8)                  while x:
10067: (12)                     name = x.tagname
10068: (12)                     if len(name) == 2 and name[0]=="h" and name[1] in "123456":
10069: (16)                         attr = x.get_attribute_value("id")
10070: (16)                         if not attr:
10071: (20)                             id_ = f"h_id_{i}"
10072: (20)                             x.set_attribute("id", id_)
10073: (20)                             i += 1
10074: (12)                     x = x.find_next(None, None, None)
10075: (4)              @staticmethod
10076: (4)              def add_pdf_links(document_or_stream, positions):
10077: (8)                  """
10078: (8)                  Adds links to PDF document.
10079: (8)                  Args:
10080: (12)                     document_or_stream:
10081: (16)                         A PDF `Document` or raw PDF content, for example an
10082: (16)                         `io.BytesIO` instance.
10083: (12)                     positions:
10084: (16)                         List of `ElementPosition`'s for `document_or_stream`,
10085: (16)                         typically from Story.element_positions(). We raise an
10086: (16)                         exception if two or more positions have same id.
10087: (8)                  Returns:
10088: (12)                     `document_or_stream` if a `Document` instance, otherwise a
10089: (12)                     new `Document` instance.
10090: (8)                  We raise an exception if an `href` in `positions` refers to an
10091: (8)                  internal position `#<name>` but no item in `positions` has `id =
10092: (8)                  name`.
10093: (8)                  """
10094: (8)                  if isinstance(document_or_stream, Document):
10095: (12)                     document = document_or_stream
10096: (8)                  else:
10097: (12)                     document = Document("pdf", document_or_stream)
10098: (8)                  id_to_position = dict()
10099: (8)                  for position in positions:
10100: (12)                     if (position.open_close & 1) and position.id:
10101: (16)                         if position.id in id_to_position:
10102: (20)                             pass
10103: (16)                         else:
10104: (20)                             id_to_position[ position.id] = position
10105: (8)                  for position_from in positions:
10106: (12)                     if (position_from.open_close & 1) and position_from.href:
10107: (16)                         link = dict()
10108: (16)                         link['from'] = Rect(position_from.rect)
10109: (16)                         if position_from.href.startswith("#"):
10110: (20)                             target_id = position_from.href[1:]
10111: (20)                             try:
10112: (24)                                 position_to = id_to_position[ target_id]
10113: (20)                             except Exception as e:
10114: (24)                                 if g_exceptions_verbose > 1:    exception_info()
10115: (24)                                 raise RuntimeError(f"No destination with id=
{target_id}, required by position_from: {position_from}") from e
10116: (20)                             if 0:
10117: (24)                                 log(f"add_pdf_links(): making link from:")
10118: (24)                                 log(f"add_pdf_links():    {position_from}")
10119: (24)                                 log(f"add_pdf_links(): to:")
10120: (24)                                 log(f"add_pdf_links():    {position_to}")
10121: (20)                             link["kind"] = LINK_GOTO
10122: (20)                             x0, y0, x1, y1 = position_to.rect
10123: (20)                             link["to"] = Point(x0, y0)
10124: (20)                             link["page"] = position_to.page_num - 1
10125: (16)                         else:
10126: (20)                             if position_from.href.startswith('name:'):
10127: (24)                                 link['kind'] = LINK_NAMED
10128: (24)                                 link['name'] = position_from.href[5:]
10129: (20)                             else:
10130: (24)                                 link['kind'] = LINK_URI
10131: (24)                                 link['uri'] = position_from.href
```

```
10132: (16)                         document[position_from.page_num - 1].insert_link(link)
10133: (8)                  return document
10134: (4)              @property
10135: (4)              def body(self):
10136: (8)                  dom = self.document()
10137: (8)                  return dom.bodytag()
10138: (4)              def document( self):
10139: (8)                  dom = mupdf.fz_story_document( self.this)
10140: (8)                  return Xml( dom)
10141: (4)              def draw( self, device, matrix=None):
10142: (8)                  ctm2 = JM_matrix_from_py( matrix)
10143: (8)                  dev = device.this if device else mupdf.FzDevice( None)
10144: (8)                  mupdf.fz_draw_story( self.this, dev, ctm2)
10145: (4)              def element_positions( self, function, args=None):
10146: (8)                  '''
10147: (8)                  Trigger a callback function to record where items have been placed.
10148: (8)                  '''
10149: (8)                  if type(args) is dict:
10150: (12)                     for k in args.keys():
10151: (16)                         if not (type(k) is str and k.isidentifier()):
10152: (20)                             raise ValueError(f"invalid key '{k}'")
10153: (8)                  else:
10154: (12)                     args = {}
10155: (8)                  if not callable(function) or function.__code__.co_argcount != 1:
10156: (12)                     raise ValueError("callback 'function' must be a callable with
exactly one argument")
10157: (8)                  def function2( position):
10158: (12)                     class Position2:
10159: (16)                         pass
10160: (12)                     position2 = Position2()
10161: (12)                     position2.depth = position.depth
10162: (12)                     position2.heading = position.heading
10163: (12)                     position2.id = position.id
10164: (12)                     position2.rect = JM_py_from_rect(position.rect)
10165: (12)                     position2.text = position.text
10166: (12)                     position2.open_close = position.open_close
10167: (12)                     position2.rect_num = position.rectangle_num
10168: (12)                     position2.href = position.href
10169: (12)                     if args:
10170: (16)                         for k, v in args.items():
10171: (20)                             setattr( position2, k, v)
10172: (12)                     function( position2)
10173: (8)                  mupdf.fz_story_positions( self.this, function2)
10174: (4)              def place( self, where):
10175: (8)                  where = JM_rect_from_py( where)
10176: (8)                  filled = mupdf.FzRect()
10177: (8)                  more = mupdf.fz_place_story( self.this, where, filled)
10178: (8)                  return more, JM_py_from_rect( filled)
10179: (4)              def reset( self):
10180: (8)                  mupdf.fz_reset_story( self.this)
10181: (4)              def write(self, writer, rectfn, positionfn=None, pagefn=None):
10182: (8)                  dev = None
10183: (8)                  page_num = 0
10184: (8)                  rect_num = 0
10185: (8)                  filled = Rect(0, 0, 0, 0)
10186: (8)                  while 1:
10187: (12)                     mediabox, rect, ctm = rectfn(rect_num, filled)
10188: (12)                     rect_num += 1
10189: (12)                     if mediabox:
10190: (16)                         page_num += 1
10191: (12)                     more, filled = self.place( rect)
10192: (12)                     if positionfn:
10193: (16)                         def positionfn2(position):
10194: (20)                             position.page_num = page_num
10195: (20)                             positionfn(position)
10196: (16)                         self.element_positions(positionfn2)
10197: (12)                     if writer:
10198: (16)                         if mediabox:
10199: (20)                             if dev:
```

```
10200: (24)                                 if pagefn:
10201: (28)                                     pagefn(page_num, mediabox, dev, 1)
10202: (24)                                 writer.end_page()
10203: (20)                             dev = writer.begin_page( mediabox)
10204: (20)                             if pagefn:
10205: (24)                                 pagefn(page_num, mediabox, dev, 0)
10206: (16)                         self.draw( dev, ctm)
10207: (16)                         if not more:
10208: (20)                             if pagefn:
10209: (24)                                 pagefn( page_num, mediabox, dev, 1)
10210: (20)                             writer.end_page()
10211: (12)                     else:
10212: (16)                         self.draw(None, ctm)
10213: (12)                     if not more:
10214: (16)                         break
10215: (4)              @staticmethod
10216: (4)              def write_stabilized(writer, contentfn, rectfn, user_css=None, em=12,
positionfn=None, pagefn=None, archive=None, add_header_ids=True):
10217: (8)                  positions = list()
10218: (8)                  content = None
10219: (8)                  while 1:
10220: (12)                     content_prev = content
10221: (12)                     content = contentfn( positions)
10222: (12)                     stable = False
10223: (12)                     if content == content_prev:
10224: (16)                         stable = True
10225: (12)                     content2 = content
10226: (12)                     story = Story(content2, user_css, em, archive)
10227: (12)                     if add_header_ids:
10228: (16)                         story.add_header_ids()
10229: (12)                     positions = list()
10230: (12)                     def positionfn2(position):
10231: (16)                         positions.append(position)
10232: (16)                         if stable and positionfn:
10233: (20)                             positionfn(position)
10234: (12)                     story.write(
10235: (20)                             writer if stable else None,
10236: (20)                             rectfn,
10237: (20)                             positionfn2,
10238: (20)                             pagefn,
10239: (20)                             )
10240: (12)                     if stable:
10241: (16)                         break
10242: (4)              @staticmethod
10243: (4)              def write_stabilized_with_links(contentfn, rectfn, user_css=None, em=12,
positionfn=None, pagefn=None, archive=None, add_header_ids=True):
10244: (8)                  stream = io.BytesIO()
10245: (8)                  writer = DocumentWriter(stream)
10246: (8)                  positions = []
10247: (8)                  def positionfn2(position):
10248: (12)                     positions.append(position)
10249: (12)                     if positionfn:
10250: (16)                         positionfn(position)
10251: (8)                  Story.write_stabilized(writer, contentfn, rectfn, user_css, em,
positionfn2, pagefn, archive, add_header_ids)
10252: (8)                  writer.close()
10253: (8)                  stream.seek(0)
10254: (8)                  return Story.add_pdf_links(stream, positions)
10255: (4)              def write_with_links(self, rectfn, positionfn=None, pagefn=None):
10256: (8)                  stream = io.BytesIO()
10257: (8)                  writer = DocumentWriter(stream)
10258: (8)                  positions = []
10259: (8)                  def positionfn2(position):
10260: (12)                     positions.append(position)
10261: (12)                     if positionfn:
10262: (16)                         positionfn(position)
10263: (8)                  self.write(writer, rectfn, positionfn=positionfn2, pagefn=pagefn)
10264: (8)                  writer.close()
10265: (8)                  stream.seek(0)
```

```
10266: (8)                  return Story.add_pdf_links(stream, positions)
10267: (4)              class FitResult:
10268: (8)                  '''
10269: (8)                  The result from a `Story.fit*()` method.
10270: (8)                  Members:
10271: (8)                  `big_enough`:
10272: (12)                     `True` if the fit succeeded.
10273: (8)                  `filled`:
10274: (12)                     From the last call to `Story.place()`.
10275: (8)                  `more`:
10276: (12)                     `False` if the fit succeeded.
10277: (8)                  `numcalls`:
10278: (12)                     Number of calls made to `self.place()`.
10279: (8)                  `parameter`:
10280: (12)                     The successful parameter value, or the largest failing value.
10281: (8)                  `rect`:
10282: (12)                     The rect created from `parameter`.
10283: (8)                  '''
10284: (8)                  def __init__(self, big_enough=None, filled=None, more=None,
numcalls=None, parameter=None, rect=None):
10285: (12)                     self.big_enough = big_enough
10286: (12)                     self.filled = filled
10287: (12)                     self.more = more
10288: (12)                     self.numcalls = numcalls
10289: (12)                     self.parameter = parameter
10290: (12)                     self.rect = rect
10291: (8)                  def __repr__(self):
10292: (12)                     return (
10293: (20)                             f' big_enough={self.big_enough}'
10294: (20)                             f' filled={self.filled}'
10295: (20)                             f' more={self.more}'
10296: (20)                             f' numcalls={self.numcalls}'
10297: (20)                             f' parameter={self.parameter}'
10298: (20)                             f' rect={self.rect}'
10299: (20)                             )
10300: (4)              def fit(self, fn, pmin=None, pmax=None, delta=0.001, verbose=False):
10301: (8)                  '''
10302: (8)                  Finds optimal rect that contains the story `self`.
10303: (8)                  Returns a `Story.FitResult` instance.
10304: (8)                  On success, the last call to `self.place()` will have been with the
10305: (8)                  returned rectangle, so `self.draw()` can be used directly.
10306: (8)                  Args:
10307: (8)                  :arg fn:
10308: (12)                     A callable taking a floating point `parameter` and returning a
10309: (12)                     `pymupdf.Rect()`. If the rect is empty, we assume the story will
10310: (12)                     not fit and do not call `self.place()`.
10311: (12)                     Must guarantee that `self.place()` behaves monotonically when
10312: (12)                     given rect `fn(parameter`) as `parameter` increases. This
10313: (12)                     usually means that both width and height increase or stay
10314: (12)                     unchanged as `parameter` increases.
10315: (8)                  :arg pmin:
10316: (12)                     Minimum parameter to consider; `None` for -infinity.
10317: (8)                  :arg pmax:
10318: (12)                     Maximum parameter to consider; `None` for +infinity.
10319: (8)                  :arg delta:
10320: (12)                     Maximum error in returned `parameter`.
10321: (8)                  :arg verbose:
10322: (12)                     If true we output diagnostics.
10323: (8)                  '''
10324: (8)                  def log(text):
10325: (12)                     assert verbose
10326: (12)                     message(f'fit(): {text}')
10327: (8)                  assert isinstance(pmin, (int, float)) or pmin is None
10328: (8)                  assert isinstance(pmax, (int, float)) or pmax is None
10329: (8)                  class State:
10330: (12)                     def __init__(self):
10331: (16)                         self.pmin = pmin
10332: (16)                         self.pmax = pmax
10333: (16)                         self.pmin_result = None
```

```
10334: (16)                         self.pmax_result = None
10335: (16)                         self.result = None
10336: (16)                         self.numcalls = 0
10337: (16)                         if verbose:
10338: (20)                             self.pmin0 = pmin
10339: (20)                             self.pmax0 = pmax
10340: (8)                  state = State()
10341: (8)                  if verbose:
10342: (12)                     log(f'starting. {state.pmin=} {state.pmax=}.')
10343: (8)                  self.reset()
10344: (8)                  def ret():
10345: (12)                     if state.pmax is not None:
10346: (16)                         if state.last_p != state.pmax:
10347: (20)                             if verbose:
10348: (24)                                 log(f'Calling update() with pmax, because was
overwritten by later calls.')
10349: (20)                             big_enough = update(state.pmax)
10350: (20)                             assert big_enough
10351: (16)                         result = state.pmax_result
10352: (12)                     else:
10353: (16)                         result = state.pmin_result if state.pmin_result else
Story.FitResult(numcalls=state.numcalls)
10354: (12)                     if verbose:
10355: (16)                         log(f'finished. {state.pmin0=} {state.pmax0=} {state.pmax=}:
returning {result=}')
10356: (12)                     return result
10357: (8)                  def update(parameter):
10358: (12)                     '''
10359: (12)                     Evaluates `more, _ = self.place(fn(parameter))`. If `more` is
10360: (12)                     false, then `rect` is big enough to contain `self` and we
10361: (12)                     set `state.pmax=parameter` and return True. Otherwise we set
10362: (12)                     `state.pmin=parameter` and return False.
10363: (12)                     '''
10364: (12)                     rect = fn(parameter)
10365: (12)                     assert isinstance(rect, Rect), f'{type(rect)=} {rect=}'
10366: (12)                     if rect.is_empty:
10367: (16)                         big_enough = False
10368: (16)                         result = Story.FitResult(parameter=parameter,
numcalls=state.numcalls)
10369: (16)                         if verbose:
10370: (20)                             log(f'update(): not calling self.place() because rect is
empty.')
10371: (12)                     else:
10372: (16)                         more, filled = self.place(rect)
10373: (16)                         state.numcalls += 1
10374: (16)                         big_enough = not more
10375: (16)                         result = Story.FitResult(
10376: (24)                                 filled=filled,
10377: (24)                                 more=more,
10378: (24)                                 numcalls=state.numcalls,
10379: (24)                                 parameter=parameter,
10380: (24)                                 rect=rect,
10381: (24)                                 big_enough=big_enough,
10382: (24)                                 )
10383: (16)                         if verbose:
10384: (20)                             log(f'update(): called self.place(): {state.numcalls:>2d}:
{more=} {parameter=} {rect=}.')
10385: (12)                     if big_enough:
10386: (16)                         state.pmax = parameter
10387: (16)                         state.pmax_result = result
10388: (12)                     else:
10389: (16)                         state.pmin = parameter
10390: (16)                         state.pmin_result = result
10391: (12)                     state.last_p = parameter
10392: (12)                     return big_enough
10393: (8)                  def opposite(p, direction):
10394: (12)                     '''
10395: (12)                     Returns same sign as `direction`, larger or smaller than `p` if
10396: (12)                     direction is positive or negative respectively.
```

```
10397: (12)                     '''
10398: (12)                     if p is None or p==0:
10399: (16)                         return direction
10400: (12)                     if direction * p > 0:
10401: (16)                         return 2 * p
10402: (12)                     return -p
10403: (8)                 if state.pmin is None:
10404: (12)                     if verbose: log(f'finding pmin.')
10405: (12)                     parameter = opposite(state.pmax, -1)
10406: (12)                     while 1:
10407: (16)                         if not update(parameter):
10408: (20)                             break
10409: (16)                         parameter *= 2
10410: (8)                 else:
10411: (12)                     if update(state.pmin):
10412: (16)                         if verbose: log(f'{state.pmin=} is big enough.')
10413: (16)                         return ret()
10414: (8)                 if state.pmax is None:
10415: (12)                     if verbose: log(f'finding pmax.')
10416: (12)                     parameter = opposite(state.pmin, +1)
10417: (12)                     while 1:
10418: (16)                         if update(parameter):
10419: (20)                             break
10420: (16)                         parameter *= 2
10421: (8)                 else:
10422: (12)                     if not update(state.pmax):
10423: (16)                         state.pmax = None
10424: (16)                         if verbose: log(f'No solution possible {state.pmax=}.')
10425: (16)                         return ret()
10426: (8)                 if verbose: log(f'doing binary search with {state.pmin=}
{state.pmax=}.')
10427: (8)                 while 1:
10428: (12)                     if state.pmax - state.pmin < delta:
10429: (16)                         return ret()
10430: (12)                     parameter = (state.pmin + state.pmax) / 2
10431: (12)                     update(parameter)
10432: (4)             def fit_scale(self, rect, scale_min=0, scale_max=None, delta=0.001,
verbose=False):
10433: (8)                 '''
10434: (8)                 Finds smallest value `scale` in range `scale_min..scale_max` where
10435: (8)                 `scale * rect` is large enough to contain the story `self`.
10436: (8)                 Returns a `Story.FitResult` instance.
10437: (8)                 :arg width:
10438: (12)                     width of rect.
10439: (8)                 :arg height:
10440: (12)                     height of rect.
10441: (8)                 :arg scale_min:
10442: (12)                     Minimum scale to consider; must be >= 0.
10443: (8)                 :arg scale_max:
10444: (12)                     Maximum scale to consider, must be >= scale_min or `None` for
10445: (12)                     infinite.
10446: (8)                 :arg delta:
10447: (12)                     Maximum error in returned scale.
10448: (8)                 :arg verbose:
10449: (12)                     If true we output diagnostics.
10450: (8)                 '''
10451: (8)                 x0, y0, x1, y1 = rect
10452: (8)                 width = x1 - x0
10453: (8)                 height = y1 - y0
10454: (8)                 def fn(scale):
10455: (12)                     return Rect(x0, y0, x0 + scale*width, y0 + scale*height)
10456: (8)                 return self.fit(fn, scale_min, scale_max, delta, verbose)
10457: (4)             def fit_height(self, width, height_min=0, height_max=None, origin=(0, 0),
delta=0.001, verbose=False):
10458: (8)                 '''
10459: (8)                 Finds smallest height in range `height_min..height_max` where a rect
10460: (8)                 with size `(width, height)` is large enough to contain the story
10461: (8)                 `self`.
10462: (8)                 Returns a `Story.FitResult` instance.
```

```
10463: (8)                  :arg width:
10464: (12)                     width of rect.
10465: (8)                  :arg height_min:
10466: (12)                     Minimum height to consider; must be >= 0.
10467: (8)                  :arg height_max:
10468: (12)                     Maximum height to consider, must be >= height_min or `None` for
10469: (12)                     infinite.
10470: (8)                  :arg origin:
10471: (12)                     `(x0, y0)` of rect.
10472: (8)                  :arg delta:
10473: (12)                     Maximum error in returned height.
10474: (8)                  :arg verbose:
10475: (12)                     If true we output diagnostics.
10476: (8)                  '''
10477: (8)                  x0, y0 = origin
10478: (8)                  x1 = x0 + width
10479: (8)                  def fn(height):
10480: (12)                     return Rect(x0, y0, x1, y0+height)
10481: (8)                  return self.fit(fn, height_min, height_max, delta, verbose)
10482: (4)              def fit_width(self, height, width_min=0, width_max=None, origin=(0, 0),
delta=0.001, verbose=False):
10483: (8)                  '''
10484: (8)                  Finds smallest width in range `width_min..width_max` where a rect with
size
10485: (8)                  `(width, height)` is large enough to contain the story `self`.
10486: (8)                  Returns a `Story.FitResult` instance.
10487: (8)                  Returns a `FitResult` instance.
10488: (8)                  :arg height:
10489: (12)                     height of rect.
10490: (8)                  :arg width_min:
10491: (12)                     Minimum width to consider; must be >= 0.
10492: (8)                  :arg width_max:
10493: (12)                     Maximum width to consider, must be >= width_min or `None` for
10494: (12)                     infinite.
10495: (8)                  :arg origin:
10496: (12)                     `(x0, y0)` of rect.
10497: (8)                  :arg delta:
10498: (12)                     Maximum error in returned width.
10499: (8)                  :arg verbose:
10500: (12)                     If true we output diagnostics.
10501: (8)                  '''
10502: (8)                  x0, y0 = origin
10503: (8)                  y1 = y0 + height
10504: (8)                  def fn(width):
10505: (12)                     return Rect(x0, y0, x0+width, y1)
10506: (8)                  return self.fit(fn, width_min, width_max, delta, verbose)
10507: (0)          class TextPage:
10508: (4)              def __init__(self, *args):
10509: (8)                  if args_match(args, mupdf.FzRect):
10510: (12)                     mediabox = args[0]
10511: (12)                     self.this = mupdf.FzStextPage( mediabox)
10512: (8)                  elif args_match(args, mupdf.FzStextPage):
10513: (12)                     self.this = args[0]
10514: (8)                  else:
10515: (12)                     raise Exception(f'Unrecognised args: {args}')
10516: (8)                  self.thisown = True
10517: (8)                  self.parent = None
10518: (4)              def _extractText(self, format_):
10519: (8)                  this_tpage = self.this
10520: (8)                  res = mupdf.fz_new_buffer(1024)
10521: (8)                  out = mupdf.FzOutput( res)
10522: (8)                  if format_ == 1:
10523: (12)                     mupdf.fz_print_stext_page_as_html(out, this_tpage, 0)
10524: (8)                  elif format_ == 3:
10525: (12)                     mupdf.fz_print_stext_page_as_xml(out, this_tpage, 0)
10526: (8)                  elif format_ == 4:
10527: (12)                     mupdf.fz_print_stext_page_as_xhtml(out, this_tpage, 0)
10528: (8)                  else:
10529: (12)                     JM_print_stext_page_as_text(res, this_tpage)
```

```
10530: (8)                  out.fz_close_output()
10531: (8)                  text = JM_EscapeStrFromBuffer(res)
10532: (8)                  return text
10533: (4)              def _getNewBlockList(self, page_dict, raw):
10534: (8)                  JM_make_textpage_dict(self.this, page_dict, raw)
10535: (4)              def _textpage_dict(self, raw=False):
10536: (8)                  page_dict = {"width": self.rect.width, "height": self.rect.height}
10537: (8)                  self._getNewBlockList(page_dict, raw)
10538: (8)                  return page_dict
10539: (4)              def extractBLOCKS(self):
10540: (8)                  """Return a list with text block information."""
10541: (8)                  if g_use_extra:
10542: (12)                     return extra.extractBLOCKS(self.this)
10543: (8)                  block_n = -1
10544: (8)                  this_tpage = self.this
10545: (8)                  tp_rect = mupdf.FzRect(this_tpage.m_internal.mediabox)
10546: (8)                  res = mupdf.fz_new_buffer(1024)
10547: (8)                  lines = []
10548: (8)                  for block in this_tpage:
10549: (12)                     block_n += 1
10550: (12)                     blockrect = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)
10551: (12)                     if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_TEXT:
10552: (16)                         mupdf.fz_clear_buffer(res) # set text buffer to empty
10553: (16)                         line_n = -1
10554: (16)                         last_char = 0
10555: (16)                         for line in block:
10556: (20)                             line_n += 1
10557: (20)                             linerect = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)
10558: (20)                             for ch in line:
10559: (24)                                 cbbox = JM_char_bbox(line, ch)
10560: (24)                                 if (not JM_rects_overlap(tp_rect, cbbox)
10561: (32)                                         and not mupdf.fz_is_infinite_rect(tp_rect)
10562: (32)                                         ):
10563: (28)                                     continue
10564: (24)                                 JM_append_rune(res, ch.m_internal.c)
10565: (24)                                 last_char = ch.m_internal.c
10566: (24)                                 linerect = mupdf.fz_union_rect(linerect, cbbox)
10567: (20)                             if last_char != 10 and not
mupdf.fz_is_empty_rect(linerect):
10568: (24)                                 mupdf.fz_append_byte(res, 10)
10569: (20)                             blockrect = mupdf.fz_union_rect(blockrect, linerect)
10570: (16)                         text = JM_EscapeStrFromBuffer(res)
10571: (12)                     elif (JM_rects_overlap(tp_rect, block.m_internal.bbox)
10572: (20)                             or mupdf.fz_is_infinite_rect(tp_rect)
10573: (20)                             ):
10574: (16)                         img = block.i_image()
10575: (16)                         cs = img.colorspace()
10576: (16)                         text = "<image: %s, width: %d, height: %d, bpc: %d>" % (
10577: (24)                                 mupdf.fz_colorspace_name(cs),
10578: (24)                                 img.w(), img.h(), img.bpc()
10579: (24)                                 )
10580: (16)                         blockrect = mupdf.fz_union_rect(blockrect,
mupdf.FzRect(block.m_internal.bbox))
10581: (12)                     if not mupdf.fz_is_empty_rect(blockrect):
10582: (16)                         litem = (
10583: (24)                                 blockrect.x0,
10584: (24)                                 blockrect.y0,
10585: (24)                                 blockrect.x1,
10586: (24)                                 blockrect.y1,
10587: (24)                                 text,
10588: (24)                                 block_n,
10589: (24)                                 block.m_internal.type,
10590: (24)                                 )
10591: (16)                         lines.append(litem)
10592: (8)                  return lines
10593: (4)              def extractDICT(self, cb=None, sort=False) -> dict:
10594: (8)                  """Return page content as a Python dict of images and text spans."""
10595: (8)                  val = self._textpage_dict(raw=False)
10596: (8)                  if cb is not None:
```

```
10597: (12)                     val["width"] = cb.width
10598: (12)                     val["height"] = cb.height
10599: (8)                  if sort is True:
10600: (12)                     blocks = val["blocks"]
10601: (12)                     blocks.sort(key=lambda b: (b["bbox"][3], b["bbox"][0]))
10602: (12)                     val["blocks"] = blocks
10603: (8)                  return val
10604: (4)              def extractHTML(self) -> str:
10605: (8)                  """Return page content as a HTML string."""
10606: (8)                  return self._extractText(1)
10607: (4)              def extractIMGINFO(self, hashes=0):
10608: (8)                  """Return a list with image meta information."""
10609: (8)                  block_n = -1
10610: (8)                  this_tpage = self.this
10611: (8)                  rc = []
10612: (8)                  for block in this_tpage:
10613: (12)                     block_n += 1
10614: (12)                     if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_TEXT:
10615: (16)                         continue
10616: (12)                     img = block.i_image()
10617: (12)                     img_size = 0
10618: (12)                     if mupdf_version_tuple >= (1, 24):
10619: (16)                         compr_buff = mupdf.fz_compressed_image_buffer(img)
10620: (16)                         if compr_buff.m_internal:
10621: (20)                             img_size = compr_buff.fz_compressed_buffer_size()
10622: (20)                             compr_buff = None
10623: (12)                     else:
10624: (16)                         compr_buff =
mupdf.ll_fz_compressed_image_buffer(img.m_internal)
10625: (16)                         if compr_buff:
10626: (20)                             img_size = mupdf.ll_fz_compressed_buffer_size(compr_buff)
10627: (12)                     if hashes:
10628: (16)                         r = mupdf.FzIrect(FZ_MIN_INF_RECT, FZ_MIN_INF_RECT,
FZ_MAX_INF_RECT, FZ_MAX_INF_RECT)
10629: (16)                         assert mupdf.fz_is_infinite_irect(r)
10630: (16)                         m = mupdf.FzMatrix(img.w(), 0, 0, img.h(), 0, 0)
10631: (16)                         pix, w, h = mupdf.fz_get_pixmap_from_image(img, r, m)
10632: (16)                         digest = mupdf.fz_md5_pixmap2(pix)
10633: (16)                         digest = bytes(digest)
10634: (16)                         if img_size == 0:
10635: (20)                             img_size = img.w() * img.h() * img.n()
10636: (12)                     cs =
mupdf.FzColorspace(mupdf.ll_fz_keep_colorspace(img.m_internal.colorspace))
10637: (12)                     block_dict = dict()
10638: (12)                     block_dict[ dictkey_number] = block_n
10639: (12)                     block_dict[ dictkey_bbox] = JM_py_from_rect(block.m_internal.bbox)
10640: (12)                     block_dict[ dictkey_matrix] =
JM_py_from_matrix(block.i_transform())
10641: (12)                     block_dict[ dictkey_width] = img.w()
10642: (12)                     block_dict[ dictkey_height] = img.h()
10643: (12)                     block_dict[ dictkey_colorspace] = mupdf.fz_colorspace_n(cs)
10644: (12)                     block_dict[ dictkey_cs_name] = mupdf.fz_colorspace_name(cs)
10645: (12)                     block_dict[ dictkey_xres] = img.xres()
10646: (12)                     block_dict[ dictkey_yres] = img.yres()
10647: (12)                     block_dict[ dictkey_bpc] = img.bpc()
10648: (12)                     block_dict[ dictkey_size] = img_size
10649: (12)                     if hashes:
10650: (16)                         block_dict[ "digest"] = digest
10651: (12)                     rc.append(block_dict)
10652: (8)                  return rc
10653: (4)              def extractJSON(self, cb=None, sort=False) -> str:
10654: (8)                  """Return 'extractDICT' converted to JSON format."""
10655: (8)                  import base64
10656: (8)                  import json
10657: (8)                  val = self._textpage_dict(raw=False)
10658: (8)                  class b64encode(json.JSONEncoder):
10659: (12)                     def default(self, s):
10660: (16)                         if type(s) in (bytes, bytearray):
10661: (20)                             return base64.b64encode(s).decode()
```

```
10662: (8)                  if cb is not None:
10663: (12)                     val["width"] = cb.width
10664: (12)                     val["height"] = cb.height
10665: (8)                  if sort is True:
10666: (12)                     blocks = val["blocks"]
10667: (12)                     blocks.sort(key=lambda b: (b["bbox"][3], b["bbox"][0]))
10668: (12)                     val["blocks"] = blocks
10669: (8)                  val = json.dumps(val, separators=(",", ":"), cls=b64encode, indent=1)
10670: (8)                  return val
10671: (4)              def extractRAWDICT(self, cb=None, sort=False) -> dict:
10672: (8)                  """Return page content as a Python dict of images and text
characters."""
10673: (8)                  val = self._textpage_dict(raw=True)
10674: (8)                  if cb is not None:
10675: (12)                     val["width"] = cb.width
10676: (12)                     val["height"] = cb.height
10677: (8)                  if sort is True:
10678: (12)                     blocks = val["blocks"]
10679: (12)                     blocks.sort(key=lambda b: (b["bbox"][3], b["bbox"][0]))
10680: (12)                     val["blocks"] = blocks
10681: (8)                  return val
10682: (4)              def extractRAWJSON(self, cb=None, sort=False) -> str:
10683: (8)                  """Return 'extractRAWDICT' converted to JSON format."""
10684: (8)                  import base64
10685: (8)                  import json
10686: (8)                  val = self._textpage_dict(raw=True)
10687: (8)                  class b64encode(json.JSONEncoder):
10688: (12)                     def default(self,s):
10689: (16)                         if type(s) in (bytes, bytearray):
10690: (20)                             return base64.b64encode(s).decode()
10691: (8)                  if cb is not None:
10692: (12)                     val["width"] = cb.width
10693: (12)                     val["height"] = cb.height
10694: (8)                  if sort is True:
10695: (12)                     blocks = val["blocks"]
10696: (12)                     blocks.sort(key=lambda b: (b["bbox"][3], b["bbox"][0]))
10697: (12)                     val["blocks"] = blocks
10698: (8)                  val = json.dumps(val, separators=(",", ":"), cls=b64encode, indent=1)
10699: (8)                  return val
10700: (4)              def extractSelection(self, pointa, pointb):
10701: (8)                  a = JM_point_from_py(pointa)
10702: (8)                  b = JM_point_from_py(pointb)
10703: (8)                  found = mupdf.fz_copy_selection(self.this, a, b, 0)
10704: (8)                  return found
10705: (4)              def extractText(self, sort=False) -> str:
10706: (8)                  """Return simple, bare text on the page."""
10707: (8)                  if sort is False:
10708: (12)                     return self._extractText(0)
10709: (8)                  blocks = self.extractBLOCKS()[:]
10710: (8)                  blocks.sort(key=lambda b: (b[3], b[0]))
10711: (8)                  return "".join([b[4] for b in blocks])
10712: (4)              def extractTextbox(self, rect):
10713: (8)                  this_tpage = self.this
10714: (8)                  assert isinstance(this_tpage, mupdf.FzStextPage)
10715: (8)                  area = JM_rect_from_py(rect)
10716: (8)                  found = JM_copy_rectangle(this_tpage, area)
10717: (8)                  rc = PyUnicode_DecodeRawUnicodeEscape(found)
10718: (8)                  return rc
10719: (4)              def extractWORDS(self, delimiters=None):
10720: (8)                  """Return a list with text word information."""
10721: (8)                  if g_use_extra:
10722: (12)                     return extra.extractWORDS(self.this, delimiters)
10723: (8)                  buflen = 0
10724: (8)                  block_n = -1
10725: (8)                  wbbox = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)  # word bbox
10726: (8)                  this_tpage = self.this
10727: (8)                  tp_rect = mupdf.FzRect(this_tpage.m_internal.mediabox)
10728: (8)                  lines = None
10729: (8)                  buff = mupdf.fz_new_buffer(64)
```

```
10730: (8)                  lines = []
10731: (8)                  for block in this_tpage:
10732: (12)                     block_n += 1
10733: (12)                     if block.m_internal.type != mupdf.FZ_STEXT_BLOCK_TEXT:
10734: (16)                         continue
10735: (12)                     line_n = -1
10736: (12)                     for line in block:
10737: (16)                         line_n += 1
10738: (16)                         word_n = 0                  # word counter per line
10739: (16)                         mupdf.fz_clear_buffer(buff) # reset word buffer
10740: (16)                         buflen = 0                  # reset char counter
10741: (16)                         for ch in line:
10742: (20)                             cbbox = JM_char_bbox(line, ch)
10743: (20)                             if (not JM_rects_overlap(tp_rect, cbbox)
10744: (28)                                     and not mupdf.fz_is_infinite_rect(tp_rect)
10745: (28)                                     ):
10746: (24)                                 continue
10747: (20)                             word_delimiter = JM_is_word_delimiter(ch.m_internal.c,
delimiters)
10748: (20)                             if word_delimiter:
10749: (24)                                 if buflen == 0:
10750: (28)                                     continue    # skip delimiters at line start
10751: (24)                                 if not mupdf.fz_is_empty_rect(wbbox):
10752: (28)                                     word_n, wbbox = JM_append_word(lines, buff, wbbox,
block_n, line_n, word_n)
10753: (24)                                 mupdf.fz_clear_buffer(buff)
10754: (24)                                 buflen = 0  # reset char counter
10755: (24)                                 continue
10756: (20)                             JM_append_rune(buff, ch.m_internal.c)
10757: (20)                             buflen += 1
10758: (20)                             wbbox = mupdf.fz_union_rect(wbbox, JM_char_bbox(line, ch))
10759: (16)                         if buflen and not mupdf.fz_is_empty_rect(wbbox):
10760: (20)                             word_n, wbbox = JM_append_word(lines, buff, wbbox,
block_n, line_n, word_n)
10761: (16)                         buflen = 0
10762: (8)                  return lines
10763: (4)              def extractXHTML(self) -> str:
10764: (8)                  """Return page content as a XHTML string."""
10765: (8)                  return self._extractText(4)
10766: (4)              def extractXML(self) -> str:
10767: (8)                  """Return page content as a XML string."""
10768: (8)                  return self._extractText(3)
10769: (4)              def poolsize(self):
10770: (8)                  """TextPage current poolsize."""
10771: (8)                  tpage = self.this
10772: (8)                  pool = mupdf.Pool(tpage.m_internal.pool)
10773: (8)                  size = mupdf.fz_pool_size( pool)
10774: (8)                  pool.m_internal = None  # Ensure that pool's destructor does not free
the pool.
10775: (8)                  return size
10776: (4)              @property
10777: (4)              def rect(self):
10778: (8)                  """Page rectangle."""
10779: (8)                  this_tpage = self.this
10780: (8)                  mediabox = this_tpage.m_internal.mediabox
10781: (8)                  val = JM_py_from_rect(mediabox)
10782: (8)                  val = Rect(val)
10783: (8)                  return val
10784: (4)              def search(self, needle, hit_max=0, quads=1):
10785: (8)                  """Locate 'needle' returning rects or quads."""
10786: (8)                  val = JM_search_stext_page(self.this, needle)
10787: (8)                  if not val:
10788: (12)                     return val
10789: (8)                  items = len(val)
10790: (8)                  for i in range(items):  # change entries to quads or rects
10791: (12)                     q = Quad(val[i])
10792: (12)                     if quads:
10793: (16)                         val[i] = q
10794: (12)                     else:
```

```
10795: (16)                         val[i] = q.rect
10796: (8)                  if quads:
10797: (12)                     return val
10798: (8)                  i = 0  # join overlapping rects on the same line
10799: (8)                  while i < items - 1:
10800: (12)                     v1 = val[i]
10801: (12)                     v2 = val[i + 1]
10802: (12)                     if v1.y1 != v2.y1 or (v1 & v2).is_empty:
10803: (16)                         i += 1
10804: (16)                         continue  # no overlap on same line
10805: (12)                     val[i] = v1 | v2  # join rectangles
10806: (12)                     del val[i + 1]  # remove v2
10807: (12)                     items -= 1  # reduce item count
10808: (8)                  return val
10809: (4)              extractTEXT = extractText
10810: (0)          class TextWriter:
10811: (4)              def __init__(self, page_rect, opacity=1, color=None):
10812: (8)                  """Stores text spans for later output on compatible PDF pages."""
10813: (8)                  self.this = mupdf.fz_new_text()
10814: (8)                  self.opacity = opacity
10815: (8)                  self.color = color
10816: (8)                  self.rect = Rect(page_rect)
10817: (8)                  self.ctm = Matrix(1, 0, 0, -1, 0, self.rect.height)
10818: (8)                  self.ictm = ~self.ctm
10819: (8)                  self.last_point = Point()
10820: (8)                  self.last_point.__doc__ = "Position following last text insertion."
10821: (8)                  self.text_rect = Rect()
10822: (8)                  self.text_rect.__doc__ = "Accumulated area of text spans."
10823: (8)                  self.used_fonts = set()
10824: (8)                  self.thisown = True
10825: (4)              @property
10826: (4)              def _bbox(self):
10827: (8)                  val = JM_py_from_rect( mupdf.fz_bound_text( self.this,
mupdf.FzStrokeState(None), mupdf.FzMatrix()))
10828: (8)                  val = Rect(val)
10829: (8)                  return val
10830: (4)              def append(self, pos, text, font=None, fontsize=11, language=None,
right_to_left=0, small_caps=0):
10831: (8)                  """Store 'text' at point 'pos' using 'font' and 'fontsize'."""
10832: (8)                  pos = Point(pos) * self.ictm
10833: (8)                  if font is None:
10834: (12)                     font = Font("helv")
10835: (8)                  if not font.is_writable:
10836: (12)                     if 0:
10837: (16)                         log( '{font.this.m_internal.name=}')
10838: (16)                         log( '{font.this.m_internal.t3matrix=}')
10839: (16)                         log( '{font.this.m_internal.bbox=}')
10840: (16)                         log( '{font.this.m_internal.glyph_count=}')
10841: (16)                         log( '{font.this.m_internal.use_glyph_bbox=}')
10842: (16)                         log( '{font.this.m_internal.width_count=}')
10843: (16)                         log( '{font.this.m_internal.width_default=}')
10844: (16)                         log( '{font.this.m_internal.has_digest=}')
10845: (16)                         log( 'Unsupported font {font.name=}')
10846: (16)                         if mupdf_cppyy:
10847: (20)                             import cppyy
10848: (20)                             log( f'Unsupported font
{cppyy.gbl.mupdf_font_name(font.this.m_internal)=}')
10849: (12)                     raise ValueError("Unsupported font '%s'." % font.name)
10850: (8)                  if right_to_left:
10851: (12)                     text = self.clean_rtl(text)
10852: (12)                     text = "".join(reversed(text))
10853: (12)                     right_to_left = 0
10854: (8)                  lang = mupdf.fz_text_language_from_string(language)
10855: (8)                  p = JM_point_from_py(pos)
10856: (8)                  trm = mupdf.fz_make_matrix(fontsize, 0, 0, fontsize, p.x, p.y)
10857: (8)                  markup_dir = 0
10858: (8)                  wmode = 0
10859: (8)                  if small_caps == 0:
10860: (12)                     trm = mupdf.fz_show_string( self.this, font.this, trm, text,
```

```
wmode, right_to_left, markup_dir, lang)
10861: (8)                  else:
10862: (12)                     trm = JM_show_string_cs( self.this, font.this, trm, text, wmode,
right_to_left, markup_dir, lang)
10863: (8)                  val = JM_py_from_matrix(trm)
10864: (8)                  self.last_point = Point(val[-2:]) * self.ctm
10865: (8)                  self.text_rect = self._bbox * self.ctm
10866: (8)                  val = self.text_rect, self.last_point
10867: (8)                  if font.flags["mono"] == 1:
10868: (12)                     self.used_fonts.add(font)
10869: (8)                  return val
10870: (4)              def appendv(self, pos, text, font=None, fontsize=11, language=None,
small_caps=False):
10871: (8)                  lheight = fontsize * 1.2
10872: (8)                  for c in text:
10873: (12)                     self.append(pos, c, font=font, fontsize=fontsize,
10874: (16)                         language=language, small_caps=small_caps)
10875: (12)                     pos.y += lheight
10876: (8)                  return self.text_rect, self.last_point
10877: (4)              def clean_rtl(self, text):
10878: (8)                  """Revert the sequence of Latin text parts.
10879: (8)                  Text with right-to-left writing direction (Arabic, Hebrew) often
10880: (8)                  contains Latin parts, which are written in left-to-right: numbers,
names,
10881: (8)                  etc. For output as PDF text we need *everything* in right-to-left.
10882: (8)                  E.g. an input like "<arabic> ABCDE FG HIJ <arabic> KL <arabic>" will
be
10883: (8)                  converted to "<arabic> JIH GF EDCBA <arabic> LK <arabic>". The Arabic
10884: (8)                  parts remain untouched.
10885: (8)                  Args:
10886: (12)                     text: str
10887: (8)                  Returns:
10888: (12)                     Massaged string.
10889: (8)                  """
10890: (8)                  if not text:
10891: (12)                     return text
10892: (8)                  words = text.split(" ")
10893: (8)                  idx = []
10894: (8)                  for i in range(len(words)):
10895: (12)                     w = words[i]
10896: (12)                     if not (len(w) < 2 or max([ord(c) for c in w]) > 255):
10897: (16)                         words[i] = "".join(reversed(w))
10898: (16)                         idx.append(i)  # stored index of Latin word
10899: (8)                  idx2 = []  # store indices of adjacent Latin words
10900: (8)                  for i in range(len(idx)):
10901: (12)                     if idx2 == []:  # empty yet?
10902: (16)                         idx2.append(idx[i]) # store Latin word number
10903: (12)                     elif idx[i] > idx2[-1] + 1:  # large gap to last?
10904: (16)                         if len(idx2) > 1:  # at least two consecutives?
10905: (20)                             words[idx2[0] : idx2[-1] + 1] = reversed(
10906: (24)                                 words[idx2[0] : idx2[-1] + 1]
10907: (20)                             )  # revert their sequence
10908: (16)                         idx2 = [idx[i]]  # re-initialize
10909: (12)                     elif idx[i] == idx2[-1] + 1:  # new adjacent Latin word
10910: (16)                         idx2.append(idx[i])
10911: (8)                  text = " ".join(words)
10912: (8)                  return text
10913: (4)              def write_text(self, page, color=None, opacity=-1, overlay=1, morph=None,
matrix=None, render_mode=0, oc=0):
10914: (8)                  """Write the text to a PDF page having the TextWriter's page size.
10915: (8)                  Args:
10916: (12)                     page: a PDF page having same size.
10917: (12)                     color: override text color.
10918: (12)                     opacity: override transparency.
10919: (12)                     overlay: put in foreground or background.
10920: (12)                     morph: tuple(Point, Matrix), apply a matrix with a fixpoint.
10921: (12)                     matrix: Matrix to be used instead of 'morph' argument.
10922: (12)                     render_mode: (int) PDF render mode operator 'Tr'.
10923: (8)                  """
```

```
10924: (8)                  CheckParent(page)
10925: (8)                  if abs(self.rect - page.rect) > 1e-3:
10926: (12)                     raise ValueError("incompatible page rect")
10927: (8)                  if morph is not None:
10928: (12)                     if (type(morph) not in (tuple, list)
10929: (20)                             or type(morph[0]) is not Point
10930: (20)                             or type(morph[1]) is not Matrix
10931: (20)                             ):
10932: (16)                         raise ValueError("morph must be (Point, Matrix) or None")
10933: (8)                  if matrix is not None and morph is not None:
10934: (12)                     raise ValueError("only one of matrix, morph is allowed")
10935: (8)                  if getattr(opacity, "__float__", None) is None or opacity == -1:
10936: (12)                     opacity = self.opacity
10937: (8)                  if color is None:
10938: (12)                     color = self.color
10939: (8)                  if 1:
10940: (12)                     pdfpage = page._pdf_page()
10941: (12)                     alpha = 1
10942: (12)                     if opacity >= 0 and opacity < 1:
10943: (16)                         alpha = opacity
10944: (12)                     ncol = 1
10945: (12)                     dev_color = [0, 0, 0, 0]
10946: (12)                     if color:
10947: (16)                         ncol, dev_color = JM_color_FromSequence(color)
10948: (12)                     if ncol == 3:
10949: (16)                         colorspace = mupdf.fz_device_rgb()
10950: (12)                     elif ncol == 4:
10951: (16)                         colorspace = mupdf.fz_device_cmyk()
10952: (12)                     else:
10953: (16)                         colorspace = mupdf.fz_device_gray()
10954: (12)                     resources = mupdf.pdf_new_dict(pdfpage.doc(), 5)
10955: (12)                     contents = mupdf.fz_new_buffer(1024)
10956: (12)                     dev = mupdf.pdf_new_pdf_device( pdfpage.doc(), mupdf.FzMatrix(),
resources, contents)
10957: (12)                     mupdf.fz_fill_text(
10958: (20)                             dev,
10959: (20)                             self.this,
10960: (20)                             mupdf.FzMatrix(),
10961: (20)                             colorspace,
10962: (20)                             dev_color,
10963: (20)                             alpha,
10964: (20)                             mupdf.FzColorParams(mupdf.fz_default_color_params),
10965: (20)                             )
10966: (12)                     mupdf.fz_close_device( dev)
10967: (12)                     max_nums = JM_merge_resources( pdfpage, resources)
10968: (12)                     cont_string = JM_EscapeStrFromBuffer( contents)
10969: (12)                     result = (max_nums, cont_string)
10970: (12)                     val = result
10971: (8)                  max_nums = val[0]
10972: (8)                  content = val[1]
10973: (8)                  max_alp, max_font = max_nums
10974: (8)                  old_cont_lines = content.splitlines()
10975: (8)                  optcont = page._get_optional_content(oc)
10976: (8)                  if optcont is not None:
10977: (12)                     bdc = "/OC /%s BDC" % optcont
10978: (12)                     emc = "EMC"
10979: (8)                  else:
10980: (12)                     bdc = emc = ""
10981: (8)                  new_cont_lines = ["q"]
10982: (8)                  if bdc:
10983: (12)                     new_cont_lines.append(bdc)
10984: (8)                  cb = page.cropbox_position
10985: (8)                  if page.rotation in (90, 270):
10986: (12)                     delta = page.rect.height - page.rect.width
10987: (8)                  else:
10988: (12)                     delta = 0
10989: (8)                  mb = page.mediabox
10990: (8)                  if bool(cb) or mb.y0 != 0 or delta != 0:
10991: (12)                     new_cont_lines.append(f"1 0 0 1 {_format_g((cb.x, cb.y + mb.y0 -
```

```
delta))} cm")
10992: (8)                  if morph:
10993: (12)                     p = morph[0] * self.ictm
10994: (12)                     delta = Matrix(1, 1).pretranslate(p.x, p.y)
10995: (12)                     matrix = ~delta * morph[1] * delta
10996: (8)                  if morph or matrix:
10997: (12)                     new_cont_lines.append(_format_g(JM_TUPLE(matrix)) + " cm")
10998: (8)                  for line in old_cont_lines:
10999: (12)                     if line.endswith(" cm"):
11000: (16)                         continue
11001: (12)                     if line == "BT":
11002: (16)                         new_cont_lines.append(line)
11003: (16)                         new_cont_lines.append("%i Tr" % render_mode)
11004: (16)                         continue
11005: (12)                     if line.endswith(" gs"):
11006: (16)                         alp = int(line.split()[0][4:]) + max_alp
11007: (16)                         line = "/Alp%i gs" % alp
11008: (12)                     elif line.endswith(" Tf"):
11009: (16)                         temp = line.split()
11010: (16)                         fsize = float(temp[1])
11011: (16)                         if render_mode != 0:
11012: (20)                             w = fsize * 0.05
11013: (16)                         else:
11014: (20)                             w = 1
11015: (16)                         new_cont_lines.append(_format_g(w) + " w")
11016: (16)                         font = int(temp[0][2:]) + max_font
11017: (16)                         line = " ".join(["/F%i" % font] + temp[1:])
11018: (12)                     elif line.endswith(" rg"):
11019: (16)                         new_cont_lines.append(line.replace("rg", "RG"))
11020: (12)                     elif line.endswith(" g"):
11021: (16)                         new_cont_lines.append(line.replace(" g", " G"))
11022: (12)                     elif line.endswith(" k"):
11023: (16)                         new_cont_lines.append(line.replace(" k", " K"))
11024: (12)                     new_cont_lines.append(line)
11025: (8)                  if emc:
11026: (12)                     new_cont_lines.append(emc)
11027: (8)                  new_cont_lines.append("Q\n")
11028: (8)                  content = "\n".join(new_cont_lines).encode("utf-8")
11029: (8)                  TOOLS._insert_contents(page, content, overlay=overlay)
11030: (8)                  val = None
11031: (8)                  for font in self.used_fonts:
11032: (12)                     repair_mono_font(page, font)
11033: (8)                  return val
11034: (0)          class IRect:
11035: (4)              """
11036: (4)              IRect() - all zeros
11037: (4)              IRect(x0, y0, x1, y1) - 4 coordinates
11038: (4)              IRect(top-left, x1, y1) - point and 2 coordinates
11039: (4)              IRect(x0, y0, bottom-right) - 2 coordinates and point
11040: (4)              IRect(top-left, bottom-right) - 2 points
11041: (4)              IRect(sequ) - new from sequence or rect-like
11042: (4)              """
11043: (4)              def __add__(self, p):
11044: (8)                  return Rect.__add__(self, p).round()
11045: (4)              def __and__(self, x):
11046: (8)                  return Rect.__and__(self, x).round()
11047: (4)              def __contains__(self, x):
11048: (8)                  return Rect.__contains__(self, x)
11049: (4)              def __eq__(self, r):
11050: (8)                  if not hasattr(r, "__len__"):
11051: (12)                     return False
11052: (8)                  return len(r) == 4 and self.x0 == r[0] and self.y0 == r[1] and self.x1
== r[2] and self.y1 == r[3]
11053: (4)              def __getitem__(self, i):
11054: (8)                  return (self.x0, self.y0, self.x1, self.y1)[i]
11055: (4)              def __init__(self, *args, p0=None, p1=None, x0=None, y0=None, x1=None,
y1=None):
11056: (8)                  self.x0, self.y0, self.x1, self.y1 = util_make_irect( *args, p0=p0,
p1=p1, x0=x0, y0=y0, x1=x1, y1=y1)
```

```
11057: (4)              def __len__(self):
11058: (8)                  return 4
11059: (4)              def __mul__(self, m):
11060: (8)                  return Rect.__mul__(self, m).round()
11061: (4)              def __neg__(self):
11062: (8)                  return IRect(-self.x0, -self.y0, -self.x1, -self.y1)
11063: (4)              def __or__(self, x):
11064: (8)                  return Rect.__or__(self, x).round()
11065: (4)              def __pos__(self):
11066: (8)                  return IRect(self)
11067: (4)              def __repr__(self):
11068: (8)                  return "IRect" + str(tuple(self))
11069: (4)              def __setitem__(self, i, v):
11070: (8)                  v = int(v)
11071: (8)                  if   i == 0: self.x0 = v
11072: (8)                  elif i == 1: self.y0 = v
11073: (8)                  elif i == 2: self.x1 = v
11074: (8)                  elif i == 3: self.y1 = v
11075: (8)                  else:
11076: (12)                     raise IndexError("index out of range")
11077: (8)                  return None
11078: (4)              def __sub__(self, p):
11079: (8)                  return Rect.__sub__(self, p).round()
11080: (4)              def __truediv__(self, m):
11081: (8)                  return Rect.__truediv__(self, m).round()
11082: (4)              @property
11083: (4)              def bottom_left(self):
11084: (8)                  """Bottom-left corner."""
11085: (8)                  return Point(self.x0, self.y1)
11086: (4)              @property
11087: (4)              def bottom_right(self):
11088: (8)                  """Bottom-right corner."""
11089: (8)                  return Point(self.x1, self.y1)
11090: (4)              @property
11091: (4)              def height(self):
11092: (8)                  return max(0, self.y1 - self.y0)
11093: (4)              def include_point(self, p):
11094: (8)                  """Extend rectangle to include point p."""
11095: (8)                  rect = self.rect.include_point(p)
11096: (8)                  return rect.irect
11097: (4)              def include_rect(self, r):
11098: (8)                  """Extend rectangle to include rectangle r."""
11099: (8)                  rect = self.rect.include_rect(r)
11100: (8)                  return rect.irect
11101: (4)              def intersect(self, r):
11102: (8)                  """Restrict rectangle to intersection with rectangle r."""
11103: (8)                  return Rect.intersect(self, r).round()
11104: (4)              def intersects(self, x):
11105: (8)                  return Rect.intersects(self, x)
11106: (4)              @property
11107: (4)              def is_empty(self):
11108: (8)                  """True if rectangle area is empty."""
11109: (8)                  return self.x0 >= self.x1 or self.y0 >= self.y1
11110: (4)              @property
11111: (4)              def is_infinite(self):
11112: (8)                  """True if rectangle is infinite."""
11113: (8)                  return self.x0 == self.y0 == FZ_MIN_INF_RECT and self.x1 == self.y1 ==
FZ_MAX_INF_RECT
11114: (4)              @property
11115: (4)              def is_valid(self):
11116: (8)                  """True if rectangle is valid."""
11117: (8)                  return self.x0 <= self.x1 and self.y0 <= self.y1
11118: (4)              def morph(self, p, m):
11119: (8)                  """Morph with matrix-like m and point-like p.
11120: (8)                  Returns a new quad."""
11121: (8)                  if self.is_infinite:
11122: (12)                     return INFINITE_QUAD()
11123: (8)                  return self.quad.morph(p, m)
11124: (4)              def norm(self):
```

```
11125: (8)                  return math.sqrt(sum([c*c for c in self]))
11126: (4)              def normalize(self):
11127: (8)                  """Replace rectangle with its valid version."""
11128: (8)                  if self.x1 < self.x0:
11129: (12)                     self.x0, self.x1 = self.x1, self.x0
11130: (8)                  if self.y1 < self.y0:
11131: (12)                     self.y0, self.y1 = self.y1, self.y0
11132: (8)                  return self
11133: (4)              @property
11134: (4)              def quad(self):
11135: (8)                  """Return Quad version of rectangle."""
11136: (8)                  return Quad(self.tl, self.tr, self.bl, self.br)
11137: (4)              @property
11138: (4)              def rect(self):
11139: (8)                  return Rect(self)
11140: (4)              @property
11141: (4)              def top_left(self):
11142: (8)                  """Top-left corner."""
11143: (8)                  return Point(self.x0, self.y0)
11144: (4)              @property
11145: (4)              def top_right(self):
11146: (8)                  """Top-right corner."""
11147: (8)                  return Point(self.x1, self.y0)
11148: (4)              def torect(self, r):
11149: (8)                  """Return matrix that converts to target rect."""
11150: (8)                  r = Rect(r)
11151: (8)                  if self.is_infinite or self.is_empty or r.is_infinite or r.is_empty:
11152: (12)                     raise ValueError("rectangles must be finite and not empty")
11153: (8)                  return (
11154: (16)                         Matrix(1, 0, 0, 1, -self.x0, -self.y0)
11155: (16)                         * Matrix(r.width / self.width, r.height / self.height)
11156: (16)                         * Matrix(1, 0, 0, 1, r.x0, r.y0)
11157: (16)                         )
11158: (4)              def transform(self, m):
11159: (8)                  return Rect.transform(self, m).round()
11160: (4)              @property
11161: (4)              def width(self):
11162: (8)                  return max(0, self.x1 - self.x0)
11163: (4)              br = bottom_right
11164: (4)              bl = bottom_left
11165: (4)              tl = top_left
11166: (4)              tr = top_right
11167: (0)          if 1:
11168: (4)              _self = sys.modules[__name__]
11169: (4)              if 1:
11170: (8)                  for _name, _value in mupdf.__dict__.items():
11171: (12)                     if _name.startswith(('PDF_', 'UCDN_SCRIPT_')):
11172: (16)                         if _name.startswith('PDF_ENUM_NAME_'):
11173: (20)                             pass
11174: (16)                         else:
11175: (20)                             setattr(_self, _name, _value)
11176: (4)              else:
11177: (8)                  for _name, _value in inspect.getmembers(mupdf):
11178: (12)                     if _name.startswith(('PDF_', 'UCDN_SCRIPT_')):
11179: (16)                         if _name.startswith('PDF_ENUM_NAME_'):
11180: (20)                             pass
11181: (16)                         else:
11182: (20)                             setattr(_self, _name, _value)
11183: (4)              PDF_SIGNATURE_DEFAULT_APPEARANCE = (0
11184: (12)                     | mupdf.PDF_SIGNATURE_SHOW_LABELS
11185: (12)                     | mupdf.PDF_SIGNATURE_SHOW_DN
11186: (12)                     | mupdf.PDF_SIGNATURE_SHOW_DATE
11187: (12)                     | mupdf.PDF_SIGNATURE_SHOW_TEXT_NAME
11188: (12)                     | mupdf.PDF_SIGNATURE_SHOW_GRAPHIC_NAME
11189: (12)                     | mupdf.PDF_SIGNATURE_SHOW_LOGO
11190: (12)                     )
11191: (4)              assert mupdf.UCDN_EAST_ASIAN_H == 1
11192: (4)              assert PDF_TX_FIELD_IS_MULTILINE == mupdf.PDF_TX_FIELD_IS_MULTILINE #
noqa: F821
```

```
(4)             assert UCDN_SCRIPT_ADLAM == mupdf.UCDN_SCRIPT_ADLAM # noqa: F821
(4)             del _self, _name, _value
(0)         _adobe_glyphs = {}
(0)         _adobe_unicodes = {}
(0)         AnyType = typing.Any
(0)         Base14_fontnames = (
(4)             "Courier",
(4)             "Courier-Oblique",
(4)             "Courier-Bold",
(4)             "Courier-BoldOblique",
(4)             "Helvetica",
(4)             "Helvetica-Oblique",
(4)             "Helvetica-Bold",
(4)             "Helvetica-BoldOblique",
(4)             "Times-Roman",
(4)             "Times-Italic",
(4)             "Times-Bold",
(4)             "Times-BoldItalic",
(4)             "Symbol",
(4)             "ZapfDingbats",
(4)             )
(0)         Base14_fontdict = {}
(0)         for f in Base14_fontnames:
(4)             Base14_fontdict[f.lower()] = f
(0)         Base14_fontdict["helv"] = "Helvetica"
(0)         Base14_fontdict["heit"] = "Helvetica-Oblique"
(0)         Base14_fontdict["hebo"] = "Helvetica-Bold"
(0)         Base14_fontdict["hebi"] = "Helvetica-BoldOblique"
(0)         Base14_fontdict["cour"] = "Courier"
(0)         Base14_fontdict["coit"] = "Courier-Oblique"
(0)         Base14_fontdict["cobo"] = "Courier-Bold"
(0)         Base14_fontdict["cobi"] = "Courier-BoldOblique"
(0)         Base14_fontdict["tiro"] = "Times-Roman"
(0)         Base14_fontdict["tibo"] = "Times-Bold"
(0)         Base14_fontdict["tiit"] = "Times-Italic"
(0)         Base14_fontdict["tibi"] = "Times-BoldItalic"
(0)         Base14_fontdict["symb"] = "Symbol"
(0)         Base14_fontdict["zadb"] = "ZapfDingbats"
(0)         EPSILON = 1e-5
(0)         FLT_EPSILON = 1e-5
(0)         FZ_MIN_INF_RECT = -0x80000000
(0)         FZ_MAX_INF_RECT = 0x7fffff80
(0)         JM_annot_id_stem = "fitz"
(0)         JM_mupdf_warnings_store = []
(0)         JM_mupdf_show_errors = 1
(0)         JM_mupdf_show_warnings = 0
(0)         PDF_OC_ON = 0
(0)         PDF_OC_TOGGLE = 1
(0)         PDF_OC_OFF = 2
(0)         LINK_NONE = 0
(0)         LINK_GOTO = 1
(0)         LINK_URI = 2
(0)         LINK_LAUNCH = 3
(0)         LINK_NAMED = 4
(0)         LINK_GOTOR = 5
(0)         LINK_FLAG_L_VALID = 1
(0)         LINK_FLAG_T_VALID = 2
(0)         LINK_FLAG_R_VALID = 4
(0)         LINK_FLAG_B_VALID = 8
(0)         LINK_FLAG_FIT_H = 16
(0)         LINK_FLAG_FIT_V = 32
(0)         LINK_FLAG_R_IS_ZOOM = 64
(0)         SigFlag_SignaturesExist = 1
(0)         SigFlag_AppendOnly = 2
(0)         STAMP_Approved = 0
(0)         STAMP_AsIs = 1
(0)         STAMP_Confidential = 2
(0)         STAMP_Departmental = 3
(0)         STAMP_Experimental = 4
```

```
11262: (0)          STAMP_Expired = 5
11263: (0)          STAMP_Final = 6
11264: (0)          STAMP_ForComment = 7
11265: (0)          STAMP_ForPublicRelease = 8
11266: (0)          STAMP_NotApproved = 9
11267: (0)          STAMP_NotForPublicRelease = 10
11268: (0)          STAMP_Sold = 11
11269: (0)          STAMP_TopSecret = 12
11270: (0)          STAMP_Draft = 13
11271: (0)          TEXT_ALIGN_LEFT = 0
11272: (0)          TEXT_ALIGN_CENTER = 1
11273: (0)          TEXT_ALIGN_RIGHT = 2
11274: (0)          TEXT_ALIGN_JUSTIFY = 3
11275: (0)          TEXT_FONT_SUPERSCRIPT = 1
11276: (0)          TEXT_FONT_ITALIC = 2
11277: (0)          TEXT_FONT_SERIFED = 4
11278: (0)          TEXT_FONT_MONOSPACED = 8
11279: (0)          TEXT_FONT_BOLD = 16
11280: (0)          TEXT_OUTPUT_TEXT = 0
11281: (0)          TEXT_OUTPUT_HTML = 1
11282: (0)          TEXT_OUTPUT_JSON = 2
11283: (0)          TEXT_OUTPUT_XML = 3
11284: (0)          TEXT_OUTPUT_XHTML = 4
11285: (0)          TEXT_PRESERVE_LIGATURES = 1
11286: (0)          TEXT_PRESERVE_WHITESPACE = 2
11287: (0)          TEXT_PRESERVE_IMAGES = 4
11288: (0)          TEXT_INHIBIT_SPACES = 8
11289: (0)          TEXT_DEHYPHENATE = 16
11290: (0)          TEXT_PRESERVE_SPANS = 32
11291: (0)          TEXT_MEDIABOX_CLIP = 64
11292: (0)          TEXT_CID_FOR_UNKNOWN_UNICODE = 128
11293: (0)          TEXTFLAGS_WORDS = (0
11294: (8)                  | TEXT_PRESERVE_LIGATURES
11295: (8)                  | TEXT_PRESERVE_WHITESPACE
11296: (8)                  | TEXT_MEDIABOX_CLIP
11297: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11298: (8)                  )
11299: (0)          TEXTFLAGS_BLOCKS = (0
11300: (8)                  | TEXT_PRESERVE_LIGATURES
11301: (8)                  | TEXT_PRESERVE_WHITESPACE
11302: (8)                  | TEXT_MEDIABOX_CLIP
11303: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11304: (8)                  )
11305: (0)          TEXTFLAGS_DICT = (0
11306: (8)                  | TEXT_PRESERVE_LIGATURES
11307: (8)                  | TEXT_PRESERVE_WHITESPACE
11308: (8)                  | TEXT_MEDIABOX_CLIP
11309: (8)                  | TEXT_PRESERVE_IMAGES
11310: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11311: (8)                  )
11312: (0)          TEXTFLAGS_RAWDICT = TEXTFLAGS_DICT
11313: (0)          TEXTFLAGS_SEARCH = (0
11314: (8)                  | TEXT_PRESERVE_WHITESPACE
11315: (8)                  | TEXT_MEDIABOX_CLIP
11316: (8)                  | TEXT_DEHYPHENATE
11317: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11318: (8)                  )
11319: (0)          TEXTFLAGS_HTML = (0
11320: (8)                  | TEXT_PRESERVE_LIGATURES
11321: (8)                  | TEXT_PRESERVE_WHITESPACE
11322: (8)                  | TEXT_MEDIABOX_CLIP
11323: (8)                  | TEXT_PRESERVE_IMAGES
11324: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11325: (8)                  )
11326: (0)          TEXTFLAGS_XHTML = (0
11327: (8)                  | TEXT_PRESERVE_LIGATURES
11328: (8)                  | TEXT_PRESERVE_WHITESPACE
11329: (8)                  | TEXT_MEDIABOX_CLIP
11330: (8)                  | TEXT_PRESERVE_IMAGES
```

```
11331: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11332: (8)                  )
11333: (0)          TEXTFLAGS_XML = (0
11334: (8)                  | TEXT_PRESERVE_LIGATURES
11335: (8)                  | TEXT_PRESERVE_WHITESPACE
11336: (8)                  | TEXT_MEDIABOX_CLIP
11337: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11338: (8)                  )
11339: (0)          TEXTFLAGS_TEXT = (0
11340: (8)                  | TEXT_PRESERVE_LIGATURES
11341: (8)                  | TEXT_PRESERVE_WHITESPACE
11342: (8)                  | TEXT_MEDIABOX_CLIP
11343: (8)                  | TEXT_CID_FOR_UNKNOWN_UNICODE
11344: (8)                  )
11345: (0)          TEXT_ENCODING_LATIN = 0
11346: (0)          TEXT_ENCODING_GREEK = 1
11347: (0)          TEXT_ENCODING_CYRILLIC = 2
11348: (0)          TOOLS_JM_UNIQUE_ID = 0
11349: (0)          CS_RGB = 1
11350: (0)          CS_GRAY = 2
11351: (0)          CS_CMYK = 3
11352: (0)          PDF_BM_Color = "Color"
11353: (0)          PDF_BM_ColorBurn = "ColorBurn"
11354: (0)          PDF_BM_ColorDodge = "ColorDodge"
11355: (0)          PDF_BM_Darken = "Darken"
11356: (0)          PDF_BM_Difference = "Difference"
11357: (0)          PDF_BM_Exclusion = "Exclusion"
11358: (0)          PDF_BM_HardLight = "HardLight"
11359: (0)          PDF_BM_Hue = "Hue"
11360: (0)          PDF_BM_Lighten = "Lighten"
11361: (0)          PDF_BM_Luminosity = "Luminosity"
11362: (0)          PDF_BM_Multiply = "Multiply"
11363: (0)          PDF_BM_Normal = "Normal"
11364: (0)          PDF_BM_Overlay = "Overlay"
11365: (0)          PDF_BM_Saturation = "Saturation"
11366: (0)          PDF_BM_Screen = "Screen"
11367: (0)          PDF_BM_SoftLight = "Softlight"
11368: (0)          TEXT_FONT_SUPERSCRIPT = 1
11369: (0)          TEXT_FONT_ITALIC = 2
11370: (0)          TEXT_FONT_SERIFED = 4
11371: (0)          TEXT_FONT_MONOSPACED = 8
11372: (0)          TEXT_FONT_BOLD = 16
11373: (0)          annot_skel = {
11374: (8)                  "goto1": lambda a, b, c, d, e: f"<</A<</S/GoTo/D[{a} 0 R/XYZ
{_format_g((b, c, d))}]>>/Rect[{e}]/BS<<W 0>>/Subtype/Link>>",
11375: (8)                  "goto2": lambda a, b: f"<</A<</S/GoTo/D{a}>>/Rect[{b}]/BS<<W
0>>/Subtype/Link>>",
11376: (8)                  "gotor1": lambda a, b, c, d, e, f, g: f"<</A<</S/GoToR/D[{a} /XYZ
{_format_g((b, c, d))}]/F<<F({e})/UF({f})/Type/Filespec>>>>/Rect[{g}]/BS<<W 0>>/Subtype/Link>>",
11377: (8)                  "gotor2": lambda a, b, c: f"
<</A<</S/GoToR/D{a}/F({b})>>/Rect[{c}]/BS<<W 0>>/Subtype/Link>>",
11378: (8)                  "launch": lambda a, b, c: f"
<</A<</S/Launch/F<<F({a})/UF({b})/Type/Filespec>>>>/Rect[{c}]/BS<<W 0>>/Subtype/Link>>",
11379: (8)                  "uri": lambda a, b: f"<</A<</S/URI/URI({a})>>/Rect[{b}]/BS<<W
0>>/Subtype/Link>>",
11380: (8)                  "named": lambda a, b: f"
<</A<</S/GoTo/D({a})/Type/Action>>/Rect[{b}]/BS<<W 0>>/Subtype/Link>>",
11381: (8)                  }
11382: (0)          class FileDataError(RuntimeError):
11383: (4)              """Raised for documents with file structure issues."""
11384: (4)              pass
11385: (0)          class FileNotFoundError(RuntimeError):
11386: (4)              """Raised if file does not exist."""
11387: (4)              pass
11388: (0)          class EmptyFileError(FileDataError):
11389: (4)              """Raised when creating documents from zero-length data."""
11390: (4)              pass
11391: (0)          csRGB = Colorspace(CS_RGB)
11392: (0)          csGRAY = Colorspace(CS_GRAY)
```

```
(0)          csCMYK = Colorspace(CS_CMYK)
(0)          dictkey_align = "align"
(0)          dictkey_asc = "ascender"
(0)          dictkey_bbox = "bbox"
(0)          dictkey_blocks = "blocks"
(0)          dictkey_bpc = "bpc"
(0)          dictkey_c = "c"
(0)          dictkey_chars = "chars"
(0)          dictkey_color = "color"
(0)          dictkey_colorspace = "colorspace"
(0)          dictkey_content = "content"
(0)          dictkey_creationDate = "creationDate"
(0)          dictkey_cs_name = "cs-name"
(0)          dictkey_da = "da"
(0)          dictkey_dashes = "dashes"
(0)          dictkey_desc = "desc"
(0)          dictkey_desc = "descender"
(0)          dictkey_dir = "dir"
(0)          dictkey_effect = "effect"
(0)          dictkey_ext = "ext"
(0)          dictkey_filename = "filename"
(0)          dictkey_fill = "fill"
(0)          dictkey_flags = "flags"
(0)          dictkey_font = "font"
(0)          dictkey_glyph = "glyph"
(0)          dictkey_height = "height"
(0)          dictkey_id = "id"
(0)          dictkey_image = "image"
(0)          dictkey_items = "items"
(0)          dictkey_length = "length"
(0)          dictkey_lines = "lines"
(0)          dictkey_matrix = "transform"
(0)          dictkey_modDate = "modDate"
(0)          dictkey_name = "name"
(0)          dictkey_number = "number"
(0)          dictkey_origin = "origin"
(0)          dictkey_rect = "rect"
(0)          dictkey_size = "size"
(0)          dictkey_smask = "smask"
(0)          dictkey_spans = "spans"
(0)          dictkey_stroke = "stroke"
(0)          dictkey_style = "style"
(0)          dictkey_subject = "subject"
(0)          dictkey_text = "text"
(0)          dictkey_title = "title"
(0)          dictkey_type = "type"
(0)          dictkey_ufilename = "ufilename"
(0)          dictkey_width = "width"
(0)          dictkey_wmode = "wmode"
(0)          dictkey_xref = "xref"
(0)          dictkey_xres = "xres"
(0)          dictkey_yres = "yres"
(0)          try:
(4)              from pymupdf_fonts import fontdescriptors, fontbuffers
(4)              fitz_fontdescriptors = fontdescriptors.copy()
(4)              for k in fitz_fontdescriptors.keys():
(8)                  fitz_fontdescriptors[k]["loader"] = fontbuffers[k]
(4)              del fontdescriptors, fontbuffers
(0)          except ImportError:
(4)              fitz_fontdescriptors = {}
(0)          symbol_glyphs = (   # Glyph list for the built-in font 'Symbol'
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
(8)                  (183, 0.46),
```

```
11462: (8)                  (183, 0.46),
11463: (8)                  (183, 0.46),
11464: (8)                  (183, 0.46),
11465: (8)                  (183, 0.46),
11466: (8)                  (183, 0.46),
11467: (8)                  (183, 0.46),
11468: (8)                  (183, 0.46),
11469: (8)                  (183, 0.46),
11470: (8)                  (183, 0.46),
11471: (8)                  (183, 0.46),
11472: (8)                  (183, 0.46),
11473: (8)                  (183, 0.46),
11474: (8)                  (183, 0.46),
11475: (8)                  (183, 0.46),
11476: (8)                  (183, 0.46),
11477: (8)                  (183, 0.46),
11478: (8)                  (183, 0.46),
11479: (8)                  (183, 0.46),
11480: (8)                  (183, 0.46),
11481: (8)                  (183, 0.46),
11482: (8)                  (183, 0.46),
11483: (8)                  (183, 0.46),
11484: (8)                  (183, 0.46),
11485: (8)                  (183, 0.46),
11486: (8)                  (32, 0.25),
11487: (8)                  (33, 0.333),
11488: (8)                  (34, 0.713),
11489: (8)                  (35, 0.5),
11490: (8)                  (36, 0.549),
11491: (8)                  (37, 0.833),
11492: (8)                  (38, 0.778),
11493: (8)                  (39, 0.439),
11494: (8)                  (40, 0.333),
11495: (8)                  (41, 0.333),
11496: (8)                  (42, 0.5),
11497: (8)                  (43, 0.549),
11498: (8)                  (44, 0.25),
11499: (8)                  (45, 0.549),
11500: (8)                  (46, 0.25),
11501: (8)                  (47, 0.278),
11502: (8)                  (48, 0.5),
11503: (8)                  (49, 0.5),
11504: (8)                  (50, 0.5),
11505: (8)                  (51, 0.5),
11506: (8)                  (52, 0.5),
11507: (8)                  (53, 0.5),
11508: (8)                  (54, 0.5),
11509: (8)                  (55, 0.5),
11510: (8)                  (56, 0.5),
11511: (8)                  (57, 0.5),
11512: (8)                  (58, 0.278),
11513: (8)                  (59, 0.278),
11514: (8)                  (60, 0.549),
11515: (8)                  (61, 0.549),
11516: (8)                  (62, 0.549),
11517: (8)                  (63, 0.444),
11518: (8)                  (64, 0.549),
11519: (8)                  (65, 0.722),
11520: (8)                  (66, 0.667),
11521: (8)                  (67, 0.722),
11522: (8)                  (68, 0.612),
11523: (8)                  (69, 0.611),
11524: (8)                  (70, 0.763),
11525: (8)                  (71, 0.603),
11526: (8)                  (72, 0.722),
11527: (8)                  (73, 0.333),
11528: (8)                  (74, 0.631),
11529: (8)                  (75, 0.722),
11530: (8)                  (76, 0.686),
```

```
11531: (8)                  (77, 0.889),
11532: (8)                  (78, 0.722),
11533: (8)                  (79, 0.722),
11534: (8)                  (80, 0.768),
11535: (8)                  (81, 0.741),
11536: (8)                  (82, 0.556),
11537: (8)                  (83, 0.592),
11538: (8)                  (84, 0.611),
11539: (8)                  (85, 0.69),
11540: (8)                  (86, 0.439),
11541: (8)                  (87, 0.768),
11542: (8)                  (88, 0.645),
11543: (8)                  (89, 0.795),
11544: (8)                  (90, 0.611),
11545: (8)                  (91, 0.333),
11546: (8)                  (92, 0.863),
11547: (8)                  (93, 0.333),
11548: (8)                  (94, 0.658),
11549: (8)                  (95, 0.5),
11550: (8)                  (96, 0.5),
11551: (8)                  (97, 0.631),
11552: (8)                  (98, 0.549),
11553: (8)                  (99, 0.549),
11554: (8)                  (100, 0.494),
11555: (8)                  (101, 0.439),
11556: (8)                  (102, 0.521),
11557: (8)                  (103, 0.411),
11558: (8)                  (104, 0.603),
11559: (8)                  (105, 0.329),
11560: (8)                  (106, 0.603),
11561: (8)                  (107, 0.549),
11562: (8)                  (108, 0.549),
11563: (8)                  (109, 0.576),
11564: (8)                  (110, 0.521),
11565: (8)                  (111, 0.549),
11566: (8)                  (112, 0.549),
11567: (8)                  (113, 0.521),
11568: (8)                  (114, 0.549),
11569: (8)                  (115, 0.603),
11570: (8)                  (116, 0.439),
11571: (8)                  (117, 0.576),
11572: (8)                  (118, 0.713),
11573: (8)                  (119, 0.686),
11574: (8)                  (120, 0.493),
11575: (8)                  (121, 0.686),
11576: (8)                  (122, 0.494),
11577: (8)                  (123, 0.48),
11578: (8)                  (124, 0.2),
11579: (8)                  (125, 0.48),
11580: (8)                  (126, 0.549),
11581: (8)                  (183, 0.46),
11582: (8)                  (183, 0.46),
11583: (8)                  (183, 0.46),
11584: (8)                  (183, 0.46),
11585: (8)                  (183, 0.46),
11586: (8)                  (183, 0.46),
11587: (8)                  (183, 0.46),
11588: (8)                  (183, 0.46),
11589: (8)                  (183, 0.46),
11590: (8)                  (183, 0.46),
11591: (8)                  (183, 0.46),
11592: (8)                  (183, 0.46),
11593: (8)                  (183, 0.46),
11594: (8)                  (183, 0.46),
11595: (8)                  (183, 0.46),
11596: (8)                  (183, 0.46),
11597: (8)                  (183, 0.46),
11598: (8)                  (183, 0.46),
11599: (8)                  (183, 0.46),
```

```
11600: (8)                  (183, 0.46),
11601: (8)                  (183, 0.46),
11602: (8)                  (183, 0.46),
11603: (8)                  (183, 0.46),
11604: (8)                  (183, 0.46),
11605: (8)                  (183, 0.46),
11606: (8)                  (183, 0.46),
11607: (8)                  (183, 0.46),
11608: (8)                  (183, 0.46),
11609: (8)                  (183, 0.46),
11610: (8)                  (183, 0.46),
11611: (8)                  (183, 0.46),
11612: (8)                  (183, 0.46),
11613: (8)                  (183, 0.46),
11614: (8)                  (160, 0.25),
11615: (8)                  (161, 0.62),
11616: (8)                  (162, 0.247),
11617: (8)                  (163, 0.549),
11618: (8)                  (164, 0.167),
11619: (8)                  (165, 0.713),
11620: (8)                  (166, 0.5),
11621: (8)                  (167, 0.753),
11622: (8)                  (168, 0.753),
11623: (8)                  (169, 0.753),
11624: (8)                  (170, 0.753),
11625: (8)                  (171, 1.042),
11626: (8)                  (172, 0.713),
11627: (8)                  (173, 0.603),
11628: (8)                  (174, 0.987),
11629: (8)                  (175, 0.603),
11630: (8)                  (176, 0.4),
11631: (8)                  (177, 0.549),
11632: (8)                  (178, 0.411),
11633: (8)                  (179, 0.549),
11634: (8)                  (180, 0.549),
11635: (8)                  (181, 0.576),
11636: (8)                  (182, 0.494),
11637: (8)                  (183, 0.46),
11638: (8)                  (184, 0.549),
11639: (8)                  (185, 0.549),
11640: (8)                  (186, 0.549),
11641: (8)                  (187, 0.549),
11642: (8)                  (188, 1),
11643: (8)                  (189, 0.603),
11644: (8)                  (190, 1),
11645: (8)                  (191, 0.658),
11646: (8)                  (192, 0.823),
11647: (8)                  (193, 0.686),
11648: (8)                  (194, 0.795),
11649: (8)                  (195, 0.987),
11650: (8)                  (196, 0.768),
11651: (8)                  (197, 0.768),
11652: (8)                  (198, 0.823),
11653: (8)                  (199, 0.768),
11654: (8)                  (200, 0.768),
11655: (8)                  (201, 0.713),
11656: (8)                  (202, 0.713),
11657: (8)                  (203, 0.713),
11658: (8)                  (204, 0.713),
11659: (8)                  (205, 0.713),
11660: (8)                  (206, 0.713),
11661: (8)                  (207, 0.713),
11662: (8)                  (208, 0.768),
11663: (8)                  (209, 0.713),
11664: (8)                  (210, 0.79),
11665: (8)                  (211, 0.79),
11666: (8)                  (212, 0.89),
11667: (8)                  (213, 0.823),
11668: (8)                  (214, 0.549),
```

```
11669: (8)                  (215, 0.549),
11670: (8)                  (216, 0.713),
11671: (8)                  (217, 0.603),
11672: (8)                  (218, 0.603),
11673: (8)                  (219, 1.042),
11674: (8)                  (220, 0.987),
11675: (8)                  (221, 0.603),
11676: (8)                  (222, 0.987),
11677: (8)                  (223, 0.603),
11678: (8)                  (224, 0.494),
11679: (8)                  (225, 0.329),
11680: (8)                  (226, 0.79),
11681: (8)                  (227, 0.79),
11682: (8)                  (228, 0.786),
11683: (8)                  (229, 0.713),
11684: (8)                  (230, 0.384),
11685: (8)                  (231, 0.384),
11686: (8)                  (232, 0.384),
11687: (8)                  (233, 0.384),
11688: (8)                  (234, 0.384),
11689: (8)                  (235, 0.384),
11690: (8)                  (236, 0.494),
11691: (8)                  (237, 0.494),
11692: (8)                  (238, 0.494),
11693: (8)                  (239, 0.494),
11694: (8)                  (183, 0.46),
11695: (8)                  (241, 0.329),
11696: (8)                  (242, 0.274),
11697: (8)                  (243, 0.686),
11698: (8)                  (244, 0.686),
11699: (8)                  (245, 0.686),
11700: (8)                  (246, 0.384),
11701: (8)                  (247, 0.549),
11702: (8)                  (248, 0.384),
11703: (8)                  (249, 0.384),
11704: (8)                  (250, 0.384),
11705: (8)                  (251, 0.384),
11706: (8)                  (252, 0.494),
11707: (8)                  (253, 0.494),
11708: (8)                  (254, 0.494),
11709: (8)                  (183, 0.46),
11710: (8)                  )
11711: (0)          zapf_glyphs = ( # Glyph list for the built-in font 'ZapfDingbats'
11712: (8)                  (183, 0.788),
11713: (8)                  (183, 0.788),
11714: (8)                  (183, 0.788),
11715: (8)                  (183, 0.788),
11716: (8)                  (183, 0.788),
11717: (8)                  (183, 0.788),
11718: (8)                  (183, 0.788),
11719: (8)                  (183, 0.788),
11720: (8)                  (183, 0.788),
11721: (8)                  (183, 0.788),
11722: (8)                  (183, 0.788),
11723: (8)                  (183, 0.788),
11724: (8)                  (183, 0.788),
11725: (8)                  (183, 0.788),
11726: (8)                  (183, 0.788),
11727: (8)                  (183, 0.788),
11728: (8)                  (183, 0.788),
11729: (8)                  (183, 0.788),
11730: (8)                  (183, 0.788),
11731: (8)                  (183, 0.788),
11732: (8)                  (183, 0.788),
11733: (8)                  (183, 0.788),
11734: (8)                  (183, 0.788),
11735: (8)                  (183, 0.788),
11736: (8)                  (183, 0.788),
11737: (8)                  (183, 0.788),
```

```
11738: (8)                  (183, 0.788),
11739: (8)                  (183, 0.788),
11740: (8)                  (183, 0.788),
11741: (8)                  (183, 0.788),
11742: (8)                  (183, 0.788),
11743: (8)                  (183, 0.788),
11744: (8)                  (32, 0.278),
11745: (8)                  (33, 0.974),
11746: (8)                  (34, 0.961),
11747: (8)                  (35, 0.974),
11748: (8)                  (36, 0.98),
11749: (8)                  (37, 0.719),
11750: (8)                  (38, 0.789),
11751: (8)                  (39, 0.79),
11752: (8)                  (40, 0.791),
11753: (8)                  (41, 0.69),
11754: (8)                  (42, 0.96),
11755: (8)                  (43, 0.939),
11756: (8)                  (44, 0.549),
11757: (8)                  (45, 0.855),
11758: (8)                  (46, 0.911),
11759: (8)                  (47, 0.933),
11760: (8)                  (48, 0.911),
11761: (8)                  (49, 0.945),
11762: (8)                  (50, 0.974),
11763: (8)                  (51, 0.755),
11764: (8)                  (52, 0.846),
11765: (8)                  (53, 0.762),
11766: (8)                  (54, 0.761),
11767: (8)                  (55, 0.571),
11768: (8)                  (56, 0.677),
11769: (8)                  (57, 0.763),
11770: (8)                  (58, 0.76),
11771: (8)                  (59, 0.759),
11772: (8)                  (60, 0.754),
11773: (8)                  (61, 0.494),
11774: (8)                  (62, 0.552),
11775: (8)                  (63, 0.537),
11776: (8)                  (64, 0.577),
11777: (8)                  (65, 0.692),
11778: (8)                  (66, 0.786),
11779: (8)                  (67, 0.788),
11780: (8)                  (68, 0.788),
11781: (8)                  (69, 0.79),
11782: (8)                  (70, 0.793),
11783: (8)                  (71, 0.794),
11784: (8)                  (72, 0.816),
11785: (8)                  (73, 0.823),
11786: (8)                  (74, 0.789),
11787: (8)                  (75, 0.841),
11788: (8)                  (76, 0.823),
11789: (8)                  (77, 0.833),
11790: (8)                  (78, 0.816),
11791: (8)                  (79, 0.831),
11792: (8)                  (80, 0.923),
11793: (8)                  (81, 0.744),
11794: (8)                  (82, 0.723),
11795: (8)                  (83, 0.749),
11796: (8)                  (84, 0.79),
11797: (8)                  (85, 0.792),
11798: (8)                  (86, 0.695),
11799: (8)                  (87, 0.776),
11800: (8)                  (88, 0.768),
11801: (8)                  (89, 0.792),
11802: (8)                  (90, 0.759),
11803: (8)                  (91, 0.707),
11804: (8)                  (92, 0.708),
11805: (8)                  (93, 0.682),
11806: (8)                  (94, 0.701),
```

```
11807: (8)                  (95, 0.826),
11808: (8)                  (96, 0.815),
11809: (8)                  (97, 0.789),
11810: (8)                  (98, 0.789),
11811: (8)                  (99, 0.707),
11812: (8)                  (100, 0.687),
11813: (8)                  (101, 0.696),
11814: (8)                  (102, 0.689),
11815: (8)                  (103, 0.786),
11816: (8)                  (104, 0.787),
11817: (8)                  (105, 0.713),
11818: (8)                  (106, 0.791),
11819: (8)                  (107, 0.785),
11820: (8)                  (108, 0.791),
11821: (8)                  (109, 0.873),
11822: (8)                  (110, 0.761),
11823: (8)                  (111, 0.762),
11824: (8)                  (112, 0.762),
11825: (8)                  (113, 0.759),
11826: (8)                  (114, 0.759),
11827: (8)                  (115, 0.892),
11828: (8)                  (116, 0.892),
11829: (8)                  (117, 0.788),
11830: (8)                  (118, 0.784),
11831: (8)                  (119, 0.438),
11832: (8)                  (120, 0.138),
11833: (8)                  (121, 0.277),
11834: (8)                  (122, 0.415),
11835: (8)                  (123, 0.392),
11836: (8)                  (124, 0.392),
11837: (8)                  (125, 0.668),
11838: (8)                  (126, 0.668),
11839: (8)                  (183, 0.788),
11840: (8)                  (183, 0.788),
11841: (8)                  (183, 0.788),
11842: (8)                  (183, 0.788),
11843: (8)                  (183, 0.788),
11844: (8)                  (183, 0.788),
11845: (8)                  (183, 0.788),
11846: (8)                  (183, 0.788),
11847: (8)                  (183, 0.788),
11848: (8)                  (183, 0.788),
11849: (8)                  (183, 0.788),
11850: (8)                  (183, 0.788),
11851: (8)                  (183, 0.788),
11852: (8)                  (183, 0.788),
11853: (8)                  (183, 0.788),
11854: (8)                  (183, 0.788),
11855: (8)                  (183, 0.788),
11856: (8)                  (183, 0.788),
11857: (8)                  (183, 0.788),
11858: (8)                  (183, 0.788),
11859: (8)                  (183, 0.788),
11860: (8)                  (183, 0.788),
11861: (8)                  (183, 0.788),
11862: (8)                  (183, 0.788),
11863: (8)                  (183, 0.788),
11864: (8)                  (183, 0.788),
11865: (8)                  (183, 0.788),
11866: (8)                  (183, 0.788),
11867: (8)                  (183, 0.788),
11868: (8)                  (183, 0.788),
11869: (8)                  (183, 0.788),
11870: (8)                  (183, 0.788),
11871: (8)                  (183, 0.788),
11872: (8)                  (183, 0.788),
11873: (8)                  (161, 0.732),
11874: (8)                  (162, 0.544),
11875: (8)                  (163, 0.544),
```

```
11876: (8)                  (164, 0.91),
11877: (8)                  (165, 0.667),
11878: (8)                  (166, 0.76),
11879: (8)                  (167, 0.76),
11880: (8)                  (168, 0.776),
11881: (8)                  (169, 0.595),
11882: (8)                  (170, 0.694),
11883: (8)                  (171, 0.626),
11884: (8)                  (172, 0.788),
11885: (8)                  (173, 0.788),
11886: (8)                  (174, 0.788),
11887: (8)                  (175, 0.788),
11888: (8)                  (176, 0.788),
11889: (8)                  (177, 0.788),
11890: (8)                  (178, 0.788),
11891: (8)                  (179, 0.788),
11892: (8)                  (180, 0.788),
11893: (8)                  (181, 0.788),
11894: (8)                  (182, 0.788),
11895: (8)                  (183, 0.788),
11896: (8)                  (184, 0.788),
11897: (8)                  (185, 0.788),
11898: (8)                  (186, 0.788),
11899: (8)                  (187, 0.788),
11900: (8)                  (188, 0.788),
11901: (8)                  (189, 0.788),
11902: (8)                  (190, 0.788),
11903: (8)                  (191, 0.788),
11904: (8)                  (192, 0.788),
11905: (8)                  (193, 0.788),
11906: (8)                  (194, 0.788),
11907: (8)                  (195, 0.788),
11908: (8)                  (196, 0.788),
11909: (8)                  (197, 0.788),
11910: (8)                  (198, 0.788),
11911: (8)                  (199, 0.788),
11912: (8)                  (200, 0.788),
11913: (8)                  (201, 0.788),
11914: (8)                  (202, 0.788),
11915: (8)                  (203, 0.788),
11916: (8)                  (204, 0.788),
11917: (8)                  (205, 0.788),
11918: (8)                  (206, 0.788),
11919: (8)                  (207, 0.788),
11920: (8)                  (208, 0.788),
11921: (8)                  (209, 0.788),
11922: (8)                  (210, 0.788),
11923: (8)                  (211, 0.788),
11924: (8)                  (212, 0.894),
11925: (8)                  (213, 0.838),
11926: (8)                  (214, 1.016),
11927: (8)                  (215, 0.458),
11928: (8)                  (216, 0.748),
11929: (8)                  (217, 0.924),
11930: (8)                  (218, 0.748),
11931: (8)                  (219, 0.918),
11932: (8)                  (220, 0.927),
11933: (8)                  (221, 0.928),
11934: (8)                  (222, 0.928),
11935: (8)                  (223, 0.834),
11936: (8)                  (224, 0.873),
11937: (8)                  (225, 0.828),
11938: (8)                  (226, 0.924),
11939: (8)                  (227, 0.924),
11940: (8)                  (228, 0.917),
11941: (8)                  (229, 0.93),
11942: (8)                  (230, 0.931),
11943: (8)                  (231, 0.463),
11944: (8)                  (232, 0.883),
```

```
11945: (8)                  (233, 0.836),
11946: (8)                  (234, 0.836),
11947: (8)                  (235, 0.867),
11948: (8)                  (236, 0.867),
11949: (8)                  (237, 0.696),
11950: (8)                  (238, 0.696),
11951: (8)                  (239, 0.874),
11952: (8)                  (183, 0.788),
11953: (8)                  (241, 0.874),
11954: (8)                  (242, 0.76),
11955: (8)                  (243, 0.946),
11956: (8)                  (244, 0.771),
11957: (8)                  (245, 0.865),
11958: (8)                  (246, 0.771),
11959: (8)                  (247, 0.888),
11960: (8)                  (248, 0.967),
11961: (8)                  (249, 0.888),
11962: (8)                  (250, 0.831),
11963: (8)                  (251, 0.873),
11964: (8)                  (252, 0.927),
11965: (8)                  (253, 0.97),
11966: (8)                  (183, 0.788),
11967: (8)                  (183, 0.788),
11968: (8)                  )
11969: (0)          def _read_samples( pixmap, offset, n):
11970: (4)              ret = []
11971: (4)              for i in range( n):
11972: (8)                  ret.append( mupdf.fz_samples_get( pixmap, offset + i))
11973: (4)              return bytes( ret)
11974: (0)          def _INRANGE(v, low, high):
11975: (4)              return low <= v and v <= high
11976: (0)          def _remove_dest_range(pdf, numbers):
11977: (4)              pagecount = mupdf.pdf_count_pages(pdf)
11978: (4)              for i in range(pagecount):
11979: (8)                  n1 = i
11980: (8)                  if n1 in numbers:
11981: (12)                     continue
11982: (8)                  pageref = mupdf.pdf_lookup_page_obj( pdf, i)
11983: (8)                  annots = mupdf.pdf_dict_get( pageref, PDF_NAME('Annots'))
11984: (8)                  if not annots.m_internal:
11985: (12)                     continue
11986: (8)                  len_ = mupdf.pdf_array_len(annots)
11987: (8)                  for j in range(len_ - 1, -1, -1):
11988: (12)                     o = mupdf.pdf_array_get( annots, j)
11989: (12)                     if not mupdf.pdf_name_eq( mupdf.pdf_dict_get( o,
PDF_NAME('Subtype')), PDF_NAME('Link')):
11990: (16)                         continue
11991: (12)                     action = mupdf.pdf_dict_get( o, PDF_NAME('A'))
11992: (12)                     dest = mupdf.pdf_dict_get( o, PDF_NAME('Dest'))
11993: (12)                     if action.m_internal:
11994: (16)                         if not mupdf.pdf_name_eq( mupdf.pdf_dict_get( action,
PDF_NAME('S')), PDF_NAME('GoTo')):
11995: (20)                             continue
11996: (16)                         dest = mupdf.pdf_dict_get( action, PDF_NAME('D'))
11997: (12)                     pno = -1
11998: (12)                     if mupdf.pdf_is_array( dest):
11999: (16)                         target = mupdf.pdf_array_get( dest, 0)
12000: (16)                         pno = mupdf.pdf_lookup_page_number( pdf, target)
12001: (12)                     elif mupdf.pdf_is_string( dest):
12002: (16)                         location, _, _ = mupdf.fz_resolve_link( pdf.super(),
mupdf.pdf_to_text_string( dest))
12003: (16)                         pno = location.page
12004: (12)                     if pno < 0: # page number lookup did not work
12005: (16)                         continue
12006: (12)                     n1 = pno
12007: (12)                     if n1 in numbers:
12008: (16)                         mupdf.pdf_array_delete( annots, j)
12009: (0)          def ASSERT_PDF(cond):
12010: (4)              assert isinstance(cond, (mupdf.PdfPage, mupdf.PdfDocument)),
```

```
f'{type(cond)=} {cond=}'
12011: (4)              if not cond.m_internal:
12012: (8)                  raise Exception(MSG_IS_NO_PDF)
12013: (0)          def EMPTY_IRECT():
12014: (4)              return IRect(FZ_MAX_INF_RECT, FZ_MAX_INF_RECT, FZ_MIN_INF_RECT,
FZ_MIN_INF_RECT)
12015: (0)          def EMPTY_QUAD():
12016: (4)              return EMPTY_RECT().quad
12017: (0)          def EMPTY_RECT():
12018: (4)              return Rect(FZ_MAX_INF_RECT, FZ_MAX_INF_RECT, FZ_MIN_INF_RECT,
FZ_MIN_INF_RECT)
12019: (0)          def ENSURE_OPERATION(pdf):
12020: (4)              if not JM_have_operation(pdf):
12021: (8)                  raise Exception("No journalling operation started")
12022: (0)          def INFINITE_IRECT():
12023: (4)              return IRect(FZ_MIN_INF_RECT, FZ_MIN_INF_RECT, FZ_MAX_INF_RECT,
FZ_MAX_INF_RECT)
12024: (0)          def INFINITE_QUAD():
12025: (4)              return INFINITE_RECT().quad
12026: (0)          def INFINITE_RECT():
12027: (4)              return Rect(FZ_MIN_INF_RECT, FZ_MIN_INF_RECT, FZ_MAX_INF_RECT,
FZ_MAX_INF_RECT)
12028: (0)          def JM_BinFromBuffer(buffer_):
12029: (4)              '''
12030: (4)              Turn fz_buffer into a Python bytes object
12031: (4)              '''
12032: (4)              assert isinstance(buffer_, mupdf.FzBuffer)
12033: (4)              ret = mupdf.fz_buffer_extract_copy(buffer_)
12034: (4)              return ret
12035: (0)          def JM_EscapeStrFromStr(c):
12036: (4)              if c is None:
12037: (8)                  return ''
12038: (4)              assert isinstance(c, str), f'{type(c)=}'
12039: (4)              b = c.encode('utf8', 'surrogateescape')
12040: (4)              ret = ''
12041: (4)              for bb in b:
12042: (8)                  ret += chr(bb)
12043: (4)              return ret
12044: (0)          def JM_BufferFromBytes(stream):
12045: (4)              '''
12046: (4)              Make fz_buffer from a PyBytes, PyByteArray or io.BytesIO object. If a text
12047: (4)              io.BytesIO, we convert to binary by encoding as utf8.
12048: (4)              '''
12049: (4)              if isinstance(stream, (bytes, bytearray)):
12050: (8)                  data = stream
12051: (4)              elif hasattr(stream, 'getvalue'):
12052: (8)                  data = stream.getvalue()
12053: (8)                  if isinstance(data, str):
12054: (12)                     data = data.encode('utf-8')
12055: (8)                  if not isinstance(data, (bytes, bytearray)):
12056: (12)                     raise Exception(f'.getvalue() returned unexpected type:
{type(data)}')
12057: (4)              else:
12058: (8)                  return mupdf.FzBuffer()
12059: (4)              return mupdf.fz_new_buffer_from_copied_data(data)
12060: (0)          def JM_FLOAT_ITEM(obj, idx):
12061: (4)              if not PySequence_Check(obj):
12062: (8)                  return None
12063: (4)              return float(obj[idx])
12064: (0)          def JM_INT_ITEM(obj, idx):
12065: (4)              if idx < len(obj):
12066: (8)                  temp = obj[idx]
12067: (8)                  if isinstance(temp, (int, float)):
12068: (12)                     return 0, temp
12069: (4)              return 1, None
12070: (0)          def JM_pixmap_from_page(doc, page, ctm, cs, alpha, annots, clip):
12071: (4)              '''
12072: (4)              Pixmap creation directly using a short-lived displaylist, so we can
support
```

```
12073: (4)              separations.
12074: (4)              '''
12075: (4)              SPOTS_NONE = 0
12076: (4)              SPOTS_OVERPRINT_SIM = 1
12077: (4)              SPOTS_FULL = 2
12078: (4)              FZ_ENABLE_SPOT_RENDERING = True # fixme: this is a build-time setting in
MuPDF's config.h.
12079: (4)              if FZ_ENABLE_SPOT_RENDERING:
12080: (8)                  spots = SPOTS_OVERPRINT_SIM
12081: (4)              else:
12082: (8)                  spots = SPOTS_NONE
12083: (4)              seps = None
12084: (4)              colorspace = cs
12085: (4)              matrix = JM_matrix_from_py(ctm)
12086: (4)              rect = mupdf.fz_bound_page(page)
12087: (4)              rclip = JM_rect_from_py(clip)
12088: (4)              rect = mupdf.fz_intersect_rect(rect, rclip) # no-op if clip is not given
12089: (4)              rect = mupdf.fz_transform_rect(rect, matrix)
12090: (4)              bbox = mupdf.fz_round_rect(rect)
12091: (4)              oi = mupdf.fz_document_output_intent(doc)
12092: (4)              if oi.m_internal:
12093: (8)                  if mupdf.fz_colorspace_n(oi) == mupdf.fz_colorspace_n(cs):
12094: (12)                     colorspace = mupdf.fz_keep_colorspace(oi)
12095: (4)              if spots != SPOTS_NONE:
12096: (8)                  seps = mupdf.fz_page_separations(page)
12097: (8)                  if seps.m_internal:
12098: (12)                     n = mupdf.fz_count_separations(seps)
12099: (12)                     if spots == SPOTS_FULL:
12100: (16)                         for i in range(n):
12101: (20)                             mupdf.fz_set_separation_behavior(seps, i,
mupdf.FZ_SEPARATION_SPOT)
12102: (12)                     else:
12103: (16)                         for i in range(n):
12104: (20)                             mupdf.fz_set_separation_behavior(seps, i,
mupdf.FZ_SEPARATION_COMPOSITE)
12105: (8)                  elif mupdf.fz_page_uses_overprint(page):
12106: (12)                     seps = mupdf.fz_new_separations(0)
12107: (8)                  elif oi.m_internal and mupdf.fz_colorspace_n(oi) !=
mupdf.fz_colorspace_n(colorspace):
12108: (12)                     seps = mupdf.fz_new_separations(0)
12109: (4)              pix = mupdf.fz_new_pixmap_with_bbox(colorspace, bbox, seps, alpha)
12110: (4)              if alpha:
12111: (8)                  mupdf.fz_clear_pixmap(pix)
12112: (4)              else:
12113: (8)                  mupdf.fz_clear_pixmap_with_value(pix, 0xFF)
12114: (4)              dev = mupdf.fz_new_draw_device(matrix, pix)
12115: (4)              if annots:
12116: (8)                  mupdf.fz_run_page(page, dev, mupdf.FzMatrix(), mupdf.FzCookie())
12117: (4)              else:
12118: (8)                  mupdf.fz_run_page_contents(page, dev, mupdf.FzMatrix(),
mupdf.FzCookie())
12119: (4)              mupdf.fz_close_device(dev)
12120: (4)              return pix
12121: (0)          def JM_StrAsChar(x):
12122: (4)              return x
12123: (0)          def JM_TUPLE(o: typing.Sequence) -> tuple:
12124: (4)              return tuple(map(lambda x: round(x, 5) if abs(x) >= 1e-4 else 0, o))
12125: (0)          def JM_TUPLE3(o: typing.Sequence) -> tuple:
12126: (4)              return tuple(map(lambda x: round(x, 3) if abs(x) >= 1e-3 else 0, o))
12127: (0)          def JM_UnicodeFromStr(s):
12128: (4)              if s is None:
12129: (8)                  return ''
12130: (4)              if isinstance(s, bytes):
12131: (8)                  s = s.decode('utf8')
12132: (4)              assert isinstance(s, str), f'{type(s)=} {s=}'
12133: (4)              return s
12134: (0)          def JM_add_annot_id(annot, stem):
12135: (4)              '''
12136: (4)              Add a unique /NM key to an annotation or widget.
```

```
12137: (4)              Append a number to 'stem' such that the result is a unique name.
12138: (4)              '''
12139: (4)              assert isinstance(annot, mupdf.PdfAnnot)
12140: (4)              page = mupdf.pdf_annot_page( annot)
12141: (4)              annot_obj = mupdf.pdf_annot_obj( annot)
12142: (4)              names = JM_get_annot_id_list(page)
12143: (4)              i = 0
12144: (4)              while 1:
12145: (8)                  stem_id = f'{JM_annot_id_stem}-{stem}{i}'
12146: (8)                  if stem_id not in names:
12147: (12)                     break
12148: (8)                  i += 1
12149: (4)              response = JM_StrAsChar(stem_id)
12150: (4)              name = mupdf.pdf_new_string( response, len(response))
12151: (4)              mupdf.pdf_dict_puts(annot_obj, "NM", name)
12152: (4)              page.doc().m_internal.resynth_required = 0
12153: (0)          def JM_add_oc_object(pdf, ref, xref):
12154: (4)              '''
12155: (4)              Add OC object reference to a dictionary
12156: (4)              '''
12157: (4)              indobj = mupdf.pdf_new_indirect(pdf, xref, 0)
12158: (4)              if not mupdf.pdf_is_dict(indobj):
12159: (8)                  RAISEPY(MSG_BAD_OC_REF, PyExc_ValueError)
12160: (4)              type_ = mupdf.pdf_dict_get(indobj, PDF_NAME('Type'))
12161: (4)              if (mupdf.pdf_objcmp(type_, PDF_NAME('OCG')) == 0
12162: (12)                     or mupdf.pdf_objcmp(type_, PDF_NAME('OCMD')) == 0
12163: (12)                     ):
12164: (8)                  mupdf.pdf_dict_put(ref, PDF_NAME('OC'), indobj)
12165: (4)              else:
12166: (8)                  RAISEPY(MSG_BAD_OC_REF, PyExc_ValueError)
12167: (0)          def JM_annot_border(annot_obj):
12168: (4)              dash_py = list()
12169: (4)              style = None
12170: (4)              width = -1
12171: (4)              clouds = -1
12172: (4)              obj = None
12173: (4)              obj = mupdf.pdf_dict_get( annot_obj, PDF_NAME('Border'))
12174: (4)              if mupdf.pdf_is_array( obj):
12175: (8)                  width = mupdf.pdf_to_real( mupdf.pdf_array_get( obj, 2))
12176: (8)                  if mupdf.pdf_array_len( obj) == 4:
12177: (12)                     dash = mupdf.pdf_array_get( obj, 3)
12178: (12)                     for i in range( mupdf.pdf_array_len( dash)):
12179: (16)                         val = mupdf.pdf_to_int( mupdf.pdf_array_get( dash, i))
12180: (16)                         dash_py.append( val)
12181: (4)              bs_o = mupdf.pdf_dict_get( annot_obj, PDF_NAME('BS'))
12182: (4)              if bs_o.m_internal:
12183: (8)                  width = mupdf.pdf_to_real( mupdf.pdf_dict_get( bs_o, PDF_NAME('W')))
12184: (8)                  style = mupdf.pdf_to_name( mupdf.pdf_dict_get( bs_o, PDF_NAME('S')))
12185: (8)                  if style == '':
12186: (12)                     style = None
12187: (8)                  obj = mupdf.pdf_dict_get( bs_o, PDF_NAME('D'))
12188: (8)                  if obj.m_internal:
12189: (12)                     for i in range( mupdf.pdf_array_len( obj)):
12190: (16)                         val = mupdf.pdf_to_int( mupdf.pdf_array_get( obj, i))
12191: (16)                         dash_py.append( val)
12192: (4)              obj = mupdf.pdf_dict_get( annot_obj, PDF_NAME('BE'))
12193: (4)              if obj.m_internal:
12194: (8)                  clouds = mupdf.pdf_to_int( mupdf.pdf_dict_get( obj, PDF_NAME('I')))
12195: (4)              res = dict()
12196: (4)              res[ dictkey_width] = width
12197: (4)              res[ dictkey_dashes] = tuple( dash_py)
12198: (4)              res[ dictkey_style] = style
12199: (4)              res[ 'clouds'] = clouds
12200: (4)              return res
12201: (0)          def JM_annot_colors(annot_obj):
12202: (4)              res = dict()
12203: (4)              bc = list() # stroke colors
12204: (4)              fc =list()  # fill colors
12205: (4)              o = mupdf.pdf_dict_get(annot_obj, mupdf.PDF_ENUM_NAME_C)
```

```
12206: (4)              if mupdf.pdf_is_array(o):
12207: (8)                  n = mupdf.pdf_array_len(o)
12208: (8)                  for i in range(n):
12209: (12)                     col = mupdf.pdf_to_real( mupdf.pdf_array_get(o, i))
12210: (12)                     bc.append(col)
12211: (4)              res[dictkey_stroke] = bc
12212: (4)              o = mupdf.pdf_dict_gets(annot_obj, "IC")
12213: (4)              if mupdf.pdf_is_array(o):
12214: (8)                  n = mupdf.pdf_array_len(o)
12215: (8)                  for i in range(n):
12216: (12)                     col = mupdf.pdf_to_real( mupdf.pdf_array_get(o, i))
12217: (12)                     fc.append(col)
12218: (4)              res[dictkey_fill] = fc
12219: (4)              return res
12220: (0)          def JM_annot_set_border( border, doc, annot_obj):
12221: (4)              assert isinstance(border, dict)
12222: (4)              obj = None
12223: (4)              dashlen = 0
12224: (4)              nwidth = border.get( dictkey_width)     # new width
12225: (4)              ndashes = border.get( dictkey_dashes)   # new dashes
12226: (4)              nstyle = border.get( dictkey_style)     # new style
12227: (4)              nclouds  = border.get( 'clouds', -1)    # new clouds value
12228: (4)              oborder = JM_annot_border( annot_obj)
12229: (4)              mupdf.pdf_dict_del( annot_obj, PDF_NAME('BS'))
12230: (4)              mupdf.pdf_dict_del( annot_obj, PDF_NAME('BE'))
12231: (4)              mupdf.pdf_dict_del( annot_obj, PDF_NAME('Border'))
12232: (4)              if nwidth < 0:
12233: (8)                  nwidth = oborder.get( dictkey_width)    # no new width: keep current
12234: (4)              if ndashes is None:
12235: (8)                  ndashes = oborder.get( dictkey_dashes)  # no new dashes: keep old
12236: (4)              if nstyle is None:
12237: (8)                  nstyle  = oborder.get( dictkey_style)   # no new style: keep old
12238: (4)              if nclouds < 0:
12239: (8)                  nclouds  = oborder.get( "clouds", -1)   # no new clouds: keep old
12240: (4)              if isinstance( ndashes, tuple) and len( ndashes) > 0:
12241: (8)                  dashlen = len( ndashes)
12242: (8)                  darr = mupdf.pdf_new_array( doc, dashlen)
12243: (8)                  for d in ndashes:
12244: (12)                     mupdf.pdf_array_push_int( darr, d)
12245: (8)                  mupdf.pdf_dict_putl( annot_obj, darr, PDF_NAME('BS'), PDF_NAME('D'))
12246: (4)              mupdf.pdf_dict_putl(
12247: (12)                     annot_obj,
12248: (12)                     mupdf.pdf_new_real( nwidth),
12249: (12)                     PDF_NAME('BS'),
12250: (12)                     PDF_NAME('W'),
12251: (12)                     )
12252: (4)              if dashlen == 0:
12253: (8)                  obj = JM_get_border_style( nstyle)
12254: (4)              else:
12255: (8)                  obj = PDF_NAME('D')
12256: (4)              mupdf.pdf_dict_putl( annot_obj, obj, PDF_NAME('BS'), PDF_NAME('S'))
12257: (4)              if nclouds > 0:
12258: (8)                  mupdf.pdf_dict_put_dict( annot_obj, PDF_NAME('BE'), 2)
12259: (8)                  obj = mupdf.pdf_dict_get( annot_obj, PDF_NAME('BE'))
12260: (8)                  mupdf.pdf_dict_put( obj, PDF_NAME('S'), PDF_NAME('C'))
12261: (8)                  mupdf.pdf_dict_put_int( obj, PDF_NAME('I'), nclouds)
12262: (0)          def make_escape(ch):
12263: (4)              if ch == 92:
12264: (8)                  return "\\u005c"
12265: (4)              elif 32 <= ch <= 127 or ch == 10:
12266: (8)                  return chr(ch)
12267: (4)              elif 0xd800 <= ch <= 0xdfff:  # orphaned surrogate
12268: (8)                  return "\\ufffd"
12269: (4)              elif ch <= 0xffff:
12270: (8)                  return "\\u%04x" % ch
12271: (4)              else:
12272: (8)                  return "\\U%08x" % ch
12273: (0)          def JM_append_rune(buff, ch):
12274: (4)              """
```

```
12275: (4)              APPEND non-ascii runes in unicode escape format to fz_buffer.
12276: (4)              """
12277: (4)              mupdf.fz_append_string(buff, make_escape(ch))
12278: (0)          def JM_append_word(lines, buff, wbbox, block_n, line_n, word_n):
12279: (4)              '''
12280: (4)              Functions for wordlist output
12281: (4)              '''
12282: (4)              s = JM_EscapeStrFromBuffer(buff)
12283: (4)              litem = (
12284: (12)                     wbbox.x0,
12285: (12)                     wbbox.y0,
12286: (12)                     wbbox.x1,
12287: (12)                     wbbox.y1,
12288: (12)                     s,
12289: (12)                     block_n,
12290: (12)                     line_n,
12291: (12)                     word_n,
12292: (12)                     )
12293: (4)              lines.append(litem)
12294: (4)              return word_n + 1, mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)   # word counter
12295: (0)          def JM_add_layer_config( pdf, name, creator, ON):
12296: (4)              '''
12297: (4)              Add OC configuration to the PDF catalog
12298: (4)              '''
12299: (4)              ocp = JM_ensure_ocproperties( pdf)
12300: (4)              configs = mupdf.pdf_dict_get( ocp, PDF_NAME('Configs'))
12301: (4)              if not mupdf.pdf_is_array( configs):
12302: (8)                  configs = mupdf.pdf_dict_put_array( ocp, PDF_NAME('Configs'), 1)
12303: (4)              D = mupdf.pdf_new_dict( pdf, 5)
12304: (4)              mupdf.pdf_dict_put_text_string( D, PDF_NAME('Name'), name)
12305: (4)              if creator is not None:
12306: (8)                  mupdf.pdf_dict_put_text_string( D, PDF_NAME('Creator'), creator)
12307: (4)              mupdf.pdf_dict_put( D, PDF_NAME('BaseState'), PDF_NAME('OFF'))
12308: (4)              onarray = mupdf.pdf_dict_put_array( D, PDF_NAME('ON'), 5)
12309: (4)              if not ON:
12310: (8)                  pass
12311: (4)              else:
12312: (8)                  ocgs = mupdf.pdf_dict_get( ocp, PDF_NAME('OCGs'))
12313: (8)                  n = len(ON)
12314: (8)                  for i in range(n):
12315: (12)                     xref = 0
12316: (12)                     e, xref = JM_INT_ITEM(ON, i)
12317: (12)                     if e == 1:
12318: (16)                         continue
12319: (12)                     ind = mupdf.pdf_new_indirect( pdf, xref, 0)
12320: (12)                     if mupdf.pdf_array_contains( ocgs, ind):
12321: (16)                         mupdf.pdf_array_push( onarray, ind)
12322: (4)              mupdf.pdf_array_push( configs, D)
12323: (0)          def JM_char_bbox(line, ch):
12324: (4)              '''
12325: (4)              return rect of char quad
12326: (4)              '''
12327: (4)              q = JM_char_quad(line, ch)
12328: (4)              r = mupdf.fz_rect_from_quad(q)
12329: (4)              if not line.m_internal.wmode:
12330: (8)                  return r
12331: (4)              if r.y1 < r.y0 + ch.m_internal.size:
12332: (8)                  r.y0 = r.y1 - ch.m_internal.size
12333: (4)              return r
12334: (0)          def JM_char_font_flags(font, line, ch):
12335: (4)              flags = detect_super_script(line, ch)
12336: (4)              flags += mupdf.fz_font_is_italic(font) * TEXT_FONT_ITALIC
12337: (4)              flags += mupdf.fz_font_is_serif(font) * TEXT_FONT_SERIFED
12338: (4)              flags += mupdf.fz_font_is_monospaced(font) * TEXT_FONT_MONOSPACED
12339: (4)              flags += mupdf.fz_font_is_bold(font) * TEXT_FONT_BOLD
12340: (4)              return flags
12341: (0)          def JM_char_quad(line, ch):
12342: (4)              '''
12343: (4)              re-compute char quad if ascender/descender values make no sense
```

```
12344: (4)              '''
12345: (4)              if 1 and g_use_extra:
12346: (8)                  return mupdf.FzQuad(extra.JM_char_quad( line.m_internal,
ch.m_internal))
12347: (4)              assert isinstance(line, mupdf.FzStextLine)
12348: (4)              assert isinstance(ch, mupdf.FzStextChar)
12349: (4)              if _globals.skip_quad_corrections:   # no special handling
12350: (8)                  return ch.quad
12351: (4)              if line.m_internal.wmode:  # never touch vertical write mode
12352: (8)                  return ch.quad
12353: (4)              font = mupdf.FzFont(mupdf.ll_fz_keep_font(ch.m_internal.font))
12354: (4)              asc = JM_font_ascender(font)
12355: (4)              dsc = JM_font_descender(font)
12356: (4)              fsize = ch.m_internal.size
12357: (4)              asc_dsc = asc - dsc + FLT_EPSILON
12358: (4)              if asc_dsc >= 1 and _globals.small_glyph_heights == 0:   # no problem
12359: (8)                  return mupdf.FzQuad(ch.m_internal.quad)
12360: (4)              fsize = ch.m_internal.size
12361: (4)              bbox = mupdf.fz_font_bbox(font)
12362: (4)              fwidth = bbox.x1 - bbox.x0
12363: (4)              if asc < 1e-3:  # probably Tesseract glyphless font
12364: (8)                  dsc = -0.1
12365: (8)                  asc = 0.9
12366: (8)                  asc_dsc = 1.0
12367: (4)              if _globals.small_glyph_heights or asc_dsc < 1:
12368: (8)                  dsc = dsc / asc_dsc
12369: (8)                  asc = asc / asc_dsc
12370: (4)              asc_dsc = asc - dsc
12371: (4)              asc = asc * fsize / asc_dsc
12372: (4)              dsc = dsc * fsize / asc_dsc
12373: (4)              c = line.m_internal.dir.x  # cosine
12374: (4)              s = line.m_internal.dir.y  # sine
12375: (4)              trm1 = mupdf.fz_make_matrix(c, -s, s, c, 0, 0) # derotate
12376: (4)              trm2 = mupdf.fz_make_matrix(c, s, -s, c, 0, 0) # rotate
12377: (4)              if (c == -1):   # left-right flip
12378: (8)                  trm1.d = 1
12379: (8)                  trm2.d = 1
12380: (4)              xlate1 = mupdf.fz_make_matrix(1, 0, 0, 1, -ch.m_internal.origin.x, -
ch.m_internal.origin.y)
12381: (4)              xlate2 = mupdf.fz_make_matrix(1, 0, 0, 1, ch.m_internal.origin.x,
ch.m_internal.origin.y)
12382: (4)              quad = mupdf.fz_transform_quad(mupdf.FzQuad(ch.m_internal.quad), xlate1)
# move origin to (0,0)
12383: (4)              quad = mupdf.fz_transform_quad(quad, trm1) # de-rotate corners
12384: (4)              if c == 1 and quad.ul.y > 0:    # up-down flip
12385: (8)                  quad.ul.y = asc
12386: (8)                  quad.ur.y = asc
12387: (8)                  quad.ll.y = dsc
12388: (8)                  quad.lr.y = dsc
12389: (4)              else:
12390: (8)                  quad.ul.y = -asc
12391: (8)                  quad.ur.y = -asc
12392: (8)                  quad.ll.y = -dsc
12393: (8)                  quad.lr.y = -dsc
12394: (4)              if quad.ll.x < 0:
12395: (8)                  quad.ll.x = 0
12396: (8)                  quad.ul.x = 0
12397: (4)              cwidth = quad.lr.x - quad.ll.x
12398: (4)              if cwidth < FLT_EPSILON:
12399: (8)                  glyph = mupdf.fz_encode_character( font, ch.m_internal.c)
12400: (8)                  if glyph:
12401: (12)                     fwidth = mupdf.fz_advance_glyph( font, glyph,
line.m_internal.wmode)
12402: (12)                     quad.lr.x = quad.ll.x + fwidth * fsize
12403: (12)                     quad.ur.x = quad.lr.x
12404: (4)              quad = mupdf.fz_transform_quad(quad, trm2) # rotate back
12405: (4)              quad = mupdf.fz_transform_quad(quad, xlate2)   # translate back
12406: (4)              return quad
12407: (0)          def JM_choice_options(annot):
```

```
12408: (4)              '''
12409: (4)              return list of choices for list or combo boxes
12410: (4)              '''
12411: (4)              annot_obj = mupdf.pdf_annot_obj( annot.this)
12412: (4)              if mupdf_version_tuple >= (1, 24):
12413: (8)                  opts = mupdf.pdf_choice_widget_options2( annot, 0)
12414: (4)              else:
12415: (8)                  def pdf_choice_widget_options( annot, exportval):
12416: (12)                     optarr = mupdf.pdf_dict_get_inheritable(
mupdf.pdf_annot_obj(annot.this), PDF_NAME('Opt'))
12417: (12)                     n = mupdf.pdf_array_len(optarr)
12418: (12)                     opts = []
12419: (12)                     if not n:
12420: (16)                         return opts
12421: (12)                     optarr = mupdf.pdf_dict_get(annot_obj, PDF_NAME('Opt'))
12422: (12)                     for i in range(n):
12423: (16)                         m = mupdf.pdf_array_len(mupdf.pdf_array_get(optarr, i))
12424: (16)                         if m == 2:
12425: (20)                             val = (
12426: (28)
mupdf.pdf_to_text_string(mupdf.pdf_array_get(mupdf.pdf_array_get(optarr, i), 0)),
12427: (28)
mupdf.pdf_to_text_string(mupdf.pdf_array_get(mupdf.pdf_array_get(optarr, i), 1)),
12428: (28)                                     )
12429: (20)                             opts.append(val)
12430: (16)                         else:
12431: (20)                             val =
JM_UnicodeFromStr(mupdf.pdf_to_text_string(mupdf.pdf_array_get(optarr, i)))
12432: (20)                             opts.append(val)
12433: (12)                     return opts
12434: (8)                  opts = pdf_choice_widget_options( annot, 0)
12435: (4)              n = len( opts)
12436: (4)              if n == 0:
12437: (8)                  return  # wrong widget type
12438: (4)              optarr = mupdf.pdf_dict_get( annot_obj, PDF_NAME('Opt'))
12439: (4)              liste = []
12440: (4)              for i in range( n):
12441: (8)                  m = mupdf.pdf_array_len( mupdf.pdf_array_get( optarr, i))
12442: (8)                  if m == 2:
12443: (12)                     val = (
12444: (20)                             mupdf.pdf_to_text_string( mupdf.pdf_array_get(
mupdf.pdf_array_get( optarr, i), 0)),
12445: (20)                             mupdf.pdf_to_text_string( mupdf.pdf_array_get(
mupdf.pdf_array_get( optarr, i), 1)),
12446: (20)                             )
12447: (12)                     liste.append( val)
12448: (8)                  else:
12449: (12)                     val = mupdf.pdf_to_text_string( mupdf.pdf_array_get( optarr, i))
12450: (12)                     liste.append( val)
12451: (4)              return liste
12452: (0)          def JM_clear_pixmap_rect_with_value(dest, value, b):
12453: (4)              '''
12454: (4)              Clear a pixmap rectangle - my version also supports non-alpha pixmaps
12455: (4)              '''
12456: (4)              b = mupdf.fz_intersect_irect(b, mupdf.fz_pixmap_bbox(dest))
12457: (4)              w = b.x1 - b.x0
12458: (4)              y = b.y1 - b.y0
12459: (4)              if w <= 0 or y <= 0:
12460: (8)                  return 0
12461: (4)              destspan = dest.stride()
12462: (4)              destp = destspan * (b.y0 - dest.y()) + dest.n() * (b.x0 - dest.x())
12463: (4)              if mupdf.fz_colorspace_n(dest.colorspace()) == 4:
12464: (8)                  value = 255 - value
12465: (8)                  while 1:
12466: (12)                     s = destp
12467: (12)                     for x in range(0, w):
12468: (16)                         mupdf.fz_samples_set(dest, s, 0)
12469: (16)                         s += 1
12470: (16)                         mupdf.fz_samples_set(dest, s, 0)
```

```
12471: (16)                         s += 1
12472: (16)                         mupdf.fz_samples_set(dest, s, 0)
12473: (16)                         s += 1
12474: (16)                         mupdf.fz_samples_set(dest, s, value)
12475: (16)                         s += 1
12476: (16)                         if dest.alpha():
12477: (20)                             mupdf.fz_samples_set(dest, s, 255)
12478: (20)                             s += 1
12479: (12)                     destp += destspan
12480: (12)                     if y == 0:
12481: (16)                         break
12482: (12)                     y -= 1
12483: (8)                 return 1
12484: (4)             while 1:
12485: (8)                 s = destp
12486: (8)                 for x in range(w):
12487: (12)                     for k in range(dest.n()-1):
12488: (16)                         mupdf.fz_samples_set(dest, s, value)
12489: (16)                         s += 1
12490: (12)                     if dest.alpha():
12491: (16)                         mupdf.fz_samples_set(dest, s, 255)
12492: (16)                         s += 1
12493: (12)                     else:
12494: (16)                         mupdf.fz_samples_set(dest, s, value)
12495: (16)                         s += 1
12496: (8)                 destp += destspan
12497: (8)                 if y == 0:
12498: (12)                     break
12499: (8)                 y -= 1
12500: (4)             return 1
12501: (0)         def JM_color_FromSequence(color):
12502: (4)             if isinstance(color, (int, float)):    # maybe just a single float
12503: (8)                 color = color[0]
12504: (4)             if not isinstance( color, (list, tuple)):
12505: (8)                 return -1, []
12506: (4)             if len(color) not in (0, 1, 3, 4):
12507: (8)                 return -1, []
12508: (4)             ret = color[:]
12509: (4)             for i in range(len(ret)):
12510: (8)                 if ret[i] < 0 or ret[i] > 1:
12511: (12)                     ret[i] = 1
12512: (4)             return len(ret), ret
12513: (0)         def JM_color_count( pm, clip):
12514: (4)             rc = dict()
12515: (4)             cnt = 0
12516: (4)             irect = mupdf.fz_pixmap_bbox( pm)
12517: (4)             irect = mupdf.fz_intersect_irect(irect,
mupdf.fz_round_rect(JM_rect_from_py(clip)))
12518: (4)             stride = pm.stride()
12519: (4)             width = irect.x1 - irect.x0
12520: (4)             height = irect.y1 - irect.y0
12521: (4)             n = pm.n()
12522: (4)             substride = width * n
12523: (4)             s = stride * (irect.y0 - pm.y()) + (irect.x0 - pm.x()) * n
12524: (4)             oldpix = _read_samples( pm, s, n)
12525: (4)             cnt = 0
12526: (4)             if mupdf.fz_is_empty_irect(irect):
12527: (8)                 return rc
12528: (4)             for i in range( height):
12529: (8)                 for j in range( 0, substride, n):
12530: (12)                     newpix = _read_samples( pm, s + j, n)
12531: (12)                     if newpix != oldpix:
12532: (16)                         pixel = oldpix
12533: (16)                         c = rc.get( pixel, None)
12534: (16)                         if c is not None:
12535: (20)                             cnt += c
12536: (16)                         rc[ pixel] = cnt
12537: (16)                         cnt = 1
12538: (16)                         oldpix = newpix
```

```
12539: (12)                     else:
12540: (16)                         cnt += 1
12541: (8)                 s += stride
12542: (4)             pixel = oldpix
12543: (4)             c = rc.get( pixel)
12544: (4)             if c is not None:
12545: (8)                 cnt += c
12546: (4)             rc[ pixel] = cnt
12547: (4)             return rc
12548: (0)         def JM_compress_buffer(inbuffer):
12549: (4)             '''
12550: (4)             compress char* into a new buffer
12551: (4)             '''
12552: (4)             data, compressed_length = mupdf.fz_new_deflated_data_from_buffer(
12553: (12)                     inbuffer,
12554: (12)                     mupdf.FZ_DEFLATE_BEST,
12555: (12)                     )
12556: (4)             if not data or compressed_length == 0:
12557: (8)                 return None
12558: (4)             buf = mupdf.FzBuffer(mupdf.fz_new_buffer_from_data(data,
compressed_length))
12559: (4)             mupdf.fz_resize_buffer(buf, compressed_length)
12560: (4)             return buf
12561: (0)         def JM_copy_rectangle(page, area):
12562: (4)             need_new_line = 0
12563: (4)             buffer = io.StringIO()
12564: (4)             for block in page:
12565: (8)                 if block.m_internal.type != mupdf.FZ_STEXT_BLOCK_TEXT:
12566: (12)                     continue
12567: (8)                 for line in block:
12568: (12)                     line_had_text = 0
12569: (12)                     for ch in line:
12570: (16)                         r = JM_char_bbox(line, ch)
12571: (16)                         if JM_rects_overlap(area, r):
12572: (20)                             line_had_text = 1
12573: (20)                             if need_new_line:
12574: (24)                                 buffer.write("\n")
12575: (24)                                 need_new_line = 0
12576: (20)                             buffer.write(make_escape(ch.m_internal.c))
12577: (12)                     if line_had_text:
12578: (16)                         need_new_line = 1
12579: (4)             s = buffer.getvalue()   # take over the data
12580: (4)             return s
12581: (0)         def JM_convert_to_pdf(doc, fp, tp, rotate):
12582: (4)             '''
12583: (4)             Convert any MuPDF document to a PDF
12584: (4)             Returns bytes object containing the PDF, created via 'write' function.
12585: (4)             '''
12586: (4)             pdfout = mupdf.PdfDocument()
12587: (4)             incr = 1
12588: (4)             s = fp
12589: (4)             e = tp
12590: (4)             if fp > tp:
12591: (8)                 incr = -1   # count backwards
12592: (8)                 s = tp      # adjust ...
12593: (8)                 e = fp      # ... range
12594: (4)             rot = JM_norm_rotation(rotate)
12595: (4)             i = fp
12596: (4)             while 1:    # interpret & write document pages as PDF pages
12597: (8)                 if not _INRANGE(i, s, e):
12598: (12)                     break
12599: (8)                 page = mupdf.fz_load_page(doc, i)
12600: (8)                 mediabox = mupdf.fz_bound_page(page)
12601: (8)                 dev, resources, contents = mupdf.pdf_page_write(pdfout, mediabox)
12602: (8)                 mupdf.fz_run_page(page, dev, mupdf.FzMatrix(), mupdf.FzCookie())
12603: (8)                 mupdf.fz_close_device(dev)
12604: (8)                 dev = None
12605: (8)                 page_obj = mupdf.pdf_add_page(pdfout, mediabox, rot, resources,
contents)
```

```
12606: (8)                  mupdf.pdf_insert_page(pdfout, -1, page_obj)
12607: (8)                  i += incr
12608: (4)              opts = mupdf.PdfWriteOptions()
12609: (4)              opts.do_garbage         = 4
12610: (4)              opts.do_compress        = 1
12611: (4)              opts.do_compress_images = 1
12612: (4)              opts.do_compress_fonts  = 1
12613: (4)              opts.do_sanitize        = 1
12614: (4)              opts.do_incremental     = 0
12615: (4)              opts.do_ascii           = 0
12616: (4)              opts.do_decompress      = 0
12617: (4)              opts.do_linear          = 0
12618: (4)              opts.do_clean           = 1
12619: (4)              opts.do_pretty          = 0
12620: (4)              res = mupdf.fz_new_buffer(8192)
12621: (4)              out = mupdf.FzOutput(res)
12622: (4)              mupdf.pdf_write_document(pdfout, out, opts)
12623: (4)              out.fz_close_output()
12624: (4)              c = mupdf.fz_buffer_extract_copy(res)
12625: (4)              assert isinstance(c, bytes)
12626: (4)              return c
12627: (0)          def JM_create_widget(doc, page, type, fieldname):
12628: (4)              old_sigflags =
mupdf.pdf_to_int(mupdf.pdf_dict_getp(mupdf.pdf_trailer(doc), "Root/AcroForm/SigFlags"))
12629: (4)              annot = mupdf.pdf_create_annot_raw(page, mupdf.PDF_ANNOT_WIDGET)
12630: (4)              annot_obj = mupdf.pdf_annot_obj(annot)
12631: (4)              try:
12632: (8)                  JM_set_field_type(doc, annot_obj, type)
12633: (8)                  mupdf.pdf_dict_put_text_string(annot_obj, PDF_NAME('T'), fieldname)
12634: (8)                  if type == mupdf.PDF_WIDGET_TYPE_SIGNATURE:
12635: (12)                     sigflags = old_sigflags | (SigFlag_SignaturesExist |
SigFlag_AppendOnly)
12636: (12)                     mupdf.pdf_dict_putl(
12637: (20)                             mupdf.pdf_trailer(doc),
12638: (20)                             mupdf.pdf_new_nt(sigflags),
12639: (20)                             PDF_NAME('Root'),
12640: (20)                             PDF_NAME('AcroForm'),
12641: (20)                             PDF_NAME('SigFlags'),
12642: (20)                             )
12643: (8)                  form = mupdf.pdf_dict_getp(mupdf.pdf_trailer(doc),
"Root/AcroForm/Fields")
12644: (8)                  if not form.m_internal:
12645: (12)                     form = mupdf.pdf_new_array(doc, 1)
12646: (12)                     mupdf.pdf_dict_putl(
12647: (20)                             mupdf.pdf_trailer(doc),
12648: (20)                             form,
12649: (20)                             PDF_NAME('Root'),
12650: (20)                             PDF_NAME('AcroForm'),
12651: (20)                             PDF_NAME('Fields'),
12652: (20)                             )
12653: (8)                  mupdf.pdf_array_push(form, annot_obj)  # Cleanup relies on this
statement being last
12654: (4)              except Exception:
12655: (8)                  if g_exceptions_verbose:    exception_info()
12656: (8)                  mupdf.pdf_delete_annot(page, annot)
12657: (8)                  if type == mupdf.PDF_WIDGET_TYPE_SIGNATURE:
12658: (12)                     mupdf.pdf_dict_putl(
12659: (20)                             mupdf.pdf_trailer(doc),
12660: (20)                             mupdf.pdf_new_int(old_sigflags),
12661: (20)                             PDF_NAME('Root'),
12662: (20)                             PDF_NAME('AcroForm'),
12663: (20)                             PDF_NAME('SigFlags'),
12664: (20)                             )
12665: (8)                  raise
12666: (4)              return annot
12667: (0)          def JM_cropbox(page_obj):
12668: (4)              '''
12669: (4)              return a PDF page's CropBox
12670: (4)              '''
```

```
12671: (4)              if g_use_extra:
12672: (8)                  return extra.JM_cropbox(page_obj)
12673: (4)              mediabox = JM_mediabox(page_obj)
12674: (4)              cropbox = mupdf.pdf_to_rect(
12675: (16)                         mupdf.pdf_dict_get_inheritable(page_obj, PDF_NAME('CropBox'))
12676: (16)                         )
12677: (4)              if mupdf.fz_is_infinite_rect(cropbox) or mupdf.fz_is_empty_rect(cropbox):
12678: (8)                  cropbox = mediabox
12679: (4)              y0 = mediabox.y1 - cropbox.y1
12680: (4)              y1 = mediabox.y1 - cropbox.y0
12681: (4)              cropbox.y0 = y0
12682: (4)              cropbox.y1 = y1
12683: (4)              return cropbox
12684: (0)          def JM_cropbox_size(page_obj):
12685: (4)              rect = JM_cropbox(page_obj)
12686: (4)              w = abs(rect.x1 - rect.x0)
12687: (4)              h = abs(rect.y1 - rect.y0)
12688: (4)              size = mupdf.fz_make_point(w, h)
12689: (4)              return size
12690: (0)          def JM_derotate_page_matrix(page):
12691: (4)              '''
12692: (4)              just the inverse of rotation
12693: (4)              '''
12694: (4)              mp = JM_rotate_page_matrix(page)
12695: (4)              return mupdf.fz_invert_matrix(mp)
12696: (0)          def JM_embed_file(
12697: (8)                  pdf,
12698: (8)                  buf,
12699: (8)                  filename,
12700: (8)                  ufilename,
12701: (8)                  desc,
12702: (8)                  compress,
12703: (8)                  ):
12704: (4)              '''
12705: (4)              embed a new file in a PDF (not only /EmbeddedFiles entries)
12706: (4)              '''
12707: (4)              len_ = 0
12708: (4)              val = mupdf.pdf_new_dict(pdf, 6)
12709: (4)              mupdf.pdf_dict_put_dict(val, PDF_NAME('CI'), 4)
12710: (4)              ef = mupdf.pdf_dict_put_dict(val, PDF_NAME('EF'), 4)
12711: (4)              mupdf.pdf_dict_put_text_string(val, PDF_NAME('F'), filename)
12712: (4)              mupdf.pdf_dict_put_text_string(val, PDF_NAME('UF'), ufilename)
12713: (4)              mupdf.pdf_dict_put_text_string(val, PDF_NAME('Desc'), desc)
12714: (4)              mupdf.pdf_dict_put(val, PDF_NAME('Type'), PDF_NAME('Filespec'))
12715: (4)              bs = b'  '
12716: (4)              f = mupdf.pdf_add_stream(
12717: (12)                     pdf,
12718: (12)                     mupdf.fz_new_buffer_from_copied_data(bs),
12719: (12)                     mupdf.PdfObj(),
12720: (12)                     0,
12721: (12)                     )
12722: (4)              mupdf.pdf_dict_put(ef, PDF_NAME('F'), f)
12723: (4)              JM_update_stream(pdf, f, buf, compress)
12724: (4)              len_, _ = mupdf.fz_buffer_storage(buf)
12725: (4)              mupdf.pdf_dict_put_int(f, PDF_NAME('DL'), len_)
12726: (4)              mupdf.pdf_dict_put_int(f, PDF_NAME('Length'), len_)
12727: (4)              params = mupdf.pdf_dict_put_dict(f, PDF_NAME('Params'), 4)
12728: (4)              mupdf.pdf_dict_put_int(params, PDF_NAME('Size'), len_)
12729: (4)              return val
12730: (0)          def JM_embedded_clean(pdf):
12731: (4)              '''
12732: (4)              perform some cleaning if we have /EmbeddedFiles:
12733: (4)              (1) remove any /Limits if /Names exists
12734: (4)              (2) remove any empty /Collection
12735: (4)              (3) set /PageMode/UseAttachments
12736: (4)              '''
12737: (4)              root = mupdf.pdf_dict_get( mupdf.pdf_trailer( pdf), PDF_NAME('Root'))
12738: (4)              coll = mupdf.pdf_dict_get(root, PDF_NAME('Collection'))
12739: (4)              if coll.m_internal and mupdf.pdf_dict_len(coll) == 0:
```

```
12740: (8)                  mupdf.pdf_dict_del(root, PDF_NAME('Collection'))
12741: (4)              efiles = mupdf.pdf_dict_getl(
12742: (12)                     root,
12743: (12)                     PDF_NAME('Names'),
12744: (12)                     PDF_NAME('EmbeddedFiles'),
12745: (12)                     PDF_NAME('Names'),
12746: (12)                     )
12747: (4)              if efiles.m_internal:
12748: (8)                  mupdf.pdf_dict_put_name(root, PDF_NAME('PageMode'), "UseAttachments")
12749: (0)          def JM_EscapeStrFromBuffer(buff):
12750: (4)              if not buff.m_internal:
12751: (8)                  return ''
12752: (4)              s = mupdf.fz_buffer_extract_copy(buff)
12753: (4)              val = PyUnicode_DecodeRawUnicodeEscape(s, errors='replace')
12754: (4)              return val
12755: (0)          def JM_ensure_identity(pdf):
12756: (4)              '''
12757: (4)              Store ID in PDF trailer
12758: (4)              '''
12759: (4)              id_ = mupdf.pdf_dict_get( mupdf.pdf_trailer(pdf), PDF_NAME('ID'))
12760: (4)              if not id_.m_internal:
12761: (8)                  rnd0 = mupdf.fz_memrnd2(16)
12762: (8)                  rnd = ''
12763: (8)                  for i in rnd0:
12764: (12)                     rnd += chr(i)
12765: (8)                  id_ = mupdf.pdf_dict_put_array( mupdf.pdf_trailer( pdf),
PDF_NAME('ID'), 2)
12766: (8)                  mupdf.pdf_array_push( id_, mupdf.pdf_new_string( rnd, len(rnd)))
12767: (8)                  mupdf.pdf_array_push( id_, mupdf.pdf_new_string( rnd, len(rnd)))
12768: (0)          def JM_ensure_ocproperties(pdf):
12769: (4)              '''
12770: (4)              Ensure OCProperties, return /OCProperties key
12771: (4)              '''
12772: (4)              ocp = mupdf.pdf_dict_get(mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf),
PDF_NAME('Root')), PDF_NAME('OCProperties'))
12773: (4)              if ocp.m_internal:
12774: (8)                  return ocp
12775: (4)              root = mupdf.pdf_dict_get(mupdf.pdf_trailer(pdf), PDF_NAME('Root'))
12776: (4)              ocp = mupdf.pdf_dict_put_dict(root, PDF_NAME('OCProperties'), 2)
12777: (4)              mupdf.pdf_dict_put_array(ocp, PDF_NAME('OCGs'), 0)
12778: (4)              D = mupdf.pdf_dict_put_dict(ocp, PDF_NAME('D'), 5)
12779: (4)              mupdf.pdf_dict_put_array(D, PDF_NAME('ON'), 0)
12780: (4)              mupdf.pdf_dict_put_array(D, PDF_NAME('OFF'), 0)
12781: (4)              mupdf.pdf_dict_put_array(D, PDF_NAME('Order'), 0)
12782: (4)              mupdf.pdf_dict_put_array(D, PDF_NAME('RBGroups'), 0)
12783: (4)              return ocp
12784: (0)          def JM_expand_fname(name):
12785: (4)              '''
12786: (4)              Make /DA string of annotation
12787: (4)              '''
12788: (4)              if not name:    return "Helv"
12789: (4)              if name.startswith("Co"):   return "Cour"
12790: (4)              if name.startswith("co"):   return "Cour"
12791: (4)              if name.startswith("Ti"):   return "TiRo"
12792: (4)              if name.startswith("ti"):   return "TiRo"
12793: (4)              if name.startswith("Sy"):   return "Symb"
12794: (4)              if name.startswith("sy"):   return "Symb"
12795: (4)              if name.startswith("Za"):   return "ZaDb"
12796: (4)              if name.startswith("za"):   return "ZaDb"
12797: (4)              return "Helv"
12798: (0)          def JM_field_type_text(wtype):
12799: (4)              '''
12800: (4)              String from widget type
12801: (4)              '''
12802: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_BUTTON:
12803: (8)                  return "Button"
12804: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_CHECKBOX:
12805: (8)                  return "CheckBox"
12806: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON:
```

```
12807: (8)                  return "RadioButton"
12808: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_TEXT:
12809: (8)                  return "Text"
12810: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_LISTBOX:
12811: (8)                  return "ListBox"
12812: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_COMBOBOX:
12813: (8)                  return "ComboBox"
12814: (4)              if wtype == mupdf.PDF_WIDGET_TYPE_SIGNATURE:
12815: (8)                  return "Signature"
12816: (4)              return "unknown"
12817: (0)          def JM_fill_pixmap_rect_with_color(dest, col, b):
12818: (4)              assert isinstance(dest, mupdf.FzPixmap)
12819: (4)              b = mupdf.fz_intersect_irect(b, mupdf.fz_pixmap_bbox( dest))
12820: (4)              w = b.x1 - b.x0
12821: (4)              y = b.y1 - b.y0
12822: (4)              if w <= 0 or y <= 0:
12823: (8)                  return 0
12824: (4)              destspan = dest.stride()
12825: (4)              destp = destspan * (b.y0 - dest.y()) + dest.n() * (b.x0 - dest.x())
12826: (4)              while 1:
12827: (8)                  s = destp
12828: (8)                  for x in range(w):
12829: (12)                     for i in range( dest.n()):
12830: (16)                         mupdf.fz_samples_set(dest, s, col[i])
12831: (16)                         s += 1
12832: (8)                  destp += destspan
12833: (8)                  y -= 1
12834: (8)                  if y == 0:
12835: (12)                     break
12836: (4)              return 1
12837: (0)          def JM_find_annot_irt(annot):
12838: (4)              '''
12839: (4)              Return the first annotation whose /IRT key ("In Response To") points to
12840: (4)              annot. Used to remove the response chain of a given annotation.
12841: (4)              '''
12842: (4)              assert isinstance(annot, mupdf.PdfAnnot)
12843: (4)              irt_annot = None    # returning this
12844: (4)              annot_obj = mupdf.pdf_annot_obj(annot)
12845: (4)              found = 0
12846: (4)              page = mupdf.pdf_annot_page(annot)
12847: (4)              irt_annot = mupdf.pdf_first_annot(page)
12848: (4)              while 1:
12849: (8)                  assert isinstance(irt_annot, mupdf.PdfAnnot)
12850: (8)                  if not irt_annot.m_internal:
12851: (12)                     break
12852: (8)                  irt_annot_obj = mupdf.pdf_annot_obj(irt_annot)
12853: (8)                  o = mupdf.pdf_dict_gets(irt_annot_obj, 'IRT')
12854: (8)                  if o.m_internal:
12855: (12)                     if not mupdf.pdf_objcmp(o, annot_obj):
12856: (16)                         found = 1
12857: (16)                         break
12858: (8)                  irt_annot = mupdf.pdf_next_annot(irt_annot)
12859: (4)              if found:
12860: (8)                  return irt_annot
12861: (0)          def JM_font_ascender(font):
12862: (4)              '''
12863: (4)              need own versions of ascender / descender
12864: (4)              '''
12865: (4)              assert isinstance(font, mupdf.FzFont)
12866: (4)              if _globals.skip_quad_corrections:
12867: (8)                  return 0.8
12868: (4)              return mupdf.fz_font_ascender(font)
12869: (0)          def JM_font_descender(font):
12870: (4)              '''
12871: (4)              need own versions of ascender / descender
12872: (4)              '''
12873: (4)              assert isinstance(font, mupdf.FzFont)
12874: (4)              if _globals.skip_quad_corrections:
12875: (8)                  return -0.2
```

```
12876: (4)              ret = mupdf.fz_font_descender(font)
12877: (4)              return ret
12878: (0)          def JM_is_word_delimiter(ch, delimiters):
12879: (4)              """Check if ch is an extra word delimiting character.
12880: (4)              """
12881: (4)              if ch <= 32 or ch == 160:  # any whitespace?
12882: (8)                  return True
12883: (4)              if not delimiters:  # no extra delimiters provided
12884: (8)                  return False
12885: (4)              char = chr(ch)
12886: (4)              for d in delimiters:
12887: (8)                  if d == char:
12888: (12)                     return True
12889: (4)              return False
12890: (0)          def JM_font_name(font):
12891: (4)              assert isinstance(font, mupdf.FzFont)
12892: (4)              name = mupdf.fz_font_name(font)
12893: (4)              s = name.find('+')
12894: (4)              if _globals.subset_fontnames or s == -1 or s != 6:
12895: (8)                  return name
12896: (4)              return name[s + 1:]
12897: (0)          def JM_gather_fonts(pdf, dict_, fontlist, stream_xref):
12898: (4)              rc = 1
12899: (4)              n = mupdf.pdf_dict_len(dict_)
12900: (4)              for i in range(n):
12901: (8)                  refname = mupdf.pdf_dict_get_key(dict_, i)
12902: (8)                  fontdict = mupdf.pdf_dict_get_val(dict_, i)
12903: (8)                  if not mupdf.pdf_is_dict(fontdict):
12904: (12)                     mupdf.fz_warn( f"'{mupdf.pdf_to_name(refname)}' is no font dict
({mupdf.pdf_to_num(fontdict)} 0 R)")
12905: (12)                     continue
12906: (8)                  subtype = mupdf.pdf_dict_get(fontdict, mupdf.PDF_ENUM_NAME_Subtype)
12907: (8)                  basefont = mupdf.pdf_dict_get(fontdict, mupdf.PDF_ENUM_NAME_BaseFont)
12908: (8)                  if not basefont.m_internal or mupdf.pdf_is_null(basefont):
12909: (12)                     name = mupdf.pdf_dict_get(fontdict, mupdf.PDF_ENUM_NAME_Name)
12910: (8)                  else:
12911: (12)                     name = basefont
12912: (8)                  encoding = mupdf.pdf_dict_get(fontdict, mupdf.PDF_ENUM_NAME_Encoding)
12913: (8)                  if mupdf.pdf_is_dict(encoding):
12914: (12)                     encoding = mupdf.pdf_dict_get(encoding,
mupdf.PDF_ENUM_NAME_BaseEncoding)
12915: (8)                  xref = mupdf.pdf_to_num(fontdict)
12916: (8)                  ext = "n/a"
12917: (8)                  if xref:
12918: (12)                     ext = JM_get_fontextension(pdf, xref)
12919: (8)                  entry = (
12920: (16)                         xref,
12921: (16)                         ext,
12922: (16)                         mupdf.pdf_to_name(subtype),
12923: (16)                         JM_EscapeStrFromStr(mupdf.pdf_to_name(name)),
12924: (16)                         mupdf.pdf_to_name(refname),
12925: (16)                         mupdf.pdf_to_name(encoding),
12926: (16)                         stream_xref,
12927: (16)                         )
12928: (8)                  fontlist.append(entry)
12929: (4)              return rc
12930: (0)          def JM_gather_forms(doc, dict_: mupdf.PdfObj, imagelist, stream_xref: int):
12931: (4)              '''
12932: (4)              Store info of a /Form xobject in Python list
12933: (4)              '''
12934: (4)              assert isinstance(doc, mupdf.PdfDocument)
12935: (4)              rc = 1
12936: (4)              n = mupdf.pdf_dict_len(dict_)
12937: (4)              for i in range(n):
12938: (8)                  refname = mupdf.pdf_dict_get_key( dict_, i)
12939: (8)                  imagedict = mupdf.pdf_dict_get_val(dict_, i)
12940: (8)                  if not mupdf.pdf_is_dict(imagedict):
12941: (12)                     mupdf.fz_warn( f"'{mupdf.pdf_to_name(refname)}' is no form dict
({mupdf.pdf_to_num(imagedict)} 0 R)")
```

```
12942: (12)                     continue
12943: (8)                  type_ = mupdf.pdf_dict_get(imagedict, PDF_NAME('Subtype'))
12944: (8)                  if not mupdf.pdf_name_eq(type_, PDF_NAME('Form')):
12945: (12)                     continue
12946: (8)                  o = mupdf.pdf_dict_get(imagedict, PDF_NAME('BBox'))
12947: (8)                  m = mupdf.pdf_dict_get(imagedict, PDF_NAME('Matrix'))
12948: (8)                  if m.m_internal:
12949: (12)                     mat = mupdf.pdf_to_matrix(m)
12950: (8)                  else:
12951: (12)                     mat = mupdf.FzMatrix()
12952: (8)                  if o.m_internal:
12953: (12)                     bbox = mupdf.fz_transform_rect( mupdf.pdf_to_rect(o), mat)
12954: (8)                  else:
12955: (12)                     bbox = mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE)
12956: (8)                  xref = mupdf.pdf_to_num(imagedict)
12957: (8)                  entry = (
12958: (16)                         xref,
12959: (16)                         mupdf.pdf_to_name( refname),
12960: (16)                         stream_xref,
12961: (16)                         JM_py_from_rect(bbox),
12962: (16)                         )
12963: (8)                  imagelist.append(entry)
12964: (4)              return rc
12965: (0)          def JM_gather_images(doc: mupdf.PdfDocument, dict_: mupdf.PdfObj, imagelist,
stream_xref: int):
12966: (4)              '''
12967: (4)              Store info of an image in Python list
12968: (4)              '''
12969: (4)              rc = 1
12970: (4)              n = mupdf.pdf_dict_len( dict_)
12971: (4)              for i in range(n):
12972: (8)                  refname = mupdf.pdf_dict_get_key(dict_, i)
12973: (8)                  imagedict = mupdf.pdf_dict_get_val(dict_, i)
12974: (8)                  if not mupdf.pdf_is_dict(imagedict):
12975: (12)                     mupdf.fz_warn(f"'{mupdf.pdf_to_name(refname)}' is no image dict
({mupdf.pdf_to_num(imagedict)} 0 R)")
12976: (12)                     continue
12977: (8)                  type_ = mupdf.pdf_dict_get(imagedict, PDF_NAME('Subtype'))
12978: (8)                  if not mupdf.pdf_name_eq(type_, PDF_NAME('Image')):
12979: (12)                     continue
12980: (8)                  xref = mupdf.pdf_to_num(imagedict)
12981: (8)                  gen = 0
12982: (8)                  smask = mupdf.pdf_dict_geta(imagedict, PDF_NAME('SMask'),
PDF_NAME('Mask'))
12983: (8)                  if smask.m_internal:
12984: (12)                     gen = mupdf.pdf_to_num(smask)
12985: (8)                  filter_ = mupdf.pdf_dict_geta(imagedict, PDF_NAME('Filter'),
PDF_NAME('F'))
12986: (8)                  if mupdf.pdf_is_array(filter_):
12987: (12)                     filter_ = mupdf.pdf_array_get(filter_, 0)
12988: (8)                  altcs = mupdf.PdfObj(0)
12989: (8)                  cs = mupdf.pdf_dict_geta(imagedict, PDF_NAME('ColorSpace'),
PDF_NAME('CS'))
12990: (8)                  if mupdf.pdf_is_array(cs):
12991: (12)                     cses = cs
12992: (12)                     cs = mupdf.pdf_array_get(cses, 0)
12993: (12)                     if (mupdf.pdf_name_eq(cs, PDF_NAME('DeviceN'))
12994: (20)                             or mupdf.pdf_name_eq(cs, PDF_NAME('Separation'))
12995: (20)                             ):
12996: (16)                         altcs = mupdf.pdf_array_get(cses, 2)
12997: (16)                         if mupdf.pdf_is_array(altcs):
12998: (20)                             altcs = mupdf.pdf_array_get(altcs, 0)
12999: (8)                  width = mupdf.pdf_dict_geta(imagedict, PDF_NAME('Width'),
PDF_NAME('W'))
13000: (8)                  height = mupdf.pdf_dict_geta(imagedict, PDF_NAME('Height'),
PDF_NAME('H'))
13001: (8)                  bpc = mupdf.pdf_dict_geta(imagedict, PDF_NAME('BitsPerComponent'),
PDF_NAME('BPC'))
13002: (8)                  entry = (
```

```
13003: (16)                         xref,
13004: (16)                         gen,
13005: (16)                         mupdf.pdf_to_int(width),
13006: (16)                         mupdf.pdf_to_int(height),
13007: (16)                         mupdf.pdf_to_int(bpc),
13008: (16)                         JM_EscapeStrFromStr(mupdf.pdf_to_name(cs)),
13009: (16)                         JM_EscapeStrFromStr(mupdf.pdf_to_name(altcs)),
13010: (16)                         JM_EscapeStrFromStr(mupdf.pdf_to_name(refname)),
13011: (16)                         JM_EscapeStrFromStr(mupdf.pdf_to_name(filter_)),
13012: (16)                         stream_xref,
13013: (16)                         )
13014: (8)                  imagelist.append(entry)
13015: (4)              return rc
13016: (0)          def JM_get_annot_by_xref(page, xref):
13017: (4)              '''
13018: (4)              retrieve annot by its xref
13019: (4)              '''
13020: (4)              assert isinstance(page, mupdf.PdfPage)
13021: (4)              found = 0
13022: (4)              annot = mupdf.pdf_first_annot(page)
13023: (4)              while 1:
13024: (8)                  if not annot.m_internal:
13025: (12)                     break
13026: (8)                  if xref == mupdf.pdf_to_num(mupdf.pdf_annot_obj(annot)):
13027: (12)                     found = 1
13028: (12)                     break
13029: (8)                  annot = mupdf.pdf_next_annot( annot)
13030: (4)              if not found:
13031: (8)                  raise Exception("xref %d is not an annot of this page" % xref)
13032: (4)              return annot
13033: (0)          def JM_get_annot_by_name(page, name):
13034: (4)              '''
13035: (4)              retrieve annot by name (/NM key)
13036: (4)              '''
13037: (4)              assert isinstance(page, mupdf.PdfPage)
13038: (4)              if not name:
13039: (8)                  return
13040: (4)              found = 0
13041: (4)              annot = mupdf.pdf_first_annot(page)
13042: (4)              while 1:
13043: (8)                  if not annot.m_internal:
13044: (12)                     break
13045: (8)                  response, len_ =
mupdf.pdf_to_string(mupdf.pdf_dict_gets(mupdf.pdf_annot_obj(annot), "NM"))
13046: (8)                  if name == response:
13047: (12)                     found = 1
13048: (12)                     break
13049: (8)                  annot = mupdf.pdf_next_annot(annot)
13050: (4)              if not found:
13051: (8)                  raise Exception("'%s' is not an annot of this page" % name)
13052: (4)              return annot
13053: (0)          def JM_get_annot_id_list(page):
13054: (4)              names = []
13055: (4)              annots = mupdf.pdf_dict_get( page.obj(), mupdf.PDF_ENUM_NAME_Annots)
13056: (4)              if not annots.m_internal:
13057: (8)                  return names
13058: (4)              for i in range( mupdf.pdf_array_len(annots)):
13059: (8)                  annot_obj = mupdf.pdf_array_get(annots, i)
13060: (8)                  name = mupdf.pdf_dict_gets(annot_obj, "NM")
13061: (8)                  if name.m_internal:
13062: (12)                     names.append(
13063: (16)                         mupdf.pdf_to_text_string(name)
13064: (16)                         )
13065: (4)              return names
13066: (0)          def JM_get_annot_xref_list( page_obj):
13067: (4)              '''
13068: (4)              return the xrefs and /NM ids of a page's annots, links and fields
13069: (4)              '''
13070: (4)              if g_use_extra:
```

```
13071: (8)                  names = extra.JM_get_annot_xref_list( page_obj)
13072: (8)                  return names
13073: (4)              names = []
13074: (4)              annots = mupdf.pdf_dict_get( page_obj, PDF_NAME('Annots'))
13075: (4)              n = mupdf.pdf_array_len( annots)
13076: (4)              for i in range( n):
13077: (8)                  annot_obj = mupdf.pdf_array_get( annots, i)
13078: (8)                  xref = mupdf.pdf_to_num( annot_obj)
13079: (8)                  subtype = mupdf.pdf_dict_get( annot_obj, PDF_NAME('Subtype'))
13080: (8)                  if not subtype.m_internal:
13081: (12)                     continue    # subtype is required
13082: (8)                  type_ = mupdf.pdf_annot_type_from_string( mupdf.pdf_to_name( subtype))
13083: (8)                  if type_ == mupdf.PDF_ANNOT_UNKNOWN:
13084: (12)                     continue    # only accept valid annot types
13085: (8)                  id_ = mupdf.pdf_dict_gets( annot_obj, "NM")
13086: (8)                  names.append( (xref, type_, mupdf.pdf_to_text_string( id_)))
13087: (4)              return names
13088: (0)          def JM_get_annot_xref_list2(page):
13089: (4)              page = page._pdf_page(required=False)
13090: (4)              if not page.m_internal:
13091: (8)                  return list()
13092: (4)              return JM_get_annot_xref_list( page.obj())
13093: (0)          def JM_get_border_style(style):
13094: (4)              '''
13095: (4)              return pdf_obj "border style" from Python str
13096: (4)              '''
13097: (4)              val = mupdf.PDF_ENUM_NAME_S
13098: (4)              if style is None:
13099: (8)                  return val
13100: (4)              s = style
13101: (4)              if   s.startswith("b") or s.startswith("B"):    val =
mupdf.PDF_ENUM_NAME_B
13102: (4)              elif s.startswith("d") or s.startswith("D"):    val =
mupdf.PDF_ENUM_NAME_D
13103: (4)              elif s.startswith("i") or s.startswith("I"):    val =
mupdf.PDF_ENUM_NAME_I
13104: (4)              elif s.startswith("u") or s.startswith("U"):    val =
mupdf.PDF_ENUM_NAME_U
13105: (4)              elif s.startswith("s") or s.startswith("S"):    val =
mupdf.PDF_ENUM_NAME_S
13106: (4)              return val
13107: (0)          def JM_get_font(
13108: (8)                  fontname,
13109: (8)                  fontfile,
13110: (8)                  fontbuffer,
13111: (8)                  script,
13112: (8)                  lang,
13113: (8)                  ordering,
13114: (8)                  is_bold,
13115: (8)                  is_italic,
13116: (8)                  is_serif,
13117: (8)                  embed,
13118: (8)                  ):
13119: (4)              '''
13120: (4)              return a fz_font from a number of parameters
13121: (4)              '''
13122: (4)              def fertig(font):
13123: (8)                  if not font.m_internal:
13124: (12)                     raise RuntimeError(MSG_FONT_FAILED)
13125: (8)                  if not font.m_internal.flags.never_embed:
13126: (12)                     mupdf.fz_set_font_embedding(font, embed)
13127: (8)                  return font
13128: (4)              index = 0
13129: (4)              font = None
13130: (4)              if fontfile:
13131: (8)                  font = mupdf.fz_new_font_from_file( None, fontfile, index, 0)
13132: (8)                  return fertig(font)
13133: (4)              if fontbuffer:
13134: (8)                  res = JM_BufferFromBytes(fontbuffer)
```

```
13135: (8)                  font = mupdf.fz_new_font_from_buffer( None, res, index, 0)
13136: (8)                  return fertig(font)
13137: (4)              if ordering > -1:
13138: (8)                  font = mupdf.fz_new_cjk_font(ordering)
13139: (8)                  return fertig(font)
13140: (4)              if fontname:
13141: (8)                  font = mupdf.fz_new_base14_font(fontname)
13142: (8)                  if font.m_internal:
13143: (12)                     return fertig(font)
13144: (8)                  font = mupdf.fz_new_builtin_font(fontname, is_bold, is_italic)
13145: (8)                  return fertig(font)
13146: (4)              data, size, index = mupdf.fz_lookup_noto_font( script, lang)
13147: (4)              font = None
13148: (4)              if data:
13149: (8)                  font = mupdf.fz_new_font_from_memory( None, data, size, index, 0)
13150: (4)              if font.m_internal:
13151: (8)                  return fertig(font)
13152: (4)              font = mupdf.fz_load_fallback_font( script, lang, is_serif, is_bold,
is_italic)
13153: (4)              return fertig(font)
13154: (0)          def JM_get_fontbuffer(doc, xref):
13155: (4)              '''
13156: (4)              Return the contents of a font file, identified by xref
13157: (4)              '''
13158: (4)              if xref < 1:
13159: (8)                  return
13160: (4)              o = mupdf.pdf_load_object(doc, xref)
13161: (4)              desft = mupdf.pdf_dict_get(o, PDF_NAME('DescendantFonts'))
13162: (4)              if desft.m_internal:
13163: (8)                  obj = mupdf.pdf_resolve_indirect(mupdf.pdf_array_get(desft, 0))
13164: (8)                  obj = mupdf.pdf_dict_get(obj, PDF_NAME('FontDescriptor'))
13165: (4)              else:
13166: (8)                  obj = mupdf.pdf_dict_get(o, PDF_NAME('FontDescriptor'))
13167: (4)              if not obj.m_internal:
13168: (8)                  message(f"invalid font - FontDescriptor missing")
13169: (8)                  return
13170: (4)              o = obj
13171: (4)              stream = None
13172: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile'))
13173: (4)              if obj.m_internal:
13174: (8)                  stream = obj    # ext = "pfa"
13175: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile2'))
13176: (4)              if obj.m_internal:
13177: (8)                  stream = obj    # ext = "ttf"
13178: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile3'))
13179: (4)              if obj.m_internal:
13180: (8)                  stream = obj
13181: (8)                  obj = mupdf.pdf_dict_get(obj, PDF_NAME('Subtype'))
13182: (8)                  if obj.m_internal and not mupdf.pdf_is_name(obj):
13183: (12)                     message("invalid font descriptor subtype")
13184: (12)                     return
13185: (8)                  if mupdf.pdf_name_eq(obj, PDF_NAME('Type1C')):
13186: (12)                     pass    # Prev code did: ext = "cff", but this has no effect.
13187: (8)                  elif mupdf.pdf_name_eq(obj, PDF_NAME('CIDFontType0C')):
13188: (12)                     pass    # Prev code did: ext = "cid", but this has no effect.
13189: (8)                  elif mupdf.pdf_name_eq(obj, PDF_NAME('OpenType')):
13190: (12)                     pass    # Prev code did: ext = "otf", but this has no effect. */
13191: (8)                  else:
13192: (12)                     message('warning: unhandled font type {pdf_to_name(ctx, obj)!r}')
13193: (4)              if not stream:
13194: (8)                  message('warning: unhandled font type')
13195: (8)                  return
13196: (4)              return mupdf.pdf_load_stream(stream)
13197: (0)          def JM_get_resource_properties(ref):
13198: (4)              '''
13199: (4)              Return the items of Resources/Properties (used for Marked Content)
13200: (4)              Argument may be e.g. a page object or a Form XObject
13201: (4)              '''
13202: (4)              properties = mupdf.pdf_dict_getl(ref, PDF_NAME('Resources'),
```

```
PDF_NAME('Properties'))
13203: (4)              if not properties.m_internal:
13204: (8)                  return ()
13205: (4)              else:
13206: (8)                  n = mupdf.pdf_dict_len(properties)
13207: (8)                  if n < 1:
13208: (12)                     return ()
13209: (8)                  rc = []
13210: (8)                  for i in range(n):
13211: (12)                     key = mupdf.pdf_dict_get_key(properties, i)
13212: (12)                     val = mupdf.pdf_dict_get_val(properties, i)
13213: (12)                     c = mupdf.pdf_to_name(key)
13214: (12)                     xref = mupdf.pdf_to_num(val)
13215: (12)                     rc.append((c, xref))
13216: (4)              return rc
13217: (0)          def JM_get_widget_by_xref( page, xref):
13218: (4)              '''
13219: (4)              retrieve widget by its xref
13220: (4)              '''
13221: (4)              found = False
13222: (4)              annot = mupdf.pdf_first_widget( page)
13223: (4)              while annot.m_internal:
13224: (8)                  annot_obj = mupdf.pdf_annot_obj( annot)
13225: (8)                  if xref == mupdf.pdf_to_num( annot_obj):
13226: (12)                     found = True
13227: (12)                     break
13228: (8)                  annot = mupdf.pdf_next_widget( annot)
13229: (4)              if not found:
13230: (8)                  raise Exception( f"xref {xref} is not a widget of this page")
13231: (4)              return Annot( annot)
13232: (0)          def JM_get_widget_properties(annot, Widget):
13233: (4)              '''
13234: (4)              Populate a Python Widget object with the values from a PDF form field.
13235: (4)              Called by "Page.first_widget" and "Widget.next".
13236: (4)              '''
13237: (4)              annot_obj = mupdf.pdf_annot_obj(annot.this)
13238: (4)              page = mupdf.pdf_annot_page(annot.this)
13239: (4)              pdf = page.doc()
13240: (4)              tw = annot
13241: (4)              def SETATTR(key, value):
13242: (8)                  setattr(Widget, key, value)
13243: (4)              def SETATTR_DROP(mod, key, value):
13244: (8)                  setattr(mod, key, value)
13245: (4)              field_type = mupdf.pdf_widget_type(tw.this)
13246: (4)              Widget.field_type = field_type
13247: (4)              if field_type == mupdf.PDF_WIDGET_TYPE_SIGNATURE:
13248: (8)                  if mupdf.pdf_signature_is_signed(pdf, annot_obj):
13249: (12)                     SETATTR("is_signed", True)
13250: (8)                  else:
13251: (12)                     SETATTR("is_signed",False)
13252: (4)              else:
13253: (8)                  SETATTR("is_signed", None)
13254: (4)              SETATTR_DROP(Widget, "border_style",
JM_UnicodeFromStr(mupdf.pdf_field_border_style(annot_obj)))
13255: (4)              SETATTR_DROP(Widget, "field_type_string",
JM_UnicodeFromStr(JM_field_type_text(field_type)))
13256: (4)              field_name = mupdf.pdf_load_field_name(annot_obj)
13257: (4)              SETATTR_DROP(Widget, "field_name", field_name)
13258: (4)              obj = mupdf.pdf_dict_get(annot_obj, PDF_NAME('TU'))
13259: (4)              if obj.m_internal:
13260: (8)                  label = mupdf.pdf_to_text_string(obj)
13261: (8)                  SETATTR_DROP(Widget, "field_label", label)
13262: (4)              fvalue = None
13263: (4)              if field_type == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON:
13264: (8)                  obj = mupdf.pdf_dict_get( annot_obj, PDF_NAME('Parent'))    # owning
RB group
13265: (8)                  if obj.m_internal:
13266: (12)                     SETATTR_DROP(Widget, "rb_parent", mupdf.pdf_to_num( obj))
13267: (8)                  obj = mupdf.pdf_dict_get(annot_obj, PDF_NAME('AS'))
```

```
13268: (8)                  if obj.m_internal:
13269: (12)                     fvalue = mupdf.pdf_to_name(obj)
13270: (4)              if not fvalue:
13271: (8)                  fvalue = mupdf.pdf_field_value(annot_obj)
13272: (4)              SETATTR_DROP(Widget, "field_value", JM_UnicodeFromStr(fvalue))
13273: (4)              SETATTR_DROP(Widget, "field_display", mupdf.pdf_field_display(annot_obj))
13274: (4)              border_width = mupdf.pdf_to_real(mupdf.pdf_dict_getl(annot_obj,
PDF_NAME('BS'), PDF_NAME('W')))
13275: (4)              if border_width == 0:
13276: (8)                  border_width = 1
13277: (4)              SETATTR_DROP(Widget, "border_width", border_width)
13278: (4)              obj = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('BS'), PDF_NAME('D'))
13279: (4)              if mupdf.pdf_is_array(obj):
13280: (8)                  n = mupdf.pdf_array_len(obj)
13281: (8)                  d = [0] * n
13282: (8)                  for i in range(n):
13283: (12)                     d[i] = mupdf.pdf_to_int(mupdf.pdf_array_get(obj, i))
13284: (8)                  SETATTR_DROP(Widget, "border_dashes", d)
13285: (4)              SETATTR_DROP(Widget, "text_maxlen",
mupdf.pdf_text_widget_max_len(tw.this))
13286: (4)              SETATTR_DROP(Widget, "text_format", mupdf.pdf_text_widget_format(tw.this))
13287: (4)              obj = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('MK'), PDF_NAME('BG'))
13288: (4)              if mupdf.pdf_is_array(obj):
13289: (8)                  n = mupdf.pdf_array_len(obj)
13290: (8)                  col = [0] * n
13291: (8)                  for i in range(n):
13292: (12)                     col[i] = mupdf.pdf_to_real(mupdf.pdf_array_get(obj, i))
13293: (8)                  SETATTR_DROP(Widget, "fill_color", col)
13294: (4)              obj = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('MK'), PDF_NAME('BC'))
13295: (4)              if mupdf.pdf_is_array(obj):
13296: (8)                  n = mupdf.pdf_array_len(obj)
13297: (8)                  col = [0] * n
13298: (8)                  for i in range(n):
13299: (12)                     col[i] = mupdf.pdf_to_real(mupdf.pdf_array_get(obj, i))
13300: (8)                  SETATTR_DROP(Widget, "border_color", col)
13301: (4)              SETATTR_DROP(Widget, "choice_values", JM_choice_options(annot))
13302: (4)              da = mupdf.pdf_to_text_string(mupdf.pdf_dict_get_inheritable(annot_obj,
PDF_NAME('DA')))
13303: (4)              SETATTR_DROP(Widget, "_text_da", JM_UnicodeFromStr(da))
13304: (4)              obj = mupdf.pdf_dict_getl(annot_obj, PDF_NAME('MK'), PDF_NAME('CA'))
13305: (4)              if obj.m_internal:
13306: (8)                  SETATTR_DROP(Widget, "button_caption",
JM_UnicodeFromStr(mupdf.pdf_to_text_string(obj)))
13307: (4)              SETATTR_DROP(Widget, "field_flags", mupdf.pdf_field_flags(annot_obj))
13308: (4)              Widget._parse_da()
13309: (4)              s = mupdf.pdf_dict_get(annot_obj, PDF_NAME('A'))
13310: (4)              ss = JM_get_script(s)
13311: (4)              SETATTR_DROP(Widget, "script", ss)
13312: (4)              SETATTR_DROP(Widget, "script_stroke",
13313: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
PDF_NAME('K')))
13314: (12)                     )
13315: (4)              SETATTR_DROP(Widget, "script_format",
13316: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
PDF_NAME('F')))
13317: (12)                     )
13318: (4)              SETATTR_DROP(Widget, "script_change",
13319: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
PDF_NAME('V')))
13320: (12)                     )
13321: (4)              SETATTR_DROP(Widget, "script_calc",
13322: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
PDF_NAME('C')))
13323: (12)                     )
13324: (4)              SETATTR_DROP(Widget, "script_blur",
13325: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
mupdf.pdf_new_name('Bl')))
13326: (12)                     )
13327: (4)              SETATTR_DROP(Widget, "script_focus",
```

```
13328: (12)                     JM_get_script(mupdf.pdf_dict_getl(annot_obj, PDF_NAME('AA'),
mupdf.pdf_new_name('Fo')))
13329: (12)                     )
13330: (0)          def JM_get_fontextension(doc, xref):
13331: (4)              '''
13332: (4)              Return the file extension of a font file, identified by xref
13333: (4)              '''
13334: (4)              if xref < 1:
13335: (8)                  return "n/a"
13336: (4)              o = mupdf.pdf_load_object(doc, xref)
13337: (4)              desft = mupdf.pdf_dict_get(o, PDF_NAME('DescendantFonts'))
13338: (4)              if desft.m_internal:
13339: (8)                  obj = mupdf.pdf_resolve_indirect(mupdf.pdf_array_get(desft, 0))
13340: (8)                  obj = mupdf.pdf_dict_get(obj, PDF_NAME('FontDescriptor'))
13341: (4)              else:
13342: (8)                  obj = mupdf.pdf_dict_get(o, PDF_NAME('FontDescriptor'))
13343: (4)              if not obj.m_internal:
13344: (8)                  return "n/a"    # this is a base-14 font
13345: (4)              o = obj # we have the FontDescriptor
13346: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile'))
13347: (4)              if obj.m_internal:
13348: (8)                  return "pfa"
13349: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile2'))
13350: (4)              if obj.m_internal:
13351: (8)                  return "ttf"
13352: (4)              obj = mupdf.pdf_dict_get(o, PDF_NAME('FontFile3'))
13353: (4)              if obj.m_internal:
13354: (8)                  obj = mupdf.pdf_dict_get(obj, PDF_NAME('Subtype'))
13355: (8)                  if obj.m_internal and not mupdf.pdf_is_name(obj):
13356: (12)                     message("invalid font descriptor subtype")
13357: (12)                     return "n/a"
13358: (8)                  if mupdf.pdf_name_eq(obj, PDF_NAME('Type1C')):
13359: (12)                     return "cff"
13360: (8)                  elif mupdf.pdf_name_eq(obj, PDF_NAME('CIDFontType0C')):
13361: (12)                     return "cid"
13362: (8)                  elif mupdf.pdf_name_eq(obj, PDF_NAME('OpenType')):
13363: (12)                     return "otf"
13364: (8)                  else:
13365: (12)                     message("unhandled font type '%s'", mupdf.pdf_to_name(obj))
13366: (4)              return "n/a"
13367: (0)          def JM_get_ocg_arrays_imp(arr):
13368: (4)              '''
13369: (4)              Get OCG arrays from OC configuration
13370: (4)              Returns dict {"basestate":name, "on":list, "off":list, "rbg":list,
"locked":list}
13371: (4)              '''
13372: (4)              list_ = list()
13373: (4)              if mupdf.pdf_is_array( arr):
13374: (8)                  n = mupdf.pdf_array_len( arr)
13375: (8)                  for i in range(n):
13376: (12)                     obj = mupdf.pdf_array_get( arr, i)
13377: (12)                     item = mupdf.pdf_to_num( obj)
13378: (12)                     if item not in list_:
13379: (16)                         list_.append(item)
13380: (4)              return list_
13381: (0)          def JM_get_ocg_arrays(conf):
13382: (4)              rc = dict()
13383: (4)              arr = mupdf.pdf_dict_get( conf, PDF_NAME('ON'))
13384: (4)              list_ = JM_get_ocg_arrays_imp( arr)
13385: (4)              if list_:
13386: (8)                  rc["on"] = list_
13387: (4)              arr = mupdf.pdf_dict_get( conf, PDF_NAME('OFF'))
13388: (4)              list_ = JM_get_ocg_arrays_imp( arr)
13389: (4)              if list_:
13390: (8)                  rc["off"] = list_
13391: (4)              arr = mupdf.pdf_dict_get( conf, PDF_NAME('Locked'))
13392: (4)              list_ = JM_get_ocg_arrays_imp( arr)
13393: (4)              if list_:
13394: (8)                  rc['locked'] = list_
```

```
13395: (4)              list_ = list()
13396: (4)              arr = mupdf.pdf_dict_get( conf, PDF_NAME('RBGroups'))
13397: (4)              if mupdf.pdf_is_array( arr):
13398: (8)                  n = mupdf.pdf_array_len( arr)
13399: (8)                  for i in range(n):
13400: (12)                     obj = mupdf.pdf_array_get( arr, i)
13401: (12)                     list1 = JM_get_ocg_arrays_imp( obj)
13402: (12)                     list_.append(list1)
13403: (4)              if list_:
13404: (8)                  rc["rbgroups"] = list_
13405: (4)              obj = mupdf.pdf_dict_get( conf, PDF_NAME('BaseState'))
13406: (4)              if obj.m_internal:
13407: (8)                  state = mupdf.pdf_to_name( obj)
13408: (8)                  rc["basestate"] = state
13409: (4)              return rc
13410: (0)          def JM_get_page_labels(liste, nums):
13411: (4)              n = mupdf.pdf_array_len(nums)
13412: (4)              for i in range(0, n, 2):
13413: (8)                  key = mupdf.pdf_resolve_indirect( mupdf.pdf_array_get(nums, i))
13414: (8)                  pno = mupdf.pdf_to_int(key)
13415: (8)                  val = mupdf.pdf_resolve_indirect( mupdf.pdf_array_get(nums, i + 1))
13416: (8)                  res = JM_object_to_buffer(val, 1, 0)
13417: (8)                  c = mupdf.fz_buffer_extract(res)
13418: (8)                  assert isinstance(c, bytes)
13419: (8)                  c = c.decode('utf-8')
13420: (8)                  liste.append( (pno, c))
13421: (0)          def JM_get_script(key):
13422: (4)              '''
13423: (4)              JavaScript extractor
13424: (4)              Returns either the script source or None. Parameter is a PDF action
13425: (4)              dictionary, which must have keys /S and /JS. The value of /S must be
13426: (4)              '/JavaScript'. The value of /JS is returned.
13427: (4)              '''
13428: (4)              if not key.m_internal:
13429: (8)                  return
13430: (4)              j = mupdf.pdf_dict_get(key, PDF_NAME('S'))
13431: (4)              jj = mupdf.pdf_to_name(j)
13432: (4)              if jj == "JavaScript":
13433: (8)                  js = mupdf.pdf_dict_get(key, PDF_NAME('JS'))
13434: (8)                  if not js.m_internal:
13435: (12)                     return
13436: (4)              else:
13437: (8)                  return
13438: (4)              if mupdf.pdf_is_string(js):
13439: (8)                  script = JM_UnicodeFromStr(mupdf.pdf_to_text_string(js))
13440: (4)              elif mupdf.pdf_is_stream(js):
13441: (8)                  res = mupdf.pdf_load_stream(js)
13442: (8)                  script = JM_EscapeStrFromBuffer(res)
13443: (4)              else:
13444: (8)                  return
13445: (4)              if script:  # do not return an empty script
13446: (8)                  return script
13447: (4)              return
13448: (0)          def JM_have_operation(pdf):
13449: (4)              '''
13450: (4)              Ensure valid journalling state
13451: (4)              '''
13452: (4)              if pdf.m_internal.journal and not mupdf.pdf_undoredo_step(pdf, 0):
13453: (8)                  return 0
13454: (4)              return 1
13455: (0)          def JM_image_extension(type_):
13456: (4)              '''
13457: (4)              return extension for MuPDF image type
13458: (4)              '''
13459: (4)              if type_ == mupdf.FZ_IMAGE_FAX:     return "fax"
13460: (4)              if type_ == mupdf.FZ_IMAGE_RAW:     return "raw"
13461: (4)              if type_ == mupdf.FZ_IMAGE_FLATE:   return "flate"
13462: (4)              if type_ == mupdf.FZ_IMAGE_LZW:     return "lzw"
13463: (4)              if type_ == mupdf.FZ_IMAGE_RLD:     return "rld"
```

```
13464: (4)              if type_ == mupdf.FZ_IMAGE_BMP:     return "bmp"
13465: (4)              if type_ == mupdf.FZ_IMAGE_GIF:     return "gif"
13466: (4)              if type_ == mupdf.FZ_IMAGE_JBIG2:   return "jb2"
13467: (4)              if type_ == mupdf.FZ_IMAGE_JPEG:    return "jpeg"
13468: (4)              if type_ == mupdf.FZ_IMAGE_JPX:     return "jpx"
13469: (4)              if type_ == mupdf.FZ_IMAGE_JXR:     return "jxr"
13470: (4)              if type_ == mupdf.FZ_IMAGE_PNG:     return "png"
13471: (4)              if type_ == mupdf.FZ_IMAGE_PNM:     return "pnm"
13472: (4)              if type_ == mupdf.FZ_IMAGE_TIFF:    return "tiff"
13473: (4)              return "n/a"
13474: (0)          g_img_info = None
13475: (0)          def JM_image_filter(opaque, ctm, name, image):
13476: (4)              assert isinstance(ctm, mupdf.FzMatrix)
13477: (4)              r = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)
13478: (4)              q = mupdf.fz_transform_quad( mupdf.fz_quad_from_rect(r), ctm)
13479: (4)              q = mupdf.fz_transform_quad( q, g_img_info_matrix)
13480: (4)              temp = name, JM_py_from_quad(q)
13481: (4)              g_img_info.append(temp)
13482: (0)          def JM_image_profile( imagedata, keep_image):
13483: (4)              '''
13484: (4)              Return basic properties of an image provided as bytes or bytearray
13485: (4)              The function creates an fz_image and optionally returns it.
13486: (4)              '''
13487: (4)              if not imagedata:
13488: (8)                  return None # nothing given
13489: (4)              len_ = len( imagedata)
13490: (4)              if len_ < 8:
13491: (8)                  message( "bad image data")
13492: (8)                  return None
13493: (4)              c = imagedata
13494: (4)              type_ = mupdf.fz_recognize_image_format( c)
13495: (4)              if type_ == mupdf.FZ_IMAGE_UNKNOWN:
13496: (8)                  return None
13497: (4)              if keep_image:
13498: (8)                  res = mupdf.fz_new_buffer_from_copied_data( c, len_)
13499: (4)              else:
13500: (8)                  res = mupdf.fz_new_buffer_from_shared_data( c, len_)
13501: (4)              image = mupdf.fz_new_image_from_buffer( res)
13502: (4)              ctm = mupdf.fz_image_orientation_matrix( image)
13503: (4)              xres, yres = mupdf.fz_image_resolution(image)
13504: (4)              orientation = mupdf.fz_image_orientation( image)
13505: (4)              cs_name = mupdf.fz_colorspace_name( image.colorspace())
13506: (4)              result = dict()
13507: (4)              result[ dictkey_width] = image.w()
13508: (4)              result[ dictkey_height] = image.h()
13509: (4)              result[ "orientation"] = orientation
13510: (4)              result[ dictkey_matrix] = JM_py_from_matrix(ctm)
13511: (4)              result[ dictkey_xres] = xres
13512: (4)              result[ dictkey_yres] = yres
13513: (4)              result[ dictkey_colorspace] = image.n()
13514: (4)              result[ dictkey_bpc] = image.bpc()
13515: (4)              result[ dictkey_ext] = JM_image_extension(type_)
13516: (4)              result[ dictkey_cs_name] = cs_name
13517: (4)              if keep_image:
13518: (8)                  result[ dictkey_image] = image
13519: (4)              return result
13520: (0)          def JM_image_reporter(page):
13521: (4)              doc = page.doc()
13522: (4)              global g_img_info_matrix
13523: (4)              g_img_info_matrix = mupdf.FzMatrix()
13524: (4)              mediabox = mupdf.FzRect()
13525: (4)              mupdf.pdf_page_transform(page, mediabox, g_img_info_matrix)
13526: (4)              class SanitizeFilterOptions(mupdf.PdfSanitizeFilterOptions2):
13527: (8)                  def __init__(self):
13528: (12)                     super().__init__()
13529: (12)                     self.use_virtual_image_filter()
13530: (8)                  if mupdf_version_tuple >= (1, 23, 11):
13531: (12)                     def image_filter(self, ctx, ctm, name, image, scissor):
13532: (16)                         JM_image_filter(None, mupdf.FzMatrix(ctm), name, image)
```

```
13533: (8)                  else:
13534: (12)                     def image_filter(self, ctx, ctm, name, image):
13535: (16)                         JM_image_filter(None, mupdf.FzMatrix(ctm), name, image)
13536: (4)              sanitize_filter_options = SanitizeFilterOptions()
13537: (4)              filter_options = _make_PdfFilterOptions(
13538: (12)                     instance_forms=1,
13539: (12)                     ascii=1,
13540: (12)                     no_update=1,
13541: (12)                     sanitize=1,
13542: (12)                     sopts=sanitize_filter_options,
13543: (12)                     )
13544: (4)              global g_img_info
13545: (4)              g_img_info = []
13546: (4)              mupdf.pdf_filter_page_contents( doc, page, filter_options)
13547: (4)              rc = tuple(g_img_info)
13548: (4)              g_img_info = []
13549: (4)              return rc
13550: (0)          def JM_fitz_config():
13551: (4)              have_TOFU           = not hasattr(mupdf, 'TOFU')
13552: (4)              have_TOFU_BASE14    = not hasattr(mupdf, 'TOFU_BASE14')
13553: (4)              have_TOFU_CJK       = not hasattr(mupdf, 'TOFU_CJK')
13554: (4)              have_TOFU_CJK_EXT   = not hasattr(mupdf, 'TOFU_CJK_EXT')
13555: (4)              have_TOFU_CJK_LANG  = not hasattr(mupdf, 'TOFU_CJK_LANG')
13556: (4)              have_TOFU_EMOJI     = not hasattr(mupdf, 'TOFU_EMOJI')
13557: (4)              have_TOFU_HISTORIC  = not hasattr(mupdf, 'TOFU_HISTORIC')
13558: (4)              have_TOFU_SIL       = not hasattr(mupdf, 'TOFU_SIL')
13559: (4)              have_TOFU_SYMBOL    = not hasattr(mupdf, 'TOFU_SYMBOL')
13560: (4)              ret = dict()
13561: (4)              ret["base14"]           = have_TOFU_BASE14
13562: (4)              ret["cbz"]              = bool(mupdf.FZ_ENABLE_CBZ)
13563: (4)              ret["epub"]             = bool(mupdf.FZ_ENABLE_EPUB)
13564: (4)              ret["html"]             = bool(mupdf.FZ_ENABLE_HTML)
13565: (4)              ret["icc"]              = bool(mupdf.FZ_ENABLE_ICC)
13566: (4)              ret["img"]              = bool(mupdf.FZ_ENABLE_IMG)
13567: (4)              ret["jpx"]              = bool(mupdf.FZ_ENABLE_JPX)
13568: (4)              ret["js"]               = bool(mupdf.FZ_ENABLE_JS)
13569: (4)              ret["pdf"]              = bool(mupdf.FZ_ENABLE_PDF)
13570: (4)              ret["plotter-cmyk"]     = bool(mupdf.FZ_PLOTTERS_CMYK)
13571: (4)              ret["plotter-g"]        = bool(mupdf.FZ_PLOTTERS_G)
13572: (4)              ret["plotter-n"]        = bool(mupdf.FZ_PLOTTERS_N)
13573: (4)              ret["plotter-rgb"]      = bool(mupdf.FZ_PLOTTERS_RGB)
13574: (4)              ret["py-memory"]        = bool(JM_MEMORY)
13575: (4)              ret["svg"]              = bool(mupdf.FZ_ENABLE_SVG)
13576: (4)              ret["tofu"]             = have_TOFU
13577: (4)              ret["tofu-cjk"]         = have_TOFU_CJK
13578: (4)              ret["tofu-cjk-ext"]     = have_TOFU_CJK_EXT
13579: (4)              ret["tofu-cjk-lang"]    = have_TOFU_CJK_LANG
13580: (4)              ret["tofu-emoji"]       = have_TOFU_EMOJI
13581: (4)              ret["tofu-historic"]    = have_TOFU_HISTORIC
13582: (4)              ret["tofu-sil"]         = have_TOFU_SIL
13583: (4)              ret["tofu-symbol"]      = have_TOFU_SYMBOL
13584: (4)              ret["xps"]              = bool(mupdf.FZ_ENABLE_XPS)
13585: (4)              return ret
13586: (0)          def JM_insert_contents(pdf, pageref, newcont, overlay):
13587: (4)              '''
13588: (4)              Insert a buffer as a new separate /Contents object of a page.
13589: (4)              1. Create a new stream object from buffer 'newcont'
13590: (4)              2. If /Contents already is an array, then just prepend or append this
object
13591: (4)              3. Else, create new array and put old content obj and this object into it.
13592: (7)                 If the page had no /Contents before, just create a 1-item array.
13593: (4)              '''
13594: (4)              contents = mupdf.pdf_dict_get(pageref, PDF_NAME('Contents'))
13595: (4)              newconts = mupdf.pdf_add_stream(pdf, newcont, mupdf.PdfObj(), 0)
13596: (4)              xref = mupdf.pdf_to_num(newconts)
13597: (4)              if mupdf.pdf_is_array(contents):
13598: (8)                  if overlay:  # append new object
13599: (12)                     mupdf.pdf_array_push(contents, newconts)
13600: (8)                  else:   # prepend new object
```

```
13601: (12)                     mupdf.pdf_array_insert(contents, newconts, 0)
13602: (4)              else:
13603: (8)                  carr = mupdf.pdf_new_array(pdf, 5)
13604: (8)                  if overlay:
13605: (12)                     if contents.m_internal:
13606: (16)                         mupdf.pdf_array_push(carr, contents)
13607: (12)                     mupdf.pdf_array_push(carr, newconts)
13608: (8)                  else:
13609: (12)                     mupdf.pdf_array_push(carr, newconts)
13610: (12)                     if contents.m_internal:
13611: (16)                         mupdf.pdf_array_push(carr, contents)
13612: (8)                  mupdf.pdf_dict_put(pageref, PDF_NAME('Contents'), carr)
13613: (4)              return xref
13614: (0)          def JM_insert_font(pdf, bfname, fontfile, fontbuffer, set_simple, idx, wmode,
serif, encoding, ordering):
13615: (4)              '''
13616: (4)              Insert a font in a PDF
13617: (4)              '''
13618: (4)              font = None
13619: (4)              res = None
13620: (4)              data = None
13621: (4)              ixref = 0
13622: (4)              index = 0
13623: (4)              simple = 0
13624: (4)              value=None
13625: (4)              name=None
13626: (4)              subt=None
13627: (4)              exto = None
13628: (4)              ENSURE_OPERATION(pdf)
13629: (4)              if ordering > -1:
13630: (8)                  data, size, index = mupdf.fz_lookup_cjk_font(ordering)
13631: (4)              if data:
13632: (8)                  font = mupdf.fz_new_font_from_memory(None, data, size, index, 0)
13633: (8)                  font_obj = mupdf.pdf_add_cjk_font(pdf, font, ordering, wmode, serif)
13634: (8)                  exto = "n/a"
13635: (8)                  simple = 0
13636: (4)              else:
13637: (8)                  if bfname:
13638: (12)                     data, size = mupdf.fz_lookup_base14_font(bfname)
13639: (8)                  if data:
13640: (12)                     font = mupdf.fz_new_font_from_memory(bfname, data, size, 0, 0)
13641: (12)                     font_obj = mupdf.pdf_add_simple_font(pdf, font, encoding)
13642: (12)                     exto = "n/a"
13643: (12)                     simple = 1
13644: (8)                  else:
13645: (12)                     if fontfile:
13646: (16)                         font = mupdf.fz_new_font_from_file(None, fontfile, idx, 0)
13647: (12)                     else:
13648: (16)                         res = JM_BufferFromBytes(fontbuffer)
13649: (16)                         if not res.m_internal:
13650: (20)                             RAISEPY(MSG_FILE_OR_BUFFER, PyExc_ValueError)
13651: (16)                         font = mupdf.fz_new_font_from_buffer(None, res, idx, 0)
13652: (12)                     if not set_simple:
13653: (16)                         font_obj = mupdf.pdf_add_cid_font(pdf, font)
13654: (16)                         simple = 0
13655: (12)                     else:
13656: (16)                         font_obj = mupdf.pdf_add_simple_font(pdf, font, encoding)
13657: (16)                         simple = 2
13658: (4)              ixref = mupdf.pdf_to_num(font_obj)
13659: (4)              name = JM_EscapeStrFromStr( mupdf.pdf_to_name(
mupdf.pdf_dict_get(font_obj, PDF_NAME('BaseFont'))))
13660: (4)              subt = JM_UnicodeFromStr( mupdf.pdf_to_name( mupdf.pdf_dict_get( font_obj,
PDF_NAME('Subtype'))))
13661: (4)              if not exto:
13662: (8)                  exto = JM_UnicodeFromStr(JM_get_fontextension(pdf, ixref))
13663: (4)              asc = mupdf.fz_font_ascender(font)
13664: (4)              dsc = mupdf.fz_font_descender(font)
13665: (4)              value = [
13666: (12)                     ixref,
```

```
13667: (12)                     {
13668: (16)                         "name": name,        # base font name
13669: (16)                         "type": subt,        # subtype
13670: (16)                         "ext": exto,         # file extension
13671: (16)                         "simple": bool(simple), # simple font?
13672: (16)                         "ordering": ordering, # CJK font?
13673: (16)                         "ascender": asc,
13674: (16)                         "descender": dsc,
13675: (12)                     },
13676: (12)                     ]
13677: (4)               return value
13678: (0)           def JM_invert_pixmap_rect( dest, b):
13679: (4)               '''
13680: (4)               invert a rectangle - also supports non-alpha pixmaps
13681: (4)               '''
13682: (4)               assert isinstance( dest, mupdf.FzPixmap)
13683: (4)               assert isinstance( b, mupdf.FzIrect)
13684: (4)               b = mupdf.fz_intersect_irect(b, mupdf.fz_pixmap_bbox( dest))
13685: (4)               w = b.x1 - b.x0
13686: (4)               y = b.y1 - b.y0
13687: (4)               if w <= 0 or y <= 0:
13688: (8)                   return 0
13689: (4)               destspan = dest.stride()
13690: (4)               destp = destspan * (b.y0 - dest.y()) + dest.n() * (b.x0 - dest.x())
13691: (4)               n0 = dest.n() - dest.alpha()
13692: (4)               alpha = dest.alpha()
13693: (4)               while 1:
13694: (8)                   s = destp
13695: (8)                   for x in range( w):
13696: (12)                      for i in range( n0):
13697: (16)                          ss = mupdf.fz_samples_get( dest, s)
13698: (16)                          ss = 255 - ss
13699: (16)                          mupdf.fz_samples_set( dest, s, ss)
13700: (16)                          s += 1
13701: (12)                      if alpha:
13702: (16)                          ss = mupdf.fz_samples_get( dest, s)
13703: (16)                          ss += 1
13704: (16)                          mupdf.fz_samples_set( dest, s, ss)
13705: (8)                   destp += destspan
13706: (8)                   y -= 1
13707: (8)                   if y == 0:
13708: (12)                      break
13709: (4)               return 1
13710: (0)           def JM_irect_from_py(r):
13711: (4)               '''
13712: (4)               PySequence to mupdf.FzIrect. Default: infinite irect
13713: (4)               '''
13714: (4)               if isinstance(r, mupdf.FzIrect):
13715: (8)                   return r
13716: (4)               if isinstance(r, IRect):
13717: (8)                   r = mupdf.FzIrect( r.x0, r.y0, r.x1, r.y1)
13718: (8)                   return r
13719: (4)               if isinstance(r, Rect):
13720: (8)                   ret = mupdf.FzRect(r.x0, r.y0, r.x1, r.y1)
13721: (8)                   ret = mupdf.FzIrect(ret)  # Uses fz_irect_from_rect().
13722: (8)                   return ret
13723: (4)               if isinstance(r, mupdf.FzRect):
13724: (8)                   ret = mupdf.FzIrect(r)  # Uses fz_irect_from_rect().
13725: (8)                   return ret
13726: (4)               if not r or not PySequence_Check(r) or PySequence_Size(r) != 4:
13727: (8)                   return mupdf.FzIrect(mupdf.fz_infinite_irect)
13728: (4)               f = [0, 0, 0, 0]
13729: (4)               for i in range(4):
13730: (8)                   f[i] = r[i]
13731: (8)                   if f[i] is None:
13732: (12)                      return mupdf.FzIrect(mupdf.fz_infinite_irect)
13733: (8)                   if f[i] < FZ_MIN_INF_RECT:
13734: (12)                      f[i] = FZ_MIN_INF_RECT
13735: (8)                   if f[i] > FZ_MAX_INF_RECT:
```

```
13736: (12)                     f[i] = FZ_MAX_INF_RECT
13737: (4)              return mupdf.fz_make_irect(f[0], f[1], f[2], f[3])
13738: (0)          def JM_is_jbig2_image(dict_):
13739: (4)              return 0
13740: (0)          def JM_listbox_value( annot):
13741: (4)              '''
13742: (4)              ListBox retrieve value
13743: (4)              '''
13744: (4)              annot_obj = mupdf.pdf_annot_obj( annot)
13745: (4)              optarr = mupdf.pdf_dict_get( annot_obj, PDF_NAME('V'))
13746: (4)              if mupdf.pdf_is_string( optarr):   # a single string
13747: (8)                  return mupdf.pdf_to_text_string( optarr)
13748: (4)              n = mupdf.pdf_array_len( optarr)
13749: (4)              liste = []
13750: (4)              for i in range( n):
13751: (8)                  elem = mupdf.pdf_array_get( optarr, i)
13752: (8)                  if mupdf.pdf_is_array( elem):
13753: (12)                     elem = mupdf.pdf_array_get( elem, 1)
13754: (8)                  liste.append( JM_UnicodeFromStr( mupdf.pdf_to_text_string( elem)))
13755: (4)              return liste
13756: (0)          def JM_make_annot_DA(annot, ncol, col, fontname, fontsize):
13757: (4)              buf = ''
13758: (4)              if ncol < 1:
13759: (8)                  buf += f'0 g '
13760: (4)              elif ncol == 1:
13761: (8)                  buf += f'{col[0]:g} g '
13762: (4)              elif ncol == 2:
13763: (8)                  assert 0
13764: (4)              elif ncol == 3:
13765: (8)                  buf += f'{col[0]:g} {col[1]:g} {col[2]:g} rg '
13766: (4)              else:
13767: (8)                  buf += f'{col[0]:g} {col[1]:g} {col[2]:g} {col[3]:g} k '
13768: (4)              buf += f'/{JM_expand_fname(fontname)} {fontsize} Tf'
13769: (4)              mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(annot),
mupdf.PDF_ENUM_NAME_DA, buf)
13770: (0)          def JM_make_spanlist(line_dict, line, raw, buff, tp_rect):
13771: (4)              if g_use_extra:
13772: (8)                  return extra.JM_make_spanlist(line_dict, line, raw, buff, tp_rect)
13773: (4)              char_list = None
13774: (4)              span_list = []
13775: (4)              mupdf.fz_clear_buffer(buff)
13776: (4)              span_rect = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)
13777: (4)              line_rect = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)
13778: (4)              class char_style:
13779: (8)                  def __init__(self, rhs=None):
13780: (12)                     if rhs:
13781: (16)                         self.size = rhs.size
13782: (16)                         self.flags = rhs.flags
13783: (16)                         self.font = rhs.font
13784: (16)                         self.color = rhs.color
13785: (16)                         self.asc = rhs.asc
13786: (16)                         self.desc = rhs.desc
13787: (12)                     else:
13788: (16)                         self.size = -1
13789: (16)                         self.flags = -1
13790: (16)                         self.font = ''
13791: (16)                         self.color = -1
13792: (16)                         self.asc = 0
13793: (16)                         self.desc = 0
13794: (8)                  def __str__(self):
13795: (12)                     return f'{self.size} {self.flags} {self.font} {self.color}
{self.asc} {self.desc}'
13796: (4)              old_style = char_style()
13797: (4)              style = char_style()
13798: (4)              span = None
13799: (4)              span_origin = None
13800: (4)              for ch in line:
13801: (8)                  r = JM_char_bbox(line, ch)
13802: (8)                  if (not JM_rects_overlap(tp_rect, r)
```

```
13803: (16)                         and not mupdf.fz_is_infinite_rect(tp_rect)
13804: (16)                         ):
13805: (12)                     continue
13806: (8)                  flags =
JM_char_font_flags(mupdf.FzFont(mupdf.ll_fz_keep_font(ch.m_internal.font)), line, ch)
13807: (8)                  origin = mupdf.FzPoint(ch.m_internal.origin)
13808: (8)                  style.size = ch.m_internal.size
13809: (8)                  style.flags = flags
13810: (8)                  style.font =
JM_font_name(mupdf.FzFont(mupdf.ll_fz_keep_font(ch.m_internal.font)))
13811: (8)                  style.color = ch.m_internal.color
13812: (8)                  style.asc =
JM_font_ascender(mupdf.FzFont(mupdf.ll_fz_keep_font(ch.m_internal.font)))
13813: (8)                  style.desc =
JM_font_descender(mupdf.FzFont(mupdf.ll_fz_keep_font(ch.m_internal.font)))
13814: (8)                  if (style.size != old_style.size
13815: (16)                         or style.flags != old_style.flags
13816: (16)                         or style.color != old_style.color
13817: (16)                         or style.font != old_style.font
13818: (16)                         ):
13819: (12)                     if old_style.size >= 0:
13820: (16)                         if raw:
13821: (20)                             span[dictkey_chars] = char_list
13822: (20)                             char_list = None
13823: (16)                         else:
13824: (20)                             span[dictkey_text] = JM_EscapeStrFromBuffer( buff)
13825: (20)                             mupdf.fz_clear_buffer(buff)
13826: (16)                         span[dictkey_origin] = JM_py_from_point(span_origin)
13827: (16)                         span[dictkey_bbox] = JM_py_from_rect(span_rect)
13828: (16)                         line_rect = mupdf.fz_union_rect(line_rect, span_rect)
13829: (16)                         span_list.append( span)
13830: (16)                         span = None
13831: (12)                     span = dict()
13832: (12)                     asc = style.asc
13833: (12)                     desc = style.desc
13834: (12)                     if style.asc < 1e-3:
13835: (16)                         asc = 0.9
13836: (16)                         desc = -0.1
13837: (12)                     span[dictkey_size] = style.size
13838: (12)                     span[dictkey_flags] = style.flags
13839: (12)                     span[dictkey_font] = JM_EscapeStrFromStr(style.font)
13840: (12)                     span[dictkey_color] = style.color
13841: (12)                     span["ascender"] = asc
13842: (12)                     span["descender"] = desc
13843: (12)                     old_style = char_style(style)
13844: (12)                     span_rect = r
13845: (12)                     span_origin = origin
13846: (8)                  span_rect = mupdf.fz_union_rect(span_rect, r)
13847: (8)                  if raw: # make and append a char dict
13848: (12)                     char_dict = dict()
13849: (12)                     char_dict[dictkey_origin] = JM_py_from_point(
ch.m_internal.origin)
13850: (12)                     char_dict[dictkey_bbox] = JM_py_from_rect(r)
13851: (12)                     char_dict[dictkey_c] = chr(ch.m_internal.c)
13852: (12)                     if char_list is None:
13853: (16)                         char_list = []
13854: (12)                     char_list.append(char_dict)
13855: (8)                  else:   # add character byte to buffer
13856: (12)                     JM_append_rune(buff, ch.m_internal.c)
13857: (4)              if span:
13858: (8)                  if raw:
13859: (12)                     span[dictkey_chars] = char_list
13860: (12)                     char_list = None
13861: (8)                  else:
13862: (12)                     span[dictkey_text] = JM_EscapeStrFromBuffer(buff)
13863: (12)                     mupdf.fz_clear_buffer(buff)
13864: (8)                  span[dictkey_origin] = JM_py_from_point(span_origin)
13865: (8)                  span[dictkey_bbox] = JM_py_from_rect(span_rect)
13866: (8)                  if not mupdf.fz_is_empty_rect(span_rect):
```

```
13867: (12)                     span_list.append(span)
13868: (12)                     line_rect = mupdf.fz_union_rect(line_rect, span_rect)
13869: (8)                  span = None
13870: (4)              if not mupdf.fz_is_empty_rect(line_rect):
13871: (8)                  line_dict[dictkey_spans] = span_list
13872: (4)              else:
13873: (8)                  line_dict[dictkey_spans] = span_list
13874: (4)              return line_rect
13875: (0)          def JM_make_image_block(block, block_dict):
13876: (4)              image = block.i_image()
13877: (4)              n = mupdf.fz_colorspace_n(image.colorspace())
13878: (4)              w = image.w()
13879: (4)              h = image.h()
13880: (4)              type_ = mupdf.FZ_IMAGE_UNKNOWN
13881: (4)              ll_fz_compressed_buffer =
mupdf.ll_fz_compressed_image_buffer(image.m_internal)
13882: (4)              if ll_fz_compressed_buffer:
13883: (8)                  type_ = ll_fz_compressed_buffer.params.type
13884: (4)              if type_ < mupdf.FZ_IMAGE_BMP or type_ == mupdf.FZ_IMAGE_JBIG2:
13885: (8)                  type_ = mupdf.FZ_IMAGE_UNKNOWN
13886: (4)              bytes_ = None
13887: (4)              if ll_fz_compressed_buffer and type_ != mupdf.FZ_IMAGE_UNKNOWN:
13888: (8)                  buf = mupdf.FzBuffer( mupdf.ll_fz_keep_buffer(
ll_fz_compressed_buffer.buffer))
13889: (8)                  ext = JM_image_extension(type_)
13890: (4)              else:
13891: (8)                  buf = mupdf.fz_new_buffer_from_image_as_png(image,
mupdf.FzColorParams())
13892: (8)                  ext = "png"
13893: (4)              bytes_ = JM_BinFromBuffer(buf)
13894: (4)              block_dict[ dictkey_width] = w
13895: (4)              block_dict[ dictkey_height] = h
13896: (4)              block_dict[ dictkey_ext] = ext
13897: (4)              block_dict[ dictkey_colorspace] = n
13898: (4)              block_dict[ dictkey_xres] = image.xres()
13899: (4)              block_dict[ dictkey_yres] = image.yres()
13900: (4)              block_dict[ dictkey_bpc] = image.bpc()
13901: (4)              block_dict[ dictkey_matrix] = JM_py_from_matrix(block.i_transform())
13902: (4)              block_dict[ dictkey_size] = len(bytes_)
13903: (4)              block_dict[ dictkey_image] = bytes_
13904: (0)          def JM_make_text_block(block, block_dict, raw, buff, tp_rect):
13905: (4)              if g_use_extra:
13906: (8)                  return extra.JM_make_text_block(block.m_internal, block_dict, raw,
buff.m_internal, tp_rect.m_internal)
13907: (4)              line_list = []
13908: (4)              block_rect = mupdf.FzRect(mupdf.FzRect.Fixed_EMPTY)
13909: (4)              for line in block:
13910: (8)                  if (mupdf.fz_is_empty_rect(mupdf.fz_intersect_rect(tp_rect,
mupdf.FzRect(line.m_internal.bbox)))
13911: (16)                         and not mupdf.fz_is_infinite_rect(tp_rect)
13912: (16)                         ):
13913: (12)                     continue
13914: (8)                  line_dict = dict()
13915: (8)                  line_rect = JM_make_spanlist(line_dict, line, raw, buff, tp_rect)
13916: (8)                  block_rect = mupdf.fz_union_rect(block_rect, line_rect)
13917: (8)                  line_dict[dictkey_wmode] = line.m_internal.wmode
13918: (8)                  line_dict[dictkey_dir] = JM_py_from_point(line.m_internal.dir)
13919: (8)                  line_dict[dictkey_bbox] = JM_py_from_rect(line_rect)
13920: (8)                  line_list.append(line_dict)
13921: (4)              block_dict[dictkey_bbox] = JM_py_from_rect(block_rect)
13922: (4)              block_dict[dictkey_lines] = line_list
13923: (0)          def JM_make_textpage_dict(tp, page_dict, raw):
13924: (4)              if g_use_extra:
13925: (8)                  return extra.JM_make_textpage_dict(tp.m_internal, page_dict, raw)
13926: (4)              text_buffer = mupdf.fz_new_buffer(128)
13927: (4)              block_list = []
13928: (4)              tp_rect = mupdf.FzRect(tp.m_internal.mediabox)
13929: (4)              block_n = -1
13930: (4)              for block in tp:
```

```
13931: (8)                  block_n += 1
13932: (8)                  if (not mupdf.fz_contains_rect(tp_rect,
mupdf.FzRect(block.m_internal.bbox))
13933: (16)                         and not mupdf.fz_is_infinite_rect(tp_rect)
13934: (16)                         and block.m_internal.type == mupdf.FZ_STEXT_BLOCK_IMAGE
13935: (16)                         ):
13936: (12)                     continue
13937: (8)                  if (not mupdf.fz_is_infinite_rect(tp_rect)
13938: (16)                         and mupdf.fz_is_empty_rect(mupdf.fz_intersect_rect(tp_rect,
mupdf.FzRect(block.m_internal.bbox)))
13939: (16)                         ):
13940: (12)                     continue
13941: (8)                  block_dict = dict()
13942: (8)                  block_dict[dictkey_number] = block_n
13943: (8)                  block_dict[dictkey_type] = block.m_internal.type
13944: (8)                  if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_IMAGE:
13945: (12)                     block_dict[dictkey_bbox] = JM_py_from_rect(block.m_internal.bbox)
13946: (12)                     JM_make_image_block(block, block_dict)
13947: (8)                  else:
13948: (12)                     JM_make_text_block(block, block_dict, raw, text_buffer, tp_rect)
13949: (8)                  block_list.append(block_dict)
13950: (4)              page_dict[dictkey_blocks] = block_list
13951: (0)          def JM_matrix_from_py(m):
13952: (4)              a = [0, 0, 0, 0, 0, 0]
13953: (4)              if isinstance(m, mupdf.FzMatrix):
13954: (8)                  return m
13955: (4)              if isinstance(m, Matrix):
13956: (8)                  return mupdf.FzMatrix(m.a, m.b, m.c, m.d, m.e, m.f)
13957: (4)              if not m or not PySequence_Check(m) or PySequence_Size(m) != 6:
13958: (8)                  return mupdf.FzMatrix()
13959: (4)              for i in range(6):
13960: (8)                  a[i] = JM_FLOAT_ITEM(m, i)
13961: (8)                  if a[i] is None:
13962: (12)                     return mupdf.FzRect()
13963: (4)              return mupdf.FzMatrix(a[0], a[1], a[2], a[3], a[4], a[5])
13964: (0)          def JM_mediabox(page_obj):
13965: (4)              '''
13966: (4)              return a PDF page's MediaBox
13967: (4)              '''
13968: (4)              page_mediabox = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)
13969: (4)              mediabox = mupdf.pdf_to_rect(
13970: (12)                     mupdf.pdf_dict_get_inheritable(page_obj, PDF_NAME('MediaBox'))
13971: (12)                     )
13972: (4)              if mupdf.fz_is_empty_rect(mediabox) or
mupdf.fz_is_infinite_rect(mediabox):
13973: (8)                  mediabox.x0 = 0
13974: (8)                  mediabox.y0 = 0
13975: (8)                  mediabox.x1 = 612
13976: (8)                  mediabox.y1 = 792
13977: (4)              page_mediabox = mupdf.FzRect(
13978: (12)                     mupdf.fz_min(mediabox.x0, mediabox.x1),
13979: (12)                     mupdf.fz_min(mediabox.y0, mediabox.y1),
13980: (12)                     mupdf.fz_max(mediabox.x0, mediabox.x1),
13981: (12)                     mupdf.fz_max(mediabox.y0, mediabox.y1),
13982: (12)                     )
13983: (4)              if (page_mediabox.x1 - page_mediabox.x0 < 1
13984: (12)                     or page_mediabox.y1 - page_mediabox.y0 < 1
13985: (12)                     ):
13986: (8)                  page_mediabox = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)
13987: (4)              return page_mediabox
13988: (0)          def JM_merge_range(
13989: (8)                  doc_des,
13990: (8)                  doc_src,
13991: (8)                  spage,
13992: (8)                  epage,
13993: (8)                  apage,
13994: (8)                  rotate,
13995: (8)                  links,
13996: (8)                  annots,
```

```
13997: (8)                  show_progress,
13998: (8)                  graft_map,
13999: (8)                  ):
14000: (4)              '''
14001: (4)              Copy a range of pages (spage, epage) from a source PDF to a specified
14002: (4)              location (apage) of the target PDF.
14003: (4)              If spage > epage, the sequence of source pages is reversed.
14004: (4)              '''
14005: (4)              if g_use_extra:
14006: (8)                  return extra.JM_merge_range(
14007: (16)                         doc_des,
14008: (16)                         doc_src,
14009: (16)                         spage,
14010: (16)                         epage,
14011: (16)                         apage,
14012: (16)                         rotate,
14013: (16)                         links,
14014: (16)                         annots,
14015: (16)                         show_progress,
14016: (16)                         graft_map,
14017: (16)                         )
14018: (4)              afterpage = apage
14019: (4)              counter = 0  # copied pages counter
14020: (4)              total = mupdf.fz_absi(epage - spage) + 1   # total pages to copy
14021: (4)              if spage < epage:
14022: (8)                  page = spage
14023: (8)                  while page <= epage:
14024: (12)                     page_merge(doc_des, doc_src, page, afterpage, rotate, links,
annots, graft_map)
14025: (12)                     counter += 1
14026: (12)                     if show_progress > 0 and counter % show_progress == 0:
14027: (16)                         message(f"Inserted {counter} of {total} pages.")
14028: (12)                     page += 1
14029: (12)                     afterpage += 1
14030: (4)              else:
14031: (8)                  page = spage
14032: (8)                  while page >= epage:
14033: (12)                     page_merge(doc_des, doc_src, page, afterpage, rotate, links,
annots, graft_map)
14034: (12)                     counter += 1
14035: (12)                     if show_progress > 0 and counter % show_progress == 0:
14036: (16)                         message(f"Inserted {counter} of {total} pages.")
14037: (12)                     page -= 1
14038: (12)                     afterpage += 1
14039: (0)          def JM_merge_resources( page, temp_res):
14040: (4)              '''
14041: (4)              Merge the /Resources object created by a text pdf device into the page.
14042: (4)              The device may have created multiple /ExtGState/Alp? and /Font/F? objects.
14043: (4)              These need to be renamed (renumbered) to not overwrite existing page
14044: (4)              objects from previous executions.
14045: (4)              Returns the next available numbers n, m for objects /Alp<n>, /F<m>.
14046: (4)              '''
14047: (4)              resources = mupdf.pdf_dict_get(page.obj(), PDF_NAME('Resources'))
14048: (4)              main_extg = mupdf.pdf_dict_get(resources, PDF_NAME('ExtGState'))
14049: (4)              main_fonts = mupdf.pdf_dict_get(resources, PDF_NAME('Font'))
14050: (4)              temp_extg = mupdf.pdf_dict_get(temp_res, PDF_NAME('ExtGState'))
14051: (4)              temp_fonts = mupdf.pdf_dict_get(temp_res, PDF_NAME('Font'))
14052: (4)              max_alp = -1
14053: (4)              max_fonts = -1
14054: (4)              if mupdf.pdf_is_dict(temp_extg):   # any created at all?
14055: (8)                  n = mupdf.pdf_dict_len(temp_extg)
14056: (8)                  if mupdf.pdf_is_dict(main_extg):   # does page have /ExtGState yet?
14057: (12)                     for i in range(mupdf.pdf_dict_len(main_extg)):
14058: (16)                         alp = mupdf.pdf_to_name( mupdf.pdf_dict_get_key(main_extg, i))
14059: (16)                         if not alp.startswith('Alp'):
14060: (20)                             continue
14061: (16)                         j = mupdf.fz_atoi(alp[3:])
14062: (16)                         if j > max_alp:
14063: (20)                             max_alp = j
```

```
14064: (8)                  else:   # create a /ExtGState for the page
14065: (12)                     main_extg = mupdf.pdf_dict_put_dict(resources,
PDF_NAME('ExtGState'), n)
14066: (8)                  max_alp += 1
14067: (8)                  for i in range(n):  # copy over renumbered /Alp objects
14068: (12)                     alp = mupdf.pdf_to_name( mupdf.pdf_dict_get_key( temp_extg, i))
14069: (12)                     j = mupdf.fz_atoi(alp[3:]) + max_alp
14070: (12)                     text = f'Alp{j}'
14071: (12)                     val = mupdf.pdf_dict_get_val( temp_extg, i)
14072: (12)                     mupdf.pdf_dict_puts(main_extg, text, val)
14073: (4)              if mupdf.pdf_is_dict(main_fonts):  # has page any fonts yet?
14074: (8)                  for i in range(mupdf.pdf_dict_len(main_fonts)):    # get max font
number
14075: (12)                     font = mupdf.pdf_to_name( mupdf.pdf_dict_get_key( main_fonts, i))
14076: (12)                     if not font.startswith("F"):
14077: (16)                         continue
14078: (12)                     j = mupdf.fz_atoi(font[1:])
14079: (12)                     if j > max_fonts:
14080: (16)                         max_fonts = j
14081: (4)              else:   # create a Resources/Font for the page
14082: (8)                  main_fonts = mupdf.pdf_dict_put_dict(resources, PDF_NAME('Font'), 2)
14083: (4)              max_fonts += 1
14084: (4)              for i in range(mupdf.pdf_dict_len(temp_fonts)):    # copy renumbered fonts
14085: (8)                  font = mupdf.pdf_to_name( mupdf.pdf_dict_get_key( temp_fonts, i))
14086: (8)                  j = mupdf.fz_atoi(font[1:]) + max_fonts
14087: (8)                  text = f'F{j}'
14088: (8)                  val = mupdf.pdf_dict_get_val(temp_fonts, i)
14089: (8)                  mupdf.pdf_dict_puts(main_fonts, text, val)
14090: (4)              return (max_alp, max_fonts) # next available numbers
14091: (0)          def JM_mupdf_warning( text):
14092: (4)              '''
14093: (4)              redirect MuPDF warnings
14094: (4)              '''
14095: (4)              JM_mupdf_warnings_store.append(text)
14096: (4)              if JM_mupdf_show_warnings:
14097: (8)                  message(f'MuPDF warning: {text}')
14098: (0)          def JM_mupdf_error( text):
14099: (4)              JM_mupdf_warnings_store.append(text)
14100: (4)              if JM_mupdf_show_errors:
14101: (8)                  message(f'MuPDF error: {text}\n')
14102: (0)          def JM_new_bbox_device(rc, inc_layers):
14103: (4)              assert isinstance(rc, list)
14104: (4)              return JM_new_bbox_device_Device( rc, inc_layers)
14105: (0)          def JM_new_buffer_from_stext_page(page):
14106: (4)              '''
14107: (4)              make a buffer from an stext_page's text
14108: (4)              '''
14109: (4)              assert isinstance(page, mupdf.FzStextPage)
14110: (4)              rect = mupdf.FzRect(page.m_internal.mediabox)
14111: (4)              buf = mupdf.fz_new_buffer(256)
14112: (4)              for block in page:
14113: (8)                  if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_TEXT:
14114: (12)                     for line in block:
14115: (16)                         for ch in line:
14116: (20)                             if (not JM_rects_overlap(rect, JM_char_bbox(line, ch))
14117: (28)                                     and not mupdf.fz_is_infinite_rect(rect)
14118: (28)                                     ):
14119: (24)                                 continue
14120: (20)                             mupdf.fz_append_rune(buf, ch.m_internal.c)
14121: (16)                         mupdf.fz_append_byte(buf, ord('\n'))
14122: (12)                     mupdf.fz_append_byte(buf, ord('\n'))
14123: (4)              return buf
14124: (0)          def JM_new_javascript(pdf, value):
14125: (4)              '''
14126: (4)              make new PDF action object from JavaScript source
14127: (4)              Parameters are a PDF document and a Python string.
14128: (4)              Returns a PDF action object.
14129: (4)              '''
14130: (4)              if value is None:
```

```
14131: (8)                 return
14132: (4)             data = JM_StrAsChar(value)
14133: (4)             if data is None:
14134: (8)                 return
14135: (4)             res = mupdf.fz_new_buffer_from_copied_data(data.encode('utf8'))
14136: (4)             source = mupdf.pdf_add_stream(pdf, res, mupdf.PdfObj(), 0)
14137: (4)             newaction = mupdf.pdf_add_new_dict(pdf, 4)
14138: (4)             mupdf.pdf_dict_put(newaction, PDF_NAME('S'),
mupdf.pdf_new_name('JavaScript'))
14139: (4)             mupdf.pdf_dict_put(newaction, PDF_NAME('JS'), source)
14140: (4)             return newaction
14141: (0)         def JM_new_output_fileptr(bio):
14142: (4)             return JM_new_output_fileptr_Output( bio)
14143: (0)         def JM_norm_rotation(rotate):
14144: (4)             '''
14145: (4)             '''
14146: (4)             while rotate < 0:
14147: (8)                 rotate += 360
14148: (4)             while rotate >= 360:
14149: (8)                 rotate -= 360
14150: (4)             if rotate % 90 != 0:
14151: (8)                 return 0
14152: (4)             return rotate
14153: (0)         def JM_object_to_buffer(what, compress, ascii):
14154: (4)             res = mupdf.fz_new_buffer(512)
14155: (4)             out = mupdf.FzOutput(res)
14156: (4)             mupdf.pdf_print_obj(out, what, compress, ascii)
14157: (4)             out.fz_close_output()
14158: (4)             mupdf.fz_terminate_buffer(res)
14159: (4)             return res
14160: (0)         def JM_outline_xrefs(obj, xrefs):
14161: (4)             '''
14162: (4)             Return list of outline xref numbers. Recursive function. Arguments:
14163: (4)             'obj' first OL item
14164: (4)             'xrefs' empty Python list
14165: (4)             '''
14166: (4)             if not obj.m_internal:
14167: (8)                 return xrefs
14168: (4)             thisobj = obj
14169: (4)             while thisobj.m_internal:
14170: (8)                 newxref = mupdf.pdf_to_num( thisobj)
14171: (8)                 if newxref in xrefs or mupdf.pdf_dict_get( thisobj,
PDF_NAME('Type')).m_internal:
14172: (12)                    break
14173: (8)                 xrefs.append( newxref)
14174: (8)                 first = mupdf.pdf_dict_get( thisobj, PDF_NAME('First'))    # try go
down
14175: (8)                 if mupdf.pdf_is_dict( first):
14176: (12)                    xrefs = JM_outline_xrefs( first, xrefs)
14177: (8)                 thisobj = mupdf.pdf_dict_get( thisobj, PDF_NAME('Next'))   # try go
next
14178: (8)                 parent = mupdf.pdf_dict_get( thisobj, PDF_NAME('Parent'))  # get
parent
14179: (8)                 if not mupdf.pdf_is_dict( thisobj):
14180: (12)                    thisobj = parent
14181: (4)             return xrefs
14182: (0)         def JM_page_rotation(page):
14183: (4)             '''
14184: (4)             return a PDF page's /Rotate value: one of (0, 90, 180, 270)
14185: (4)             '''
14186: (4)             rotate = 0
14187: (4)             obj = mupdf.pdf_dict_get_inheritable( page.obj(),
mupdf.PDF_ENUM_NAME_Rotate)
14188: (4)             rotate = mupdf.pdf_to_int(obj)
14189: (4)             rotate = JM_norm_rotation(rotate)
14190: (4)             return rotate
14191: (0)         def JM_pdf_obj_from_str(doc, src):
14192: (4)             '''
14193: (4)             create PDF object from given string (new in v1.14.0: MuPDF dropped it)
```

```
14194: (4)              '''
14195: (4)              buffer_ = mupdf.fz_new_buffer_from_copied_data(bytes(src, 'utf8'))
14196: (4)              stream = mupdf.fz_open_buffer(buffer_)
14197: (4)              lexbuf = mupdf.PdfLexbuf(mupdf.PDF_LEXBUF_SMALL)
14198: (4)              result = mupdf.pdf_parse_stm_obj(doc, stream, lexbuf)
14199: (4)              return result
14200: (0)          def JM_pixmap_from_display_list(
14201: (8)                  list_,
14202: (8)                  ctm,
14203: (8)                  cs,
14204: (8)                  alpha,
14205: (8)                  clip,
14206: (8)                  seps,
14207: (8)                  ):
14208: (4)              '''
14209: (4)              Version of fz_new_pixmap_from_display_list (util.c) to also support
14210: (4)              rendering of only the 'clip' part of the displaylist rectangle
14211: (4)              '''
14212: (4)              assert isinstance(list_, mupdf.FzDisplayList)
14213: (4)              if seps is None:
14214: (8)                  seps = mupdf.FzSeparations()
14215: (4)              assert seps is None or isinstance(seps, mupdf.FzSeparations),
f'{type(seps)=}: {seps}'
14216: (4)              rect = mupdf.fz_bound_display_list(list_)
14217: (4)              matrix = JM_matrix_from_py(ctm)
14218: (4)              rclip = JM_rect_from_py(clip)
14219: (4)              rect = mupdf.fz_intersect_rect(rect, rclip)    # no-op if clip is not
given
14220: (4)              rect = mupdf.fz_transform_rect(rect, matrix)
14221: (4)              irect = mupdf.fz_round_rect(rect)
14222: (4)              assert isinstance( cs, mupdf.FzColorspace)
14223: (4)              pix = mupdf.fz_new_pixmap_with_bbox(cs, irect, seps, alpha)
14224: (4)              if alpha:
14225: (8)                  mupdf.fz_clear_pixmap(pix)
14226: (4)              else:
14227: (8)                  mupdf.fz_clear_pixmap_with_value(pix, 0xFF)
14228: (4)              if not mupdf.fz_is_infinite_rect(rclip):
14229: (8)                  dev = mupdf.fz_new_draw_device_with_bbox(matrix, pix, irect)
14230: (8)                  mupdf.fz_run_display_list(list_, dev, mupdf.FzMatrix(), rclip,
mupdf.FzCookie())
14231: (4)              else:
14232: (8)                  dev = mupdf.fz_new_draw_device(matrix, pix)
14233: (8)                  mupdf.fz_run_display_list(list_, dev, mupdf.FzMatrix(),
mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE), mupdf.FzCookie())
14234: (4)              mupdf.fz_close_device(dev)
14235: (4)              return Pixmap( 'raw', pix)
14236: (0)          def JM_point_from_py(p):
14237: (4)              '''
14238: (4)              PySequence to fz_point. Default: (FZ_MIN_INF_RECT, FZ_MIN_INF_RECT)
14239: (4)              '''
14240: (4)              if isinstance(p, mupdf.FzPoint):
14241: (8)                  return p
14242: (4)              if isinstance(p, Point):
14243: (8)                  return mupdf.FzPoint(p.x, p.y)
14244: (4)              if g_use_extra:
14245: (8)                  return extra.JM_point_from_py( p)
14246: (4)              p0 = mupdf.FzPoint(0, 0)
14247: (4)              x = JM_FLOAT_ITEM(p, 0)
14248: (4)              y = JM_FLOAT_ITEM(p, 1)
14249: (4)              if x is None or y is None:
14250: (8)                  return p0
14251: (4)              x = max( x, FZ_MIN_INF_RECT)
14252: (4)              y = max( y, FZ_MIN_INF_RECT)
14253: (4)              x = min( x, FZ_MAX_INF_RECT)
14254: (4)              y = min( y, FZ_MAX_INF_RECT)
14255: (4)              return mupdf.FzPoint(x, y)
14256: (0)          def JM_print_stext_page_as_text(res, page):
14257: (4)              '''
14258: (4)              Plain text output. An identical copy of fz_print_stext_page_as_text,
```

```
14259: (4)              but lines within a block are concatenated by space instead a new-line
14260: (4)              character (which else leads to 2 new-lines).
14261: (4)              '''
14262: (4)              if 1 and g_use_extra:
14263: (8)                  return extra.JM_print_stext_page_as_text(res, page)
14264: (4)              assert isinstance(res, mupdf.FzBuffer)
14265: (4)              assert isinstance(page, mupdf.FzStextPage)
14266: (4)              rect = mupdf.FzRect(page.m_internal.mediabox)
14267: (4)              last_char = 0
14268: (4)              n_blocks = 0
14269: (4)              n_lines = 0
14270: (4)              n_chars = 0
14271: (4)              for n_blocks2, block in enumerate( page):
14272: (8)                  if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_TEXT:
14273: (12)                     for n_lines2, line in enumerate( block):
14274: (16)                         for n_chars2, ch in enumerate( line):
14275: (20)                             pass
14276: (16)                         n_chars += n_chars2
14277: (12)                     n_lines += n_lines2
14278: (8)                  n_blocks += n_blocks2
14279: (4)              for block in page:
14280: (8)                  if block.m_internal.type == mupdf.FZ_STEXT_BLOCK_TEXT:
14281: (12)                     for line in block:
14282: (16)                         last_char = 0
14283: (16)                         for ch in line:
14284: (20)                             chbbox = JM_char_bbox(line, ch)
14285: (20)                             if (mupdf.fz_is_infinite_rect(rect)
14286: (28)                                     or JM_rects_overlap(rect, chbbox)
14287: (28)                                     ):
14288: (24)                                 last_char = ch.m_internal.c
14289: (24)                                 JM_append_rune(res, last_char)
14290: (16)                         if last_char != 10 and last_char > 0:
14291: (20)                             mupdf.fz_append_string(res, "\n")
14292: (0)          def JM_put_script(annot_obj, key1, key2, value):
14293: (4)              '''
14294: (4)              Create a JavaScript PDF action.
14295: (4)              Usable for all object types which support PDF actions, even if the
14296: (4)              argument name suggests annotations. Up to 2 key values can be specified,
so
14297: (4)              JavaScript actions can be stored for '/A' and '/AA/?' keys.
14298: (4)              '''
14299: (4)              key1_obj = mupdf.pdf_dict_get(annot_obj, key1)
14300: (4)              pdf = mupdf.pdf_get_bound_document(annot_obj)  # owning PDF
14301: (4)              if not value:
14302: (8)                  if key2 is None or not key2.m_internal:
14303: (12)                     mupdf.pdf_dict_del(annot_obj, key1)
14304: (8)                  elif key1_obj.m_internal:
14305: (12)                     mupdf.pdf_dict_del(key1_obj, key2)
14306: (8)                  return
14307: (4)              if not key2.m_internal or not key1_obj.m_internal:
14308: (8)                  script = JM_get_script(key1_obj)
14309: (4)              else:
14310: (8)                  script = JM_get_script(mupdf.pdf_dict_get(key1_obj, key2))
14311: (4)              if value != script:
14312: (8)                  newaction = JM_new_javascript(pdf, value)
14313: (8)                  if not key2.m_internal:
14314: (12)                     mupdf.pdf_dict_put(annot_obj, key1, newaction)
14315: (8)                  else:
14316: (12)                     mupdf.pdf_dict_putl(annot_obj, newaction, key1, key2)
14317: (0)          def JM_py_from_irect(r):
14318: (4)              return r.x0, r.y0, r.x1, r.y1
14319: (0)          def JM_py_from_matrix(m):
14320: (4)              return m.a, m.b, m.c, m.d, m.e, m.f
14321: (0)          def JM_py_from_point(p):
14322: (4)              return p.x, p.y
14323: (0)          def JM_py_from_quad(q):
14324: (4)              '''
14325: (4)              PySequence from fz_quad.
14326: (4)              '''
```

```
14327: (4)              return (
14328: (12)                     (q.ul.x, q.ul.y),
14329: (12)                     (q.ur.x, q.ur.y),
14330: (12)                     (q.ll.x, q.ll.y),
14331: (12)                     (q.lr.x, q.lr.y),
14332: (12)                     )
14333: (0)          def JM_py_from_rect(r):
14334: (4)              return r.x0, r.y0, r.x1, r.y1
14335: (0)          def JM_quad_from_py(r):
14336: (4)              if isinstance(r, mupdf.FzQuad):
14337: (8)                  return r
14338: (4)              if hasattr(r, "__getitem__") and len(r) == 4 and hasattr(r[0],
"__float__"):
14339: (8)                  r = mupdf.FzRect(*tuple(r))
14340: (4)              if isinstance( r, mupdf.FzRect):
14341: (8)                  return mupdf.fz_quad_from_rect( r)
14342: (4)              if isinstance( r, Quad):
14343: (8)                  return mupdf.fz_make_quad(
14344: (16)                         r.ul.x, r.ul.y,
14345: (16)                         r.ur.x, r.ur.y,
14346: (16)                         r.ll.x, r.ll.y,
14347: (16)                         r.lr.x, r.lr.y,
14348: (16)                         )
14349: (4)              q = mupdf.fz_make_quad(0, 0, 0, 0, 0, 0, 0, 0)
14350: (4)              p = [0,0,0,0]
14351: (4)              if not r or not isinstance(r, (tuple, list)) or len(r) != 4:
14352: (8)                  return q
14353: (4)              if JM_FLOAT_ITEM(r, 0) is None:
14354: (8)                  return mupdf.fz_quad_from_rect(JM_rect_from_py(r))
14355: (4)              for i in range(4):
14356: (8)                  if i >= len(r):
14357: (12)                     return q    # invalid: cancel the rest
14358: (8)                  obj = r[i]  # next point item
14359: (8)                  if not PySequence_Check(obj) or PySequence_Size(obj) != 2:
14360: (12)                     return q    # invalid: cancel the rest
14361: (8)                  p[i].x = JM_FLOAT_ITEM(obj, 0)
14362: (8)                  p[i].y = JM_FLOAT_ITEM(obj, 1)
14363: (8)                  if p[i].x is None or p[i].y is None:
14364: (12)                     return q
14365: (8)                  p[i].x = max( p[i].x, FZ_MIN_INF_RECT)
14366: (8)                  p[i].y = max( p[i].y, FZ_MIN_INF_RECT)
14367: (8)                  p[i].x = min( p[i].x, FZ_MAX_INF_RECT)
14368: (8)                  p[i].y = min( p[i].y, FZ_MAX_INF_RECT)
14369: (4)              q.ul = p[0]
14370: (4)              q.ur = p[1]
14371: (4)              q.ll = p[2]
14372: (4)              q.lr = p[3]
14373: (4)              return q
14374: (0)          def JM_read_contents(pageref):
14375: (4)              '''
14376: (4)              Read and concatenate a PDF page's /Contents object(s) in a buffer
14377: (4)              '''
14378: (4)              assert isinstance(pageref, mupdf.PdfObj), f'{type(pageref)}'
14379: (4)              contents = mupdf.pdf_dict_get(pageref, mupdf.PDF_ENUM_NAME_Contents)
14380: (4)              if mupdf.pdf_is_array(contents):
14381: (8)                  res = mupdf.FzBuffer(1024)
14382: (8)                  for i in range(mupdf.pdf_array_len(contents)):
14383: (12)                     if i > 0:
14384: (16)                         mupdf.fz_append_byte(res, 32)
14385: (12)                     obj = mupdf.pdf_array_get(contents, i)
14386: (12)                     if mupdf.pdf_is_stream(obj):
14387: (16)                         nres = mupdf.pdf_load_stream(obj)
14388: (16)                         mupdf.fz_append_buffer(res, nres)
14389: (4)              elif contents.m_internal:
14390: (8)                  res = mupdf.pdf_load_stream(contents)
14391: (4)              else:
14392: (8)                  res = b""
14393: (4)              return res
14394: (0)          def JM_rect_from_py(r):
```

```
(4)             if isinstance(r, mupdf.FzRect):
(8)                 return r
(4)             if isinstance(r, mupdf.FzIrect):
(8)                 return mupdf.FzRect(r)
(4)             if isinstance(r, Rect):
(8)                 return mupdf.fz_make_rect(r.x0, r.y0, r.x1, r.y1)
(4)             if isinstance(r, IRect):
(8)                 return mupdf.fz_make_rect(r.x0, r.y0, r.x1, r.y1)
(4)             if not r or not PySequence_Check(r) or PySequence_Size(r) != 4:
(8)                 return mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE)
(4)             f = [0, 0, 0, 0]
(4)             for i in range(4):
(8)                 f[i] = JM_FLOAT_ITEM(r, i)
(8)                 if f[i] is None:
(12)                    return mupdf.FzRect(mupdf.FzRect.Fixed_INFINITE)
(8)                 if f[i] < FZ_MIN_INF_RECT:
(12)                    f[i] = FZ_MIN_INF_RECT
(8)                 if f[i] > FZ_MAX_INF_RECT:
(12)                    f[i] = FZ_MAX_INF_RECT
(4)             return mupdf.fz_make_rect(f[0], f[1], f[2], f[3])
(0)         def JM_rects_overlap(a, b):
(4)             if (0
(12)                    or a.x0 >= b.x1
(12)                    or a.y0 >= b.y1
(12)                    or a.x1 <= b.x0
(12)                    or a.y1 <= b.y0
(12)                    ):
(8)                 return 0
(4)             return 1
(0)         def JM_refresh_links( page):
(4)             '''
(4)             refreshes the link and annotation tables of a page
(4)             '''
(4)             if page is None or not page.m_internal:
(8)                 return
(4)             obj = mupdf.pdf_dict_get( page.obj(), PDF_NAME('Annots'))
(4)             if obj.m_internal:
(8)                 pdf = page.doc()
(8)                 number = mupdf.pdf_lookup_page_number( pdf, page.obj())
(8)                 page_mediabox = mupdf.FzRect()
(8)                 page_ctm = mupdf.FzMatrix()
(8)                 mupdf.pdf_page_transform( page, page_mediabox, page_ctm)
(8)                 link = mupdf.pdf_load_link_annots( pdf, page, obj, number, page_ctm)
(8)                 page.m_internal.links = mupdf.ll_fz_keep_link( link.m_internal)
(0)         def JM_rotate_page_matrix(page):
(4)             '''
(4)             calculate page rotation matrices
(4)             '''
(4)             if not page.m_internal:
(8)                 return mupdf.FzMatrix()  # no valid pdf page given
(4)             rotation = JM_page_rotation(page)
(4)             if rotation == 0:
(8)                 return mupdf.FzMatrix()  # no rotation
(4)             cb_size = JM_cropbox_size(page.obj())
(4)             w = cb_size.x
(4)             h = cb_size.y
(4)             if rotation == 90:
(8)                 m = mupdf.fz_make_matrix(0, 1, -1, 0, h, 0)
(4)             elif rotation == 180:
(8)                 m = mupdf.fz_make_matrix(-1, 0, 0, -1, w, h)
(4)             else:
(8)                 m = mupdf.fz_make_matrix(0, -1, 1, 0, 0, w)
(4)             return m
(0)         def JM_search_stext_page(page, needle):
(4)             if g_use_extra:
(8)                 return extra.JM_search_stext_page(page.m_internal, needle)
(4)             rect = mupdf.FzRect(page.m_internal.mediabox)
(4)             if not needle:
(8)                 return
```

```
14464: (4)              quads = []
14465: (4)              class Hits:
14466: (8)                  def __str__(self):
14467: (12)                     return f'Hits(len={self.len} quads={self.quads} hfuzz={self.hfuzz}
vfuzz={self.vfuzz}'
14468: (4)              hits = Hits()
14469: (4)              hits.len = 0
14470: (4)              hits.quads = quads
14471: (4)              hits.hfuzz = 0.2    # merge kerns but not large gaps
14472: (4)              hits.vfuzz = 0.1
14473: (4)              buffer_ = JM_new_buffer_from_stext_page(page)
14474: (4)              haystack_string = mupdf.fz_string_from_buffer(buffer_)
14475: (4)              haystack = 0
14476: (4)              begin, end = find_string(haystack_string[haystack:], needle)
14477: (4)              if begin is None:
14478: (8)                  return quads
14479: (4)              begin += haystack
14480: (4)              end += haystack
14481: (4)              inside = 0
14482: (4)              i = 0
14483: (4)              for block in page:
14484: (8)                  if block.m_internal.type != mupdf.FZ_STEXT_BLOCK_TEXT:
14485: (12)                     continue
14486: (8)                  for line in block:
14487: (12)                     for ch in line:
14488: (16)                         i += 1
14489: (16)                         if not mupdf.fz_is_infinite_rect(rect):
14490: (20)                             r = JM_char_bbox(line, ch)
14491: (20)                             if not JM_rects_overlap(rect, r):
14492: (24)                                 continue
14493: (16)                         while 1:
14494: (20)                             if not inside:
14495: (24)                                 if haystack >= begin:
14496: (28)                                     inside = 1
14497: (20)                             if inside:
14498: (24)                                 if haystack < end:
14499: (28)                                     on_highlight_char(hits, line, ch)
14500: (28)                                     break
14501: (24)                                 else:
14502: (28)                                     inside = 0
14503: (28)                                     begin, end =
find_string(haystack_string[haystack:], needle)
14504: (28)                                     if begin is None:
14505: (32)                                         return quads
14506: (28)                                     else:
14507: (32)                                         begin += haystack
14508: (32)                                         end += haystack
14509: (32)                                         continue
14510: (20)                             break
14511: (16)                         haystack += 1
14512: (12)                     assert haystack_string[haystack] == '\n', \
14513: (20)                             f'{haystack=} {haystack_string[haystack]=}'
14514: (12)                     haystack += 1
14515: (8)                  assert haystack_string[haystack] == '\n', \
14516: (16)                         f'{haystack=} {haystack_string[haystack]=}'
14517: (8)                  haystack += 1
14518: (4)              return quads
14519: (0)          def JM_scan_resources(pdf, rsrc, liste, what, stream_xref, tracer):
14520: (4)              '''
14521: (4)              Step through /Resources, looking up image, xobject or font information
14522: (4)              '''
14523: (4)              if mupdf.pdf_mark_obj(rsrc):
14524: (8)                  mupdf.fz_warn('Circular dependencies! Consider page cleaning.')
14525: (8)                  return  # Circular dependencies!
14526: (4)              try:
14527: (8)                  xobj = mupdf.pdf_dict_get(rsrc, mupdf.PDF_ENUM_NAME_XObject)
14528: (8)                  if what == 1:   # lookup fonts
14529: (12)                     font = mupdf.pdf_dict_get(rsrc, mupdf.PDF_ENUM_NAME_Font)
14530: (12)                     JM_gather_fonts(pdf, font, liste, stream_xref)
```

```
14531: (8)                  elif what == 2: # look up images
14532: (12)                     JM_gather_images(pdf, xobj, liste, stream_xref)
14533: (8)                  elif what == 3: # look up form xobjects
14534: (12)                     JM_gather_forms(pdf, xobj, liste, stream_xref)
14535: (8)                  else:   # should never happen
14536: (12)                     return
14537: (8)                  n = mupdf.pdf_dict_len(xobj)
14538: (8)                  for i in range(n):
14539: (12)                     obj = mupdf.pdf_dict_get_val(xobj, i)
14540: (12)                     if mupdf.pdf_is_stream(obj):
14541: (16)                         sxref = mupdf.pdf_to_num(obj)
14542: (12)                     else:
14543: (16)                         sxref = 0
14544: (12)                     subrsrc = mupdf.pdf_dict_get(obj, mupdf.PDF_ENUM_NAME_Resources)
14545: (12)                     if subrsrc.m_internal:
14546: (16)                         sxref_t = sxref
14547: (16)                         if sxref_t not in tracer:
14548: (20)                             tracer.append(sxref_t)
14549: (20)                             JM_scan_resources( pdf, subrsrc, liste, what, sxref,
tracer)
14550: (16)                         else:
14551: (20)                             mupdf.fz_warn('Circular dependencies! Consider page
cleaning.')
14552: (20)                             return
14553: (4)              finally:
14554: (8)                  mupdf.pdf_unmark_obj(rsrc)
14555: (0)          def JM_set_choice_options(annot, liste):
14556: (4)              '''
14557: (4)              set ListBox / ComboBox values
14558: (4)              '''
14559: (4)              if not liste:
14560: (8)                  return
14561: (4)              assert isinstance( liste, (tuple, list))
14562: (4)              n = len( liste)
14563: (4)              if n == 0:
14564: (8)                  return
14565: (4)              annot_obj = mupdf.pdf_annot_obj( annot)
14566: (4)              pdf = mupdf.pdf_get_bound_document( annot_obj)
14567: (4)              optarr = mupdf.pdf_new_array( pdf, n)
14568: (4)              for i in range(n):
14569: (8)                  val = liste[i]
14570: (8)                  opt = val
14571: (8)                  if isinstance(opt, str):
14572: (12)                     mupdf.pdf_array_push_text_string( optarr, opt)
14573: (8)                  else:
14574: (12)                     assert isinstance( val, (tuple, list)) and len( val) == 2, 'bad
choice field list'
14575: (12)                     opt1, opt2 = val
14576: (12)                     assert opt1 and opt2, 'bad choice field list'
14577: (12)                     optarrsub = mupdf.pdf_array_push_array( optarr, 2)
14578: (12)                     mupdf.pdf_array_push_text_string( optarrsub, opt1)
14579: (12)                     mupdf.pdf_array_push_text_string( optarrsub, opt2)
14580: (4)              mupdf.pdf_dict_put( annot_obj, PDF_NAME('Opt'), optarr)
14581: (0)          def JM_set_field_type(doc, obj, type):
14582: (4)              '''
14583: (4)              Set the field type
14584: (4)              '''
14585: (4)              setbits = 0
14586: (4)              clearbits = 0
14587: (4)              typename = None
14588: (4)              if type == mupdf.PDF_WIDGET_TYPE_BUTTON:
14589: (8)                  typename = PDF_NAME('Btn')
14590: (8)                  setbits = mupdf.PDF_BTN_FIELD_IS_PUSHBUTTON
14591: (4)              elif type == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON:
14592: (8)                  typename = PDF_NAME('Btn')
14593: (8)                  clearbits = mupdf.PDF_BTN_FIELD_IS_PUSHBUTTON
14594: (8)                  setbits = mupdf.PDF_BTN_FIELD_IS_RADIO
14595: (4)              elif type == mupdf.PDF_WIDGET_TYPE_CHECKBOX:
14596: (8)                  typename = PDF_NAME('Btn')
```

```
14597: (8)                  clearbits = (mupdf.PDF_BTN_FIELD_IS_PUSHBUTTON |
mupdf.PDF_BTN_FIELD_IS_RADIO)
14598: (4)              elif type == mupdf.PDF_WIDGET_TYPE_TEXT:
14599: (8)                  typename = PDF_NAME('Tx')
14600: (4)              elif type == mupdf.PDF_WIDGET_TYPE_LISTBOX:
14601: (8)                  typename = PDF_NAME('Ch')
14602: (8)                  clearbits = mupdf.PDF_CH_FIELD_IS_COMBO
14603: (4)              elif type == mupdf.PDF_WIDGET_TYPE_COMBOBOX:
14604: (8)                  typename = PDF_NAME('Ch')
14605: (8)                  setbits = mupdf.PDF_CH_FIELD_IS_COMBO
14606: (4)              elif type == mupdf.PDF_WIDGET_TYPE_SIGNATURE:
14607: (8)                  typename = PDF_NAME('Sig')
14608: (4)              if typename is not None and typename.m_internal:
14609: (8)                  mupdf.pdf_dict_put(obj, PDF_NAME('FT'), typename)
14610: (4)              if setbits != 0 or clearbits != 0:
14611: (8)                  bits = mupdf.pdf_dict_get_int(obj, PDF_NAME('Ff'))
14612: (8)                  bits &= ~clearbits
14613: (8)                  bits |= setbits
14614: (8)                  mupdf.pdf_dict_put_int(obj, PDF_NAME('Ff'), bits)
14615: (0)          def JM_set_object_value(obj, key, value):
14616: (4)              '''
14617: (4)              Set a PDF dict key to some value
14618: (4)              '''
14619: (4)              eyecatcher = "fitz: replace me!"
14620: (4)              pdf = mupdf.pdf_get_bound_document(obj)
14621: (4)              list_ = key.split('/')
14622: (4)              len_ = len(list_)
14623: (4)              i = len_ - 1
14624: (4)              skey = list_[i]
14625: (4)              del list_[i]    # del the last sub-key
14626: (4)              len_ = len(list_)   # remaining length
14627: (4)              testkey = mupdf.pdf_dict_getp(obj, key)    # check if key already exists
14628: (4)              if not testkey.m_internal:
14629: (8)                  while len_ > 0:
14630: (12)                     t = '/'.join(list_) # next high level
14631: (12)                     if mupdf.pdf_is_indirect(mupdf.pdf_dict_getp(obj,
JM_StrAsChar(t))):
14632: (16)                         raise Exception("path to '%s' has indirects",
JM_StrAsChar(skey))
14633: (12)                     del list_[len_ - 1]   # del last sub-key
14634: (12)                     len_ = len(list_)   # remaining length
14635: (4)              mupdf.pdf_dict_putp(obj, key, mupdf.pdf_new_text_string(eyecatcher))
14636: (4)              testkey = mupdf.pdf_dict_getp(obj, key)
14637: (4)              if not mupdf.pdf_is_string(testkey):
14638: (8)                  raise Exception("cannot insert value for '%s'", key)
14639: (4)              temp = mupdf.pdf_to_text_string(testkey)
14640: (4)              if temp != eyecatcher:
14641: (8)                  raise Exception("cannot insert value for '%s'", key)
14642: (4)              res = JM_object_to_buffer(obj, 1, 0)
14643: (4)              objstr = JM_EscapeStrFromBuffer(res)
14644: (4)              nullval = "/%s(%s)" % ( skey, eyecatcher)
14645: (4)              newval = "/%s %s" % (skey, value)
14646: (4)              newstr = objstr.replace(nullval, newval, 1)
14647: (4)              new_obj = JM_pdf_obj_from_str(pdf, newstr)
14648: (4)              return new_obj
14649: (0)          def JM_set_ocg_arrays(conf, basestate, on, off, rbgroups, locked):
14650: (4)              if basestate:
14651: (8)                  mupdf.pdf_dict_put_name( conf, PDF_NAME('BaseState'), basestate)
14652: (4)              if on is not None:
14653: (8)                  mupdf.pdf_dict_del( conf, PDF_NAME('ON'))
14654: (8)                  if on:
14655: (12)                     arr = mupdf.pdf_dict_put_array( conf, PDF_NAME('ON'), 1)
14656: (12)                     JM_set_ocg_arrays_imp( arr, on)
14657: (4)              if off is not None:
14658: (8)                  mupdf.pdf_dict_del( conf, PDF_NAME('OFF'))
14659: (8)                  if off:
14660: (12)                     arr = mupdf.pdf_dict_put_array( conf, PDF_NAME('OFF'), 1)
14661: (12)                     JM_set_ocg_arrays_imp( arr, off)
14662: (4)              if locked is not None:
```

```
14663: (8)                  mupdf.pdf_dict_del( conf, PDF_NAME('Locked'))
14664: (8)                  if locked:
14665: (12)                     arr = mupdf.pdf_dict_put_array( conf, PDF_NAME('Locked'), 1)
14666: (12)                     JM_set_ocg_arrays_imp( arr, locked)
14667: (4)              if rbgroups is not None:
14668: (8)                  mupdf.pdf_dict_del( conf, PDF_NAME('RBGroups'))
14669: (8)                  if rbgroups:
14670: (12)                     arr = mupdf.pdf_dict_put_array( conf, PDF_NAME('RBGroups'), 1)
14671: (12)                     n =len(rbgroups)
14672: (12)                     for i in range(n):
14673: (16)                         item0 = rbgroups[i]
14674: (16)                         obj = mupdf.pdf_array_push_array( arr, 1)
14675: (16)                         JM_set_ocg_arrays_imp( obj, item0)
14676: (0)          def JM_set_ocg_arrays_imp(arr, list_):
14677: (4)              '''
14678: (4)              Set OCG arrays from dict of Python lists
14679: (4)              Works with dict like {"basestate":name, "on":list, "off":list, "rbg":list}
14680: (4)              '''
14681: (4)              pdf = mupdf.pdf_get_bound_document(arr)
14682: (4)              for xref in list_:
14683: (8)                  obj = mupdf.pdf_new_indirect(pdf, xref, 0)
14684: (8)                  mupdf.pdf_array_push(arr, obj)
14685: (0)          def JM_set_resource_property(ref, name, xref):
14686: (4)              '''
14687: (4)              Insert an item into Resources/Properties (used for Marked Content)
14688: (4)              Arguments:
14689: (4)              (1) e.g. page object, Form XObject
14690: (4)              (2) marked content name
14691: (4)              (3) xref of the referenced object (insert as indirect reference)
14692: (4)              '''
14693: (4)              pdf = mupdf.pdf_get_bound_document(ref)
14694: (4)              ind = mupdf.pdf_new_indirect(pdf, xref, 0)
14695: (4)              if not ind.m_internal:
14696: (8)                  RAISEPY(MSG_BAD_XREF, PyExc_ValueError)
14697: (4)              resources = mupdf.pdf_dict_get(ref, PDF_NAME('Resources'))
14698: (4)              if not resources.m_internal:
14699: (8)                  resources = mupdf.pdf_dict_put_dict(ref, PDF_NAME('Resources'), 1)
14700: (4)              properties = mupdf.pdf_dict_get(resources, PDF_NAME('Properties'))
14701: (4)              if not properties.m_internal:
14702: (8)                  properties = mupdf.pdf_dict_put_dict(resources,
PDF_NAME('Properties'), 1)
14703: (4)              mupdf.pdf_dict_put(properties, mupdf.pdf_new_name(name), ind)
14704: (0)          def JM_set_widget_properties(annot, Widget):
14705: (4)              '''
14706: (4)              Update the PDF form field with the properties from a Python Widget object.
14707: (4)              Called by "Page.add_widget" and "Annot.update_widget".
14708: (4)              '''
14709: (4)              if isinstance( annot, Annot):
14710: (8)                  annot = annot.this
14711: (4)              assert isinstance( annot, mupdf.PdfAnnot), f'{type(annot)=} {type=}'
14712: (4)              page = mupdf.pdf_annot_page(annot)
14713: (4)              annot_obj = mupdf.pdf_annot_obj(annot)
14714: (4)              pdf = page.doc()
14715: (4)              def GETATTR(name):
14716: (8)                  return getattr(Widget, name, None)
14717: (4)              value = GETATTR("field_type")
14718: (4)              field_type = value
14719: (4)              value = GETATTR("rect")
14720: (4)              rect = JM_rect_from_py(value)
14721: (4)              rot_mat = JM_rotate_page_matrix(page)
14722: (4)              rect = mupdf.fz_transform_rect(rect, rot_mat)
14723: (4)              mupdf.pdf_set_annot_rect(annot, rect)
14724: (4)              value = GETATTR("fill_color")
14725: (4)              if value and PySequence_Check(value):
14726: (8)                  n = len(value)
14727: (8)                  fill_col = mupdf.pdf_new_array(pdf, n)
14728: (8)                  col = 0
14729: (8)                  for i in range(n):
14730: (12)                     col = value[i]
```

```
14731: (12)                     mupdf.pdf_array_push_real(fill_col, col)
14732: (8)                  mupdf.pdf_field_set_fill_color(annot_obj, fill_col)
14733: (4)              value = GETATTR("border_dashes")
14734: (4)              if value and PySequence_Check(value):
14735: (8)                  n = len(value)
14736: (8)                  dashes = mupdf.pdf_new_array(pdf, n)
14737: (8)                  for i in range(n):
14738: (12)                     mupdf.pdf_array_push_int(dashes, value[i])
14739: (8)                  mupdf.pdf_dict_putl(annot_obj, dashes, PDF_NAME('BS'), PDF_NAME('D'))
14740: (4)              value = GETATTR("border_color")
14741: (4)              if value and PySequence_Check(value):
14742: (8)                  n = len(value)
14743: (8)                  border_col = mupdf.pdf_new_array(pdf, n)
14744: (8)                  col = 0
14745: (8)                  for i in range(n):
14746: (12)                     col = value[i]
14747: (12)                     mupdf.pdf_array_push_real(border_col, col)
14748: (8)                  mupdf.pdf_dict_putl(annot_obj, border_col, PDF_NAME('MK'),
PDF_NAME('BC'))
14749: (4)              value = GETATTR("field_label")
14750: (4)              if value is not None:
14751: (8)                  label = JM_StrAsChar(value)
14752: (8)                  mupdf.pdf_dict_put_text_string(annot_obj, PDF_NAME('TU'), label)
14753: (4)              value = GETATTR("field_name")
14754: (4)              if value is not None:
14755: (8)                  name = JM_StrAsChar(value)
14756: (8)                  old_name = mupdf.pdf_load_field_name(annot_obj)
14757: (8)                  if name != old_name:
14758: (12)                     mupdf.pdf_dict_put_text_string(annot_obj, PDF_NAME('T'), name)
14759: (4)              if field_type == mupdf.PDF_WIDGET_TYPE_TEXT:
14760: (8)                  value = GETATTR("text_maxlen")
14761: (8)                  text_maxlen = value
14762: (8)                  if text_maxlen:
14763: (12)                     mupdf.pdf_dict_put_int(annot_obj, PDF_NAME('MaxLen'), text_maxlen)
14764: (4)              value = GETATTR("field_display")
14765: (4)              d = value
14766: (4)              mupdf.pdf_field_set_display(annot_obj, d)
14767: (4)              if field_type in (mupdf.PDF_WIDGET_TYPE_LISTBOX,
mupdf.PDF_WIDGET_TYPE_COMBOBOX):
14768: (8)                  value = GETATTR("choice_values")
14769: (8)                  JM_set_choice_options(annot, value)
14770: (4)              value = GETATTR("border_style")
14771: (4)              val = JM_get_border_style(value)
14772: (4)              mupdf.pdf_dict_putl(annot_obj, val, PDF_NAME('BS'), PDF_NAME('S'))
14773: (4)              value = GETATTR("border_width")
14774: (4)              border_width = value
14775: (4)              mupdf.pdf_dict_putl(
14776: (12)                     annot_obj,
14777: (12)                     mupdf.pdf_new_real(border_width),
14778: (12)                     PDF_NAME('BS'),
14779: (12)                     PDF_NAME('W'),
14780: (12)                     )
14781: (4)              value = GETATTR("_text_da")
14782: (4)              da = JM_StrAsChar(value)
14783: (4)              mupdf.pdf_dict_put_text_string(annot_obj, PDF_NAME('DA'), da)
14784: (4)              mupdf.pdf_dict_del(annot_obj, PDF_NAME('DS'))  # not supported by MuPDF
14785: (4)              mupdf.pdf_dict_del(annot_obj, PDF_NAME('RC'))  # not supported by MuPDF
14786: (4)              field_flags = GETATTR("field_flags")
14787: (4)              if field_flags is not None:
14788: (8)                  if field_type == mupdf.PDF_WIDGET_TYPE_COMBOBOX:
14789: (12)                     field_flags |= mupdf.PDF_CH_FIELD_IS_COMBO
14790: (8)                  elif field_type == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON:
14791: (12)                     field_flags |= mupdf.PDF_BTN_FIELD_IS_RADIO
14792: (8)                  elif field_type == mupdf.PDF_WIDGET_TYPE_BUTTON:
14793: (12)                     field_flags |= mupdf.PDF_BTN_FIELD_IS_PUSHBUTTON
14794: (8)                  mupdf.pdf_dict_put_int( annot_obj, PDF_NAME('Ff'), field_flags)
14795: (4)              value = GETATTR("button_caption")
14796: (4)              ca = JM_StrAsChar(value)
14797: (4)              if ca:
```

```
14798: (8)                  mupdf.pdf_field_set_button_caption(annot_obj, ca)
14799: (4)              value = GETATTR("script")
14800: (4)              JM_put_script(annot_obj, PDF_NAME('A'), mupdf.PdfObj(), value)
14801: (4)              value = GETATTR("script_stroke")
14802: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), PDF_NAME('K'), value)
14803: (4)              value = GETATTR("script_format")
14804: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), PDF_NAME('F'), value)
14805: (4)              value = GETATTR("script_change")
14806: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), PDF_NAME('V'), value)
14807: (4)              value = GETATTR("script_calc")
14808: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), PDF_NAME('C'), value)
14809: (4)              value = GETATTR("script_blur")
14810: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), mupdf.pdf_new_name('Bl'), value)
14811: (4)              value = GETATTR("script_focus")
14812: (4)              JM_put_script(annot_obj, PDF_NAME('AA'), mupdf.pdf_new_name('Fo'), value)
14813: (4)              value = GETATTR("field_value")  # field value
14814: (4)              text = JM_StrAsChar(value)  # convert to text (may fail!)
14815: (4)              if field_type == mupdf.PDF_WIDGET_TYPE_RADIOBUTTON:
14816: (8)                  if not value:
14817: (12)                     mupdf.pdf_set_field_value(pdf, annot_obj, "Off", 1)
14818: (12)                     mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('AS'), "Off")
14819: (8)                  else:
14820: (12)                     onstate = mupdf.pdf_button_field_on_state(annot_obj)
14821: (12)                     if onstate.m_internal:
14822: (16)                         on = mupdf.pdf_to_name(onstate)
14823: (16)                         mupdf.pdf_set_field_value(pdf, annot_obj, on, 1)
14824: (16)                         mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('AS'), on)
14825: (12)                     elif text:
14826: (16)                         mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('AS'), text)
14827: (4)              elif field_type == mupdf.PDF_WIDGET_TYPE_CHECKBOX:    # will always be
"Yes" or "Off"
14828: (8)                  if value is True or text == 'Yes':
14829: (12)                     onstate = mupdf.pdf_button_field_on_state(annot_obj)
14830: (12)                     on = mupdf.pdf_to_name(onstate)
14831: (12)                     mupdf.pdf_set_field_value(pdf, annot_obj, on, 1)
14832: (12)                     mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('AS'), 'Yes')
14833: (12)                     mupdf.pdf_dict_put_name(annot_obj, PDF_NAME('V'), 'Yes')
14834: (8)                  else:
14835: (12)                     mupdf.pdf_dict_put_name( annot_obj, PDF_NAME('AS'), 'Off')
14836: (12)                     mupdf.pdf_dict_put_name( annot_obj, PDF_NAME('V'), 'Off')
14837: (4)              else:
14838: (8)                  if text:
14839: (12)                     mupdf.pdf_set_field_value(pdf, annot_obj, text, 1)
14840: (12)                     if field_type in (mupdf.PDF_WIDGET_TYPE_COMBOBOX,
mupdf.PDF_WIDGET_TYPE_LISTBOX):
14841: (16)                         mupdf.pdf_dict_del(annot_obj, PDF_NAME('I'))
14842: (4)              mupdf.pdf_dirty_annot(annot)
14843: (4)              mupdf.pdf_set_annot_hot(annot, 1)
14844: (4)              mupdf.pdf_set_annot_active(annot, 1)
14845: (4)              mupdf.pdf_update_annot(annot)
14846: (0)          def JM_show_string_cs(
14847: (8)                  text,
14848: (8)                  user_font,
14849: (8)                  trm,
14850: (8)                  s,
14851: (8)                  wmode,
14852: (8)                  bidi_level,
14853: (8)                  markup_dir,
14854: (8)                  language,
14855: (8)                  ):
14856: (4)              i = 0
14857: (4)              while i < len(s):
14858: (8)                  l, ucs = mupdf.fz_chartorune(s[i:])
14859: (8)                  i += l
14860: (8)                  gid = mupdf.fz_encode_character_sc(user_font, ucs)
14861: (8)                  if gid == 0:
14862: (12)                     gid, font = mupdf.fz_encode_character_with_fallback(user_font,
ucs, 0, language)
14863: (8)                  else:
```

```
14864: (12)                     font = user_font
14865: (8)                  mupdf.fz_show_glyph(text, font, trm, gid, ucs, wmode, bidi_level,
markup_dir, language)
14866: (8)                  adv = mupdf.fz_advance_glyph(font, gid, wmode)
14867: (8)                  if wmode == 0:
14868: (12)                     trm = mupdf.fz_pre_translate(trm, adv, 0)
14869: (8)                  else:
14870: (12)                     trm = mupdf.fz_pre_translate(trm, 0, -adv)
14871: (4)              return trm
14872: (0)          def JM_UnicodeFromBuffer(buff):
14873: (4)              buff_bytes = mupdf.fz_buffer_extract_copy(buff)
14874: (4)              val = buff_bytes.decode(errors='replace')
14875: (4)              z = val.find(chr(0))
14876: (4)              if z >= 0:
14877: (8)                  val = val[:z]
14878: (4)              return val
14879: (0)          def message_warning(text):
14880: (4)              '''
14881: (4)              Generate a warning.
14882: (4)              '''
14883: (4)              message(f'warning: {text}')
14884: (0)          def JM_update_stream(doc, obj, buffer_, compress):
14885: (4)              '''
14886: (4)              update a stream object
14887: (4)              compress stream when beneficial
14888: (4)              '''
14889: (4)              if compress:
14890: (8)                  length, _ = mupdf.fz_buffer_storage(buffer_)
14891: (8)                  if length > 30:   # ignore small stuff
14892: (12)                     buffer_compressed = JM_compress_buffer(buffer_)
14893: (12)                     assert isinstance(buffer_compressed, mupdf.FzBuffer)
14894: (12)                     if buffer_compressed.m_internal:
14895: (16)                         length_compressed, _ =
mupdf.fz_buffer_storage(buffer_compressed)
14896: (16)                         if length_compressed < length:  # was it worth the effort?
14897: (20)                             mupdf.pdf_dict_put(
14898: (28)                                     obj,
14899: (28)                                     mupdf.PDF_ENUM_NAME_Filter,
14900: (28)                                     mupdf.PDF_ENUM_NAME_FlateDecode,
14901: (28)                                     )
14902: (20)                             mupdf.pdf_update_stream(doc, obj, buffer_compressed, 1)
14903: (20)                             return
14904: (4)              mupdf.pdf_update_stream(doc, obj, buffer_, 0)
14905: (0)          def JM_xobject_from_page(pdfout, fsrcpage, xref, gmap):
14906: (4)              '''
14907: (4)              Make an XObject from a PDF page
14908: (4)              For a positive xref assume that its object can be used instead
14909: (4)              '''
14910: (4)              assert isinstance(gmap, mupdf.PdfGraftMap), f'{type(gmap)=}'
14911: (4)              if xref > 0:
14912: (8)                  xobj1 = mupdf.pdf_new_indirect(pdfout, xref, 0)
14913: (4)              else:
14914: (8)                  srcpage = _as_pdf_page(fsrcpage.this)
14915: (8)                  spageref = srcpage.obj()
14916: (8)                  mediabox = mupdf.pdf_to_rect(mupdf.pdf_dict_get_inheritable(spageref,
PDF_NAME('MediaBox')))
14917: (8)                  o = mupdf.pdf_dict_get_inheritable(spageref, PDF_NAME('Resources'))
14918: (8)                  if gmap.m_internal:
14919: (12)                     resources = mupdf.pdf_graft_mapped_object(gmap, o)
14920: (8)                  else:
14921: (12)                     resources = mupdf.pdf_graft_object(pdfout, o)
14922: (8)                  res = JM_read_contents(spageref)
14923: (8)                  xobj1 = mupdf.pdf_new_xobject(pdfout, mediabox, mupdf.FzMatrix(),
mupdf.PdfObj(0), res)
14924: (8)                  JM_update_stream(pdfout, xobj1, res, 1)
14925: (8)                  mupdf.pdf_dict_put(xobj1, PDF_NAME('Resources'), resources)
14926: (4)              return xobj1
14927: (0)          def PySequence_Check(s):
14928: (4)              return isinstance(s, (tuple, list))
```

```
(0)          def PySequence_Size(s):
(4)              return len(s)
(0)          MSG_BAD_ANNOT_TYPE = "bad annot type"
(0)          MSG_BAD_APN = "bad or missing annot AP/N"
(0)          MSG_BAD_ARG_INK_ANNOT = "arg must be seq of seq of float pairs"
(0)          MSG_BAD_ARG_POINTS = "bad seq of points"
(0)          MSG_BAD_BUFFER = "bad type: 'buffer'"
(0)          MSG_BAD_COLOR_SEQ = "bad color sequence"
(0)          MSG_BAD_DOCUMENT = "cannot open broken document"
(0)          MSG_BAD_FILETYPE = "bad filetype"
(0)          MSG_BAD_LOCATION = "bad location"
(0)          MSG_BAD_OC_CONFIG = "bad config number"
(0)          MSG_BAD_OC_LAYER = "bad layer number"
(0)          MSG_BAD_OC_REF = "bad 'oc' reference"
(0)          MSG_BAD_PAGEID = "bad page id"
(0)          MSG_BAD_PAGENO = "bad page number(s)"
(0)          MSG_BAD_PDFROOT = "PDF has no root"
(0)          MSG_BAD_RECT = "rect is infinite or empty"
(0)          MSG_BAD_TEXT = "bad type: 'text'"
(0)          MSG_BAD_XREF = "bad xref"
(0)          MSG_COLOR_COUNT_FAILED = "color count failed"
(0)          MSG_FILE_OR_BUFFER = "need font file or buffer"
(0)          MSG_FONT_FAILED = "cannot create font"
(0)          MSG_IS_NO_ANNOT = "is no annotation"
(0)          MSG_IS_NO_IMAGE = "is no image"
(0)          MSG_IS_NO_PDF = "is no PDF"
(0)          MSG_IS_NO_DICT = "object is no PDF dict"
(0)          MSG_PIX_NOALPHA = "source pixmap has no alpha"
(0)          MSG_PIXEL_OUTSIDE = "pixel(s) outside image"
(0)          JM_Exc_FileDataError = 'FileDataError'
(0)          PyExc_ValueError = 'ValueError'
(0)          def RAISEPY( msg, exc):
(4)              raise Exception( msg)
(0)          def PyUnicode_DecodeRawUnicodeEscape(s, errors='strict'):
(4)              if not s:
(8)                  return ""
(4)              if isinstance(s, str):
(8)                  rc = s.encode("utf8", errors=errors)
(4)              elif isinstance(s, bytes):
(8)                  rc = s[:]
(4)              ret = rc.decode('raw_unicode_escape', errors=errors)
(4)              return ret
(0)          def CheckColor(c: OptSeq):
(4)              if c:
(8)                  if (
(12)                     type(c) not in (list, tuple)
(12)                     or len(c) not in (1, 3, 4)
(12)                     or min(c) < 0
(12)                     or max(c) > 1
(8)                  ):
(12)                     raise ValueError("need 1, 3 or 4 color components in range 0 to 1")
(0)          def CheckFont(page: Page, fontname: str) -> tuple:
(4)              """Return an entry in the page's font list if reference name matches.
(4)              """
(4)              for f in page.get_fonts():
(8)                  if f[4] == fontname:
(12)                     return f
(0)          def CheckFontInfo(doc: Document, xref: int) -> list:
(4)              """Return a font info if present in the document.
(4)              """
(4)              for f in doc.FontInfos:
(8)                  if xref == f[0]:
(12)                     return f
(0)          def CheckMarkerArg(quads: typing.Any) -> tuple:
(4)              if CheckRect(quads):
(8)                  r = Rect(quads)
(8)                  return (r.quad,)
(4)              if CheckQuad(quads):
```

```
14997: (8)                  return (quads,)
14998: (4)              for q in quads:
14999: (8)                  if not (CheckRect(q) or CheckQuad(q)):
15000: (12)                     raise ValueError("bad quads entry")
15001: (4)              return quads
15002: (0)          def CheckMorph(o: typing.Any) -> bool:
15003: (4)              if not bool(o):
15004: (8)                  return False
15005: (4)              if not (type(o) in (list, tuple) and len(o) == 2):
15006: (8)                  raise ValueError("morph must be a sequence of length 2")
15007: (4)              if not (len(o[0]) == 2 and len(o[1]) == 6):
15008: (8)                  raise ValueError("invalid morph param 0")
15009: (4)              if not o[1][4] == o[1][5] == 0:
15010: (8)                  raise ValueError("invalid morph param 1")
15011: (4)              return True
15012: (0)          def CheckParent(o: typing.Any):
15013: (4)              return
15014: (4)              if not hasattr(o, "parent") or o.parent is None:
15015: (8)                  raise ValueError(f"orphaned object {type(o)=}: parent is None")
15016: (0)          def CheckQuad(q: typing.Any) -> bool:
15017: (4)              """Check whether an object is convex, not empty  quad-like.
15018: (4)              It must be a sequence of 4 number pairs.
15019: (4)              """
15020: (4)              try:
15021: (8)                  q0 = Quad(q)
15022: (4)              except Exception:
15023: (8)                  if g_exceptions_verbose > 1:    exception_info()
15024: (8)                  return False
15025: (4)              return q0.is_convex
15026: (0)          def CheckRect(r: typing.Any) -> bool:
15027: (4)              """Check whether an object is non-degenerate rect-like.
15028: (4)              It must be a sequence of 4 numbers.
15029: (4)              """
15030: (4)              try:
15031: (8)                  r = Rect(r)
15032: (4)              except Exception:
15033: (8)                  if g_exceptions_verbose > 1:    exception_info()
15034: (8)                  return False
15035: (4)              return not (r.is_empty or r.is_infinite)
15036: (0)          def ColorCode(c: typing.Union[list, tuple, float, None], f: str) -> str:
15037: (4)              if not c:
15038: (8)                  return ""
15039: (4)              if hasattr(c, "__float__"):
15040: (8)                  c = (c,)
15041: (4)              CheckColor(c)
15042: (4)              if len(c) == 1:
15043: (8)                  s = _format_g(c[0]) + " "
15044: (8)                  return s + "G " if f == "c" else s + "g "
15045: (4)              if len(c) == 3:
15046: (8)                  s = _format_g(tuple(c)) + " "
15047: (8)                  return s + "RG " if f == "c" else s + "rg "
15048: (4)              s = _format_g(tuple(c)) + " "
15049: (4)              return s + "K " if f == "c" else s + "k "
15050: (0)          def Page__add_text_marker(self, quads, annot_type):
15051: (4)              pdfpage = self._pdf_page()
15052: (4)              rotation = JM_page_rotation(pdfpage)
15053: (4)              def final():
15054: (8)                  if rotation != 0:
15055: (12)                     mupdf.pdf_dict_put_int(pdfpage.obj(), PDF_NAME('Rotate'),
rotation)
15056: (4)              try:
15057: (8)                  if rotation != 0:
15058: (12)                     mupdf.pdf_dict_put_int(pdfpage.obj(), PDF_NAME('Rotate'), 0)
15059: (8)                  annot = mupdf.pdf_create_annot(pdfpage, annot_type)
15060: (8)                  for item in quads:
15061: (12)                     q = JM_quad_from_py(item)
15062: (12)                     mupdf.pdf_add_annot_quad_point(annot, q)
15063: (8)                  mupdf.pdf_update_annot(annot)
15064: (8)                  JM_add_annot_id(annot, "A")
```

```
15065: (8)                  final()
15066: (4)              except Exception:
15067: (8)                  if g_exceptions_verbose:    exception_info()
15068: (8)                  final()
15069: (8)                  return
15070: (4)              return Annot(annot)
15071: (0)          def PDF_NAME(x):
15072: (4)              assert isinstance(x, str)
15073: (4)              return getattr(mupdf, f'PDF_ENUM_NAME_{x}')
15074: (0)          def UpdateFontInfo(doc: Document, info: typing.Sequence):
15075: (4)              xref = info[0]
15076: (4)              found = False
15077: (4)              for i, fi in enumerate(doc.FontInfos):
15078: (8)                  if fi[0] == xref:
15079: (12)                     found = True
15080: (12)                     break
15081: (4)              if found:
15082: (8)                  doc.FontInfos[i] = info
15083: (4)              else:
15084: (8)                  doc.FontInfos.append(info)
15085: (0)          def args_match(args, *types):
15086: (4)              '''
15087: (4)              Returns true if <args> matches <types>.
15088: (4)              Each item in <types> is a type or tuple of types. Any of these types will
15089: (4)              match an item in <args>. `None` will match anything in <args>.
`type(None)`
15090: (4)              will match an arg whose value is `None`.
15091: (4)              '''
15092: (4)              j = 0
15093: (4)              for i in range(len(types)):
15094: (8)                  type_ = types[i]
15095: (8)                  if j >= len(args):
15096: (12)                     if isinstance(type_, tuple) and None in type_:
15097: (16)                         continue
15098: (12)                     else:
15099: (16)                         return False
15100: (8)                  if type_ is not None and not isinstance(args[j], type_):
15101: (12)                     return False
15102: (8)                  j += 1
15103: (4)              if j != len(args):
15104: (8)                  return False
15105: (4)              return True
15106: (0)          def calc_image_matrix(width, height, tr, rotate, keep):
15107: (4)              '''
15108: (4)              '''
15109: (4)              trect = JM_rect_from_py(tr)
15110: (4)              rot = mupdf.fz_rotate(rotate)
15111: (4)              trw = trect.x1 - trect.x0
15112: (4)              trh = trect.y1 - trect.y0
15113: (4)              w = trw
15114: (4)              h = trh
15115: (4)              if keep:
15116: (8)                  large = max(width, height)
15117: (8)                  fw = width / large
15118: (8)                  fh = height / large
15119: (4)              else:
15120: (8)                  fw = fh = 1
15121: (4)              small = min(fw, fh)
15122: (4)              if rotate != 0 and rotate != 180:
15123: (8)                  f = fw
15124: (8)                  fw = fh
15125: (8)                  fh = f
15126: (4)              if fw < 1:
15127: (8)                  if trw / fw > trh / fh:
15128: (12)                     w = trh * small
15129: (12)                     h = trh
15130: (8)                  else:
15131: (12)                     w = trw
15132: (12)                     h = trw / small
```

```
15133: (4)              elif fw != fh:
15134: (8)                  if trw / fw > trh / fh:
15135: (12)                     w = trh / small
15136: (12)                     h = trh
15137: (8)                  else:
15138: (12)                     w = trw
15139: (12)                     h = trw * small
15140: (4)              else:
15141: (8)                  w = trw
15142: (8)                  h = trh
15143: (4)              tmp = mupdf.fz_make_point(
15144: (12)                     (trect.x0 + trect.x1) / 2,
15145: (12)                     (trect.y0 + trect.y1) / 2,
15146: (12)                     )
15147: (4)              mat = mupdf.fz_make_matrix(1, 0, 0, 1, -0.5, -0.5)
15148: (4)              mat = mupdf.fz_concat(mat, rot)
15149: (4)              mat = mupdf.fz_concat(mat, mupdf.fz_scale(w, h))
15150: (4)              mat = mupdf.fz_concat(mat, mupdf.fz_translate(tmp.x, tmp.y))
15151: (4)              return mat
15152: (0)          def detect_super_script(line, ch):
15153: (4)              if line.m_internal.wmode == 0 and line.m_internal.dir.x == 1 and
line.m_internal.dir.y == 0:
15154: (8)                  return ch.m_internal.origin.y < line.m_internal.first_char.origin.y -
ch.m_internal.size * 0.1
15155: (4)              return 0
15156: (0)          def dir_str(x):
15157: (4)              ret = f'{x} {type(x)} ({len(dir(x))}):\n'
15158: (4)              for i in dir(x):
15159: (8)                  ret += f'    {i}\n'
15160: (4)              return ret
15161: (0)          def getTJstr(text: str, glyphs: typing.Union[list, tuple, None], simple: bool,
ordering: int) -> str:
15162: (4)              """ Return a PDF string enclosed in [] brackets, suitable for the PDF TJ
15163: (4)              operator.
15164: (4)              Notes:
15165: (8)                  The input string is converted to either 2 or 4 hex digits per
character.
15166: (4)              Args:
15167: (8)                  simple: no glyphs: 2-chars, use char codes as the glyph
15168: (16)                         glyphs: 2-chars, use glyphs instead of char codes (Symbol,
15169: (16)                         ZapfDingbats)
15170: (8)                  not simple: ordering < 0: 4-chars, use glyphs not char codes
15171: (20)                             ordering >=0: a CJK font! 4 chars, use char codes as
glyphs
15172: (4)              """
15173: (4)              if text.startswith("[<") and text.endswith(">]"):  # already done
15174: (8)                  return text
15175: (4)              if not bool(text):
15176: (8)                  return "[<>]"
15177: (4)              if simple:  # each char or its glyph is coded as a 2-byte hex
15178: (8)                  if glyphs is None:  # not Symbol, not ZapfDingbats: use char code
15179: (12)                     otxt = "".join(["%02x" % ord(c) if ord(c) < 256 else "b7" for c in
text])
15180: (8)                  else:  # Symbol or ZapfDingbats: use glyphs
15181: (12)                     otxt = "".join(
15182: (16)                         ["%02x" % glyphs[ord(c)][0] if ord(c) < 256 else "b7" for c in
text]
15183: (12)                     )
15184: (8)                  return "[<" + otxt + ">]"
15185: (4)              if ordering < 0:  # not a CJK font: use the glyphs
15186: (8)                  otxt = "".join(["%04x" % glyphs[ord(c)][0] for c in text])
15187: (4)              else:  # CJK: use the char codes
15188: (8)                  otxt = "".join(["%04x" % ord(c) for c in text])
15189: (4)              return "[<" + otxt + ">]"
15190: (0)          def get_pdf_str(s: str) -> str:
15191: (4)              """ Return a PDF string depending on its coding.
15192: (4)              Notes:
15193: (8)                  Returns a string bracketed with either "()" or "<>" for hex values.
15194: (8)                  If only ascii then "(original)" is returned, else if only 8 bit chars
```

```
15195: (8)                  then "(original)" with interspersed octal strings \nnn is returned,
15196: (8)                  else a string "<FEFF[hexstring]>" is returned, where [hexstring] is
the
15197: (8)                  UTF-16BE encoding of the original.
15198: (4)              """
15199: (4)              if not bool(s):
15200: (8)                  return "()"
15201: (4)              def make_utf16be(s):
15202: (8)                  r = bytearray([254, 255]) + bytearray(s, "UTF-16BE")
15203: (8)                  return "<" + r.hex() + ">"  # brackets indicate hex
15204: (4)              r = ""
15205: (4)              for c in s:
15206: (8)                  oc = ord(c)
15207: (8)                  if oc > 255:  # shortcut if beyond 8-bit code range
15208: (12)                     return make_utf16be(s)
15209: (8)                  if oc > 31 and oc < 127:  # in ASCII range
15210: (12)                     if c in ("(", ")", "\\"):  # these need to be escaped
15211: (16)                         r += "\\"
15212: (12)                     r += c
15213: (12)                     continue
15214: (8)                  if oc > 127:  # beyond ASCII
15215: (12)                     r += "\\%03o" % oc
15216: (12)                     continue
15217: (8)                  if oc == 8:  # backspace
15218: (12)                     r += "\\b"
15219: (8)                  elif oc == 9:  # tab
15220: (12)                     r += "\\t"
15221: (8)                  elif oc == 10:  # line feed
15222: (12)                     r += "\\n"
15223: (8)                  elif oc == 12:  # form feed
15224: (12)                     r += "\\f"
15225: (8)                  elif oc == 13:  # carriage return
15226: (12)                     r += "\\r"
15227: (8)                  else:
15228: (12)                     r += "\\267"  # unsupported: replace by 0xB7
15229: (4)              return "(" + r + ")"
15230: (0)          def get_tessdata():
15231: (4)              """Detect Tesseract-OCR and return its language support folder.
15232: (4)              This function can be used to enable OCR via Tesseract even if the
15233: (4)              environment variable TESSDATA_PREFIX has not been set.
15234: (4)              If the value of TESSDATA_PREFIX is None, the function tries to locate
15235: (4)              Tesseract-OCR and fills the required variable.
15236: (4)              Returns:
15237: (8)                  Folder name of tessdata if Tesseract-OCR is available, otherwise
False.
15238: (4)              """
15239: (4)              TESSDATA_PREFIX = os.getenv("TESSDATA_PREFIX")
15240: (4)              if TESSDATA_PREFIX:  # use environment variable if set
15241: (8)                  return TESSDATA_PREFIX
15242: (4)              """
15243: (4)              Try to locate the tesseract-ocr installation.
15244: (4)              """
15245: (4)              import subprocess
15246: (4)              if sys.platform == "win32":
15247: (8)                  cp = subprocess.run("where tesseract", shell=1, capture_output=1,
check=0, text=True)
15248: (8)                  response = cp.stdout.strip()
15249: (8)                  if cp.returncode or not response:
15250: (12)                     message("Tesseract-OCR is not installed")
15251: (12)                     return False
15252: (8)                  dirname = os.path.dirname(response)  # path of tesseract.exe
15253: (8)                  tessdata = os.path.join(dirname, "tessdata")  # language support
15254: (8)                  if os.path.exists(tessdata):  # all ok?
15255: (12)                     return tessdata
15256: (8)                  else:  # should not happen!
15257: (12)                     message("unexpected: Tesseract-OCR has no 'tessdata' folder")
15258: (12)                     return False
15259: (4)              cp = subprocess.run("whereis tesseract-ocr", shell=1, capture_output=1,
check=0, text=True)
```

```
15260: (4)              response = cp.stdout.strip().split()
15261: (4)              if cp.returncode or len(response) != 2:  # if not 2 tokens: no tesseract-
ocr
15262: (8)                  message("tesseract-ocr is not installed")
15263: (8)                  return False
15264: (4)              dirname = response[1]  # contains tesseract-ocr installation folder
15265: (4)              tessdatas = glob.glob(f"{dirname}/*/tessdata")
15266: (4)              tessdatas.sort()
15267: (4)              if len(tessdatas) == 0:
15268: (8)                  message("unexpected: tesseract-ocr has no 'tessdata' folder")
15269: (8)                  return False
15270: (4)              return tessdatas[-1]
15271: (0)          def css_for_pymupdf_font(
15272: (4)              fontcode: str, *, CSS: OptStr = None, archive: AnyType = None, name:
OptStr = None
15273: (0)          ) -> str:
15274: (4)              """Create @font-face items for the given fontcode of pymupdf-fonts.
15275: (4)              Adds @font-face support for fonts contained in package pymupdf-fonts.
15276: (4)              Creates a CSS font-family for all fonts starting with string 'fontcode'.
15277: (4)              Note:
15278: (8)                  The font naming convention in package pymupdf-fonts is "fontcode<sf>",
15279: (8)                  where the suffix "sf" is either empty or one of "it", "bo" or "bi".
15280: (8)                  These suffixes thus represent the regular, italic, bold or bold-italic
15281: (8)                  variants of a font. For example, font code "notos" refers to fonts
15282: (8)                  "notos" - "Noto Sans Regular"
15283: (8)                  "notosit" - "Noto Sans Italic"
15284: (8)                  "notosbo" - "Noto Sans Bold"
15285: (8)                  "notosbi" - "Noto Sans Bold Italic"
15286: (8)                  This function creates four CSS @font-face definitions and collectively
15287: (8)                  assigns the font-family name "notos" to them (or the "name" value).
15288: (4)              All fitting font buffers of the pymupdf-fonts package are placed / added
15289: (4)              to the archive provided as parameter.
15290: (4)              To use the font in pymupdf.Story, execute 'set_font(fontcode)'. The
correct
15291: (4)              font weight (bold) or style (italic) will automatically be selected.
15292: (4)              Expects and returns the CSS source, with the new CSS definitions appended.
15293: (4)              Args:
15294: (8)                  fontcode: (str) font code for naming the font variants to include.
15295: (18)                           E.g. "fig" adds notos, notosi, notosb, notosbi fonts.
15296: (18)                           A maximum of 4 font variants is accepted.
15297: (8)                  CSS: (str) CSS string to add @font-face definitions to.
15298: (8)                  archive: (Archive, mandatory) where to place the font buffers.
15299: (8)                  name: (str) use this as family-name instead of 'fontcode'.
15300: (4)              Returns:
15301: (8)                  Modified CSS, with appended @font-face statements for each font
variant
15302: (8)                  of fontcode.
15303: (8)                  Fontbuffers associated with "fontcode" will be added to 'archive'.
15304: (4)              """
15305: (4)              CSSFONT = "\n@font-face {font-family: %s; src: url(%s);%s%s}\n"
15306: (4)              if not type(archive) is Archive:
15307: (8)                  raise ValueError("'archive' must be an Archive")
15308: (4)              if CSS is None:
15309: (8)                  CSS = ""
15310: (4)              font_keys = [k for k in fitz_fontdescriptors.keys() if
k.startswith(fontcode)]
15311: (4)              if font_keys == []:
15312: (8)                  raise ValueError(f"No font code '{fontcode}' found in pymupdf-fonts.")
15313: (4)              if len(font_keys) > 4:
15314: (8)                  raise ValueError("fontcode too short")
15315: (4)              if name is None:  # use this name for font-family
15316: (8)                  name = fontcode
15317: (4)              for fkey in font_keys:
15318: (8)                  font = fitz_fontdescriptors[fkey]
15319: (8)                  bold = font["bold"]  # determine font property
15320: (8)                  italic = font["italic"]  # determine font property
15321: (8)                  fbuff = font["loader"]()  # load the fontbuffer
15322: (8)                  archive.add(fbuff, fkey)  # update the archive
15323: (8)                  bold_text = "font-weight: bold;" if bold else ""
```

```
15324: (8)                  italic_text = "font-style: italic;" if italic else ""
15325: (8)                  CSS += CSSFONT % (name, fkey, bold_text, italic_text)
15326: (4)              return CSS
15327: (0)          def get_text_length(text: str, fontname: str ="helv", fontsize: float =11,
encoding: int =0) -> float:
15328: (4)              """Calculate length of a string for a built-in font.
15329: (4)              Args:
15330: (8)                  fontname: name of the font.
15331: (8)                  fontsize: font size points.
15332: (8)                  encoding: encoding to use, 0=Latin (default), 1=Greek, 2=Cyrillic.
15333: (4)              Returns:
15334: (8)                  (float) length of text.
15335: (4)              """
15336: (4)              fontname = fontname.lower()
15337: (4)              basename = Base14_fontdict.get(fontname, None)
15338: (4)              glyphs = None
15339: (4)              if basename == "Symbol":
15340: (8)                  glyphs = symbol_glyphs
15341: (4)              if basename == "ZapfDingbats":
15342: (8)                  glyphs = zapf_glyphs
15343: (4)              if glyphs is not None:
15344: (8)                  w = sum([glyphs[ord(c)][1] if ord(c) < 256 else glyphs[183][1] for c
in text])
15345: (8)                  return w * fontsize
15346: (4)              if fontname in Base14_fontdict.keys():
15347: (8)                  return util_measure_string(
15348: (12)                     text, Base14_fontdict[fontname], fontsize, encoding
15349: (8)                  )
15350: (4)              if fontname in (
15351: (8)                  "china-t",
15352: (8)                  "china-s",
15353: (8)                  "china-ts",
15354: (8)                  "china-ss",
15355: (8)                  "japan",
15356: (8)                  "japan-s",
15357: (8)                  "korea",
15358: (8)                  "korea-s",
15359: (4)              ):
15360: (8)                  return len(text) * fontsize
15361: (4)              raise ValueError("Font '%s' is unsupported" % fontname)
15362: (0)          def image_profile(img: typing.ByteString) -> dict:
15363: (4)              """ Return basic properties of an image.
15364: (4)              Args:
15365: (8)                  img: bytes, bytearray, io.BytesIO object or an opened image file.
15366: (4)              Returns:
15367: (8)                  A dictionary with keys width, height, colorspace.n, bpc, type, ext and
size,
15368: (8)                  where 'type' is the MuPDF image type (0 to 14) and 'ext' the suitable
15369: (8)                  file extension.
15370: (4)              """
15371: (4)              if type(img) is io.BytesIO:
15372: (8)                  stream = img.getvalue()
15373: (4)              elif hasattr(img, "read"):
15374: (8)                  stream = img.read()
15375: (4)              elif type(img) in (bytes, bytearray):
15376: (8)                  stream = img
15377: (4)              else:
15378: (8)                  raise ValueError("bad argument 'img'")
15379: (4)              return TOOLS.image_profile(stream)
15380: (0)          def jm_append_merge(dev):
15381: (4)              '''
15382: (4)              Append current path to list or merge into last path of the list.
15383: (4)              (1) Append if first path, different item lists or not a 'stroke' version
15384: (8)                  of previous path
15385: (4)              (2) If new path has the same items, merge its content into previous path
15386: (8)                  and change path["type"] to "fs".
15387: (4)              (3) If "out" is callable, skip the previous and pass dictionary to it.
15388: (4)              '''
15389: (4)              assert isinstance(dev.out, list)
```

```
15390: (4)              if callable(dev.method) or dev.method:  # function or method
15391: (8)                  if dev.method is None:
15392: (12)                     assert 0
15393: (8)                  else:
15394: (12)                     resp = getattr(dev.out, dev.method)(dev.pathdict)
15395: (8)                  if not resp:
15396: (12)                     message("calling cdrawings callback function/method failed!")
15397: (8)                  dev.pathdict = None
15398: (8)                  return
15399: (4)              def append():
15400: (8)                  dev.out.append(dev.pathdict.copy())
15401: (8)                  dev.pathdict.clear()
15402: (4)              assert isinstance(dev.out, list)
15403: (4)              len_ = len(dev.out) # len of output list so far
15404: (4)              if len_ == 0:   # always append first path
15405: (8)                  return append()
15406: (4)              thistype = dev.pathdict[ dictkey_type]
15407: (4)              if thistype != 's': # if not stroke, then append
15408: (8)                  return append()
15409: (4)              prev = dev.out[ len_-1] # get prev path
15410: (4)              prevtype = prev[ dictkey_type]
15411: (4)              if prevtype != 'f': # if previous not fill, append
15412: (8)                  return append()
15413: (4)              previtems = prev[ dictkey_items]
15414: (4)              thisitems = dev.pathdict[ dictkey_items]
15415: (4)              if previtems != thisitems:
15416: (8)                  return append()
15417: (4)              try:
15418: (8)                  for k, v in dev.pathdict.items():
15419: (12)                     if k not in prev:
15420: (16)                         prev[k] = v
15421: (8)                  rc = 0
15422: (4)              except Exception:
15423: (8)                  if g_exceptions_verbose:    exception_info()
15424: (8)                  rc = -1
15425: (4)              if rc == 0:
15426: (8)                  prev[ dictkey_type] = 'fs'
15427: (8)                  dev.pathdict.clear()
15428: (4)              else:
15429: (8)                  message("could not merge stroke and fill path")
15430: (8)                  append()
15431: (0)          def jm_bbox_add_rect( dev, ctx, rect, code):
15432: (4)              if not dev.layers:
15433: (8)                  dev.result.append( (code, JM_py_from_rect(rect)))
15434: (4)              else:
15435: (8)                  dev.result.append( (code, JM_py_from_rect(rect), dev.layer_name))
15436: (0)          def jm_bbox_fill_image( dev, ctx, image, ctm, alpha, color_params):
15437: (4)              r = mupdf.FzRect(mupdf.FzRect.Fixed_UNIT)
15438: (4)              r = mupdf.ll_fz_transform_rect( r.internal(), ctm)
15439: (4)              jm_bbox_add_rect( dev, ctx, r, "fill-image")
15440: (0)          def jm_bbox_fill_image_mask( dev, ctx, image, ctm, colorspace, color, alpha,
color_params):
15441: (4)              try:
15442: (8)                  jm_bbox_add_rect( dev, ctx,
mupdf.ll_fz_transform_rect(mupdf.fz_unit_rect, ctm), "fill-imgmask")
15443: (4)              except Exception:
15444: (8)                  if g_exceptions_verbose:    exception_info()
15445: (8)                  raise
15446: (0)          def jm_bbox_fill_path( dev, ctx, path, even_odd, ctm, colorspace, color,
alpha, color_params):
15447: (4)              even_odd = True if even_odd else False
15448: (4)              try:
15449: (8)                  jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_path(path, None, ctm),
"fill-path")
15450: (4)              except Exception:
15451: (8)                  if g_exceptions_verbose:    exception_info()
15452: (8)                  raise
15453: (0)          def jm_bbox_fill_shade( dev, ctx, shade, ctm, alpha, color_params):
15454: (4)              try:
```

```
15455: (8)                  jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_shade( shade, ctm),
"fill-shade")
15456: (4)              except Exception:
15457: (8)                  if g_exceptions_verbose:    exception_info()
15458: (8)                  raise
15459: (0)          def jm_bbox_stroke_text( dev, ctx, text, stroke, ctm, *args):
15460: (4)              try:
15461: (8)                  jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_text( text, stroke,
ctm), "stroke-text")
15462: (4)              except Exception:
15463: (8)                  if g_exceptions_verbose:    exception_info()
15464: (8)                  raise
15465: (0)          def jm_bbox_fill_text( dev, ctx, text, ctm, *args):
15466: (4)              try:
15467: (8)                  jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_text( text, None, ctm),
"fill-text")
15468: (4)              except Exception:
15469: (8)                  if g_exceptions_verbose:    exception_info()
15470: (8)                  raise
15471: (0)          def jm_bbox_ignore_text( dev, ctx, text, ctm):
15472: (4)              jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_text(text, None, ctm),
"ignore-text")
15473: (0)          def jm_bbox_stroke_path( dev, ctx, path, stroke, ctm, colorspace, color,
alpha, color_params):
15474: (4)              try:
15475: (8)                  jm_bbox_add_rect( dev, ctx, mupdf.ll_fz_bound_path( path, stroke,
ctm), "stroke-path")
15476: (4)              except Exception:
15477: (8)                  if g_exceptions_verbose:    exception_info()
15478: (8)                  raise
15479: (0)          def jm_checkquad(dev):
15480: (4)              '''
15481: (4)              Check whether the last 4 lines represent a quad.
15482: (4)              Because of how we count, the lines are a polyline already, i.e. last point
15483: (4)              of a line equals 1st point of next line.
15484: (4)              So we check for a polygon (last line's end point equals start point).
15485: (4)              If not true we return 0.
15486: (4)              '''
15487: (4)              items = dev.pathdict[ dictkey_items]
15488: (4)              len_ = len(items)
15489: (4)              f = [0] * 8 # coordinates of the 4 corners
15490: (4)              for i in range( 4): # store line start points
15491: (8)                  line = items[ len_ - 4 + i]
15492: (8)                  temp = JM_point_from_py( line[1])
15493: (8)                  f[i * 2] = temp.x
15494: (8)                  f[i * 2 + 1] = temp.y
15495: (8)                  lp = JM_point_from_py( line[ 2])
15496: (4)              if lp.x != f[0] or lp.y != f[1]:
15497: (8)                  return 0
15498: (4)              dev.linecount = 0   # reset this
15499: (4)              q = mupdf.fz_make_quad(f[0], f[1], f[6], f[7], f[2], f[3], f[4], f[5])
15500: (4)              rect = ('qu', JM_py_from_quad(q))
15501: (4)              items[ len_ - 4] = rect  # replace item -4 by rect
15502: (4)              del items[ len_ - 3 : len_]  # delete remaining 3 items
15503: (4)              return 1
15504: (0)          def jm_checkrect(dev):
15505: (4)              '''
15506: (4)              Check whether the last 3 path items represent a rectangle.
15507: (4)              Returns 1 if we have modified the path, otherwise 0.
15508: (4)              '''
15509: (4)              dev.linecount = 0   # reset line count
15510: (4)              orientation = 0 # area orientation of rectangle
15511: (4)              items = dev.pathdict[ dictkey_items]
15512: (4)              len_ = len(items)
15513: (4)              line0 = items[ len_ - 3]
15514: (4)              ll = JM_point_from_py( line0[ 1])
15515: (4)              lr = JM_point_from_py( line0[ 2])
15516: (4)              line2 = items[ len_ - 1]
15517: (4)              ur = JM_point_from_py( line2[ 1])
```

```
15518: (4)              ul = JM_point_from_py( line2[ 2])
15519: (4)              if (0
15520: (12)                     or ll.y != lr.y
15521: (12)                     or ll.x != ul.x
15522: (12)                     or ur.y != ul.y
15523: (12)                     or ur.x != lr.x
15524: (12)                     ):
15525: (8)                  return 0 # not a rectangle
15526: (4)              if ul.y < lr.y:
15527: (8)                  r = mupdf.fz_make_rect(ul.x, ul.y, lr.x, lr.y)
15528: (8)                  orientation = 1
15529: (4)              else:
15530: (8)                  r = mupdf.fz_make_rect(ll.x, ll.y, ur.x, ur.y)
15531: (8)                  orientation = -1
15532: (4)              rect = ( 're', JM_py_from_rect(r), orientation)
15533: (4)              items[ len_ - 3] = rect # replace item -3 by rect
15534: (4)              del items[ len_ - 2 : len_] # delete remaining 2 items
15535: (4)              return 1
15536: (0)          def jm_trace_text( dev, text, type_, ctm, colorspace, color, alpha, seqno):
15537: (4)              span = text.head
15538: (4)              while 1:
15539: (8)                  if not span:
15540: (12)                     break
15541: (8)                  jm_trace_text_span( dev, span, type_, ctm, colorspace, color, alpha,
seqno)
15542: (8)                  span = span.next
15543: (0)          def jm_trace_text_span(dev, span, type_, ctm, colorspace, color, alpha,
seqno):
15544: (4)              '''
15545: (4)              jm_trace_text_span(fz_context *ctx, PyObject *out, fz_text_span *span, int
type, fz_matrix ctm, fz_colorspace *colorspace, const float *color, float alpha, size_t seqno)
15546: (4)              '''
15547: (4)              out_font = None
15548: (4)              assert isinstance( span, mupdf.fz_text_span)
15549: (4)              span = mupdf.FzTextSpan( span)
15550: (4)              assert isinstance( ctm, mupdf.fz_matrix)
15551: (4)              ctm = mupdf.FzMatrix( ctm)
15552: (4)              fontname = JM_font_name( span.font())
15553: (4)              mat = mupdf.fz_concat(span.trm(), ctm)  # text transformation matrix
15554: (4)              dir = mupdf.fz_transform_vector(mupdf.fz_make_point(1, 0), mat) # writing
direction
15555: (4)              fsize = math.sqrt(dir.x * dir.x + dir.y * dir.y) # font size
15556: (4)              dir = mupdf.fz_normalize_vector(dir)
15557: (4)              space_adv = 0
15558: (4)              asc = JM_font_ascender( span.font())
15559: (4)              dsc = JM_font_descender( span.font())
15560: (4)              if asc < 1e-3:  # probably Tesseract font
15561: (8)                  dsc = -0.1
15562: (8)                  asc = 0.9
15563: (4)              ascsize = asc * fsize / (asc - dsc)
15564: (4)              dscsize = dsc * fsize / (asc - dsc)
15565: (4)              fflags = 0  # font flags
15566: (4)              mono = mupdf.fz_font_is_monospaced( span.font())
15567: (4)              fflags += mono * TEXT_FONT_MONOSPACED
15568: (4)              fflags += mupdf.fz_font_is_italic( span.font()) * TEXT_FONT_ITALIC
15569: (4)              fflags += mupdf.fz_font_is_serif( span.font()) * TEXT_FONT_SERIFED
15570: (4)              fflags += mupdf.fz_font_is_bold( span.font()) * TEXT_FONT_BOLD
15571: (4)              last_adv = 0
15572: (4)              span_bbox = mupdf.FzRect()
15573: (4)              rot = mupdf.fz_make_matrix(dir.x, dir.y, -dir.y, dir.x, 0, 0)
15574: (4)              if dir.x == -1: # left-right flip
15575: (8)                  rot.d = 1
15576: (4)              chars = []
15577: (4)              for i in range( span.m_internal.len):
15578: (8)                  adv = 0
15579: (8)                  if span.items(i).gid >= 0:
15580: (12)                     adv = mupdf.fz_advance_glyph( span.font(), span.items(i).gid,
span.m_internal.wmode)
15581: (8)                  adv *= fsize
```

```
15582: (8)                  last_adv = adv
15583: (8)                  if span.items(i).ucs == 32:
15584: (12)                     space_adv = adv
15585: (8)                  char_orig = mupdf.fz_make_point(span.items(i).x, span.items(i).y)
15586: (8)                  char_orig = mupdf.fz_transform_point(char_orig, ctm)
15587: (8)                  m1 = mupdf.fz_make_matrix(1, 0, 0, 1, -char_orig.x, -char_orig.y)
15588: (8)                  m1 = mupdf.fz_concat(m1, rot)
15589: (8)                  m1 = mupdf.fz_concat(m1, mupdf.FzMatrix(1, 0, 0, 1, char_orig.x,
char_orig.y))
15590: (8)                  x0 = char_orig.x
15591: (8)                  x1 = x0 + adv
15592: (8)                  if (
15593: (16)                         (mat.d > 0 and (dir.x == 1 or dir.x == -1))
15594: (16)                         or
15595: (16)                         (mat.b != 0 and mat.b == -mat.c)
15596: (16)                         ):  # up-down flip
15597: (12)                     y0 = char_orig.y + dscsize
15598: (12)                     y1 = char_orig.y + ascsize
15599: (8)                  else:
15600: (12)                     y0 = char_orig.y - ascsize
15601: (12)                     y1 = char_orig.y - dscsize
15602: (8)                  char_bbox = mupdf.fz_make_rect(x0, y0, x1, y1)
15603: (8)                  char_bbox = mupdf.fz_transform_rect(char_bbox, m1)
15604: (8)                  chars.append(
15605: (16)                         (
15606: (20)                             span.items(i).ucs,
15607: (20)                             span.items(i).gid,
15608: (20)                             (
15609: (24)                                 char_orig.x,
15610: (24)                                 char_orig.y,
15611: (20)                             ),
15612: (20)                             (
15613: (24)                                 char_bbox.x0,
15614: (24)                                 char_bbox.y0,
15615: (24)                                 char_bbox.x1,
15616: (24)                                 char_bbox.y1,
15617: (20)                             ),
15618: (16)                         )
15619: (16)                         )
15620: (8)                  if i > 0:
15621: (12)                     span_bbox = mupdf.fz_union_rect(span_bbox, char_bbox)
15622: (8)                  else:
15623: (12)                     span_bbox = char_bbox
15624: (4)              chars = tuple(chars)
15625: (4)              if not space_adv:
15626: (8)                  if not mono:
15627: (12)                     c, out_font = mupdf.fz_encode_character_with_fallback(
span.font(), 32, 0, 0)
15628: (12)                     space_adv = mupdf.fz_advance_glyph(
15629: (20)                             span.font(),
15630: (20)                             c,
15631: (20)                             span.m_internal.wmode,
15632: (20)                             )
15633: (12)                     space_adv *= fsize
15634: (12)                     if not space_adv:
15635: (16)                         space_adv = last_adv
15636: (8)                  else:
15637: (12)                     space_adv = last_adv    # for mono, any char width suffices
15638: (4)              span_dict = dict()
15639: (4)              span_dict[ 'dir'] = JM_py_from_point(dir)
15640: (4)              span_dict[ 'font'] = JM_EscapeStrFromStr(fontname)
15641: (4)              span_dict[ 'wmode'] = span.m_internal.wmode
15642: (4)              span_dict[ 'flags'] =fflags
15643: (4)              span_dict[ "bidi_lvl"] =span.m_internal.bidi_level
15644: (4)              span_dict[ "bidi_dir"] = span.m_internal.markup_dir
15645: (4)              span_dict[ 'ascender'] = asc
15646: (4)              span_dict[ 'descender'] = dsc
15647: (4)              span_dict[ 'colorspace'] = 3
15648: (4)              if colorspace:
```

```
15649: (8)                  rgb = mupdf.fz_convert_color(
15650: (16)                         mupdf.FzColorspace( mupdf.ll_fz_keep_colorspace( colorspace)),
15651: (16)                         color,
15652: (16)                         mupdf.fz_device_rgb(),
15653: (16)                         mupdf.FzColorspace(),
15654: (16)                         mupdf.FzColorParams(),
15655: (16)                         )
15656: (8)                  rgb = rgb[:3]   # mupdf.fz_convert_color() always returns 4 items.
15657: (4)              else:
15658: (8)                  rgb = (0, 0, 0)
15659: (4)              if dev.linewidth > 0:   # width of character border
15660: (8)                  linewidth = dev.linewidth
15661: (4)              else:
15662: (8)                  linewidth = fsize * 0.05    # default: 5% of font size
15663: (4)              span_dict[ 'color'] = rgb
15664: (4)              span_dict[ 'size'] = fsize
15665: (4)              span_dict[ "opacity"] = alpha
15666: (4)              span_dict[ "linewidth"] = linewidth
15667: (4)              span_dict[ "spacewidth"] = space_adv
15668: (4)              span_dict[ 'type'] = type_
15669: (4)              span_dict[ 'bbox'] = JM_py_from_rect(span_bbox)
15670: (4)              span_dict[ 'layer'] = dev.layer_name
15671: (4)              span_dict[ "seqno"] = seqno
15672: (4)              span_dict[ 'chars'] = chars
15673: (4)              dev.out.append( span_dict)
15674: (0)          def jm_lineart_color(colorspace, color):
15675: (4)              if colorspace:
15676: (8)                  try:
15677: (12)                     cs = mupdf.FzColorspace( mupdf.FzColorspace.Fixed_RGB)
15678: (12)                     cp = mupdf.FzColorParams()
15679: (12)                     rgb = mupdf.ll_fz_convert_color(
15680: (20)                             colorspace,
15681: (20)                             color,
15682: (20)                             cs.m_internal,
15683: (20)                             None,
15684: (20)                             cp.internal(),
15685: (20)                             )
15686: (8)                  except Exception:
15687: (12)                     if g_exceptions_verbose:    exception_info()
15688: (12)                     raise
15689: (8)                  return rgb[:3]
15690: (4)              return ()
15691: (0)          def jm_lineart_drop_device(dev, ctx):
15692: (4)              if isinstance(dev.out, list):
15693: (8)                  dev.out = []
15694: (4)              dev.scissors = []
15695: (0)          def jm_lineart_fill_path( dev, ctx, path, even_odd, ctm, colorspace, color,
alpha, color_params):
15696: (4)              even_odd = True if even_odd else False
15697: (4)              try:
15698: (8)                  assert isinstance( ctm, mupdf.fz_matrix)
15699: (8)                  dev.ctm = mupdf.FzMatrix( ctm)  # fz_concat(ctm, dev_ptm);
15700: (8)                  dev.path_type = trace_device_FILL_PATH
15701: (8)                  jm_lineart_path( dev, ctx, path)
15702: (8)                  if dev.pathdict is None:
15703: (12)                     return
15704: (8)                  dev.pathdict[ dictkey_type] ="f"
15705: (8)                  dev.pathdict[ "even_odd"] = even_odd
15706: (8)                  dev.pathdict[ "fill_opacity"] = alpha
15707: (8)                  dev.pathdict[ "fill"] = jm_lineart_color( colorspace, color)
15708: (8)                  dev.pathdict[ dictkey_rect] = JM_py_from_rect(dev.pathrect)
15709: (8)                  dev.pathdict[ "seqno"] = dev.seqno
15710: (8)                  dev.pathdict[ 'layer'] = dev.layer_name
15711: (8)                  if dev.clips:
15712: (12)                     dev.pathdict[ 'level'] = dev.depth
15713: (8)                  jm_append_merge(dev)
15714: (8)                  dev.seqno += 1
15715: (4)              except Exception:
15716: (8)                  if g_exceptions_verbose:    exception_info()
```

```
15717: (8)                  raise
15718: (0)          def jm_lineart_fill_text( dev, ctx, text, ctm, colorspace, color, alpha,
color_params):
15719: (4)              if 0:
15720: (8)                  log(f'{type(ctx)=} {ctx=}')
15721: (8)                  log(f'{type(dev)=} {dev=}')
15722: (8)                  log(f'{type(text)=} {text=}')
15723: (8)                  log(f'{type(ctm)=} {ctm=}')
15724: (8)                  log(f'{type(colorspace)=} {colorspace=}')
15725: (8)                  log(f'{type(color)=} {color=}')
15726: (8)                  log(f'{type(alpha)=} {alpha=}')
15727: (8)                  log(f'{type(color_params)=} {color_params=}')
15728: (4)              jm_trace_text(dev, text, 0, ctm, colorspace, color, alpha, dev.seqno)
15729: (4)              dev.seqno += 1
15730: (0)          def jm_lineart_ignore_text(dev, text, ctm):
15731: (4)              jm_trace_text(dev, text, 3, ctm, None, None, 1, dev.seqno)
15732: (4)              dev.seqno += 1
15733: (0)          class Walker(mupdf.FzPathWalker2):
15734: (4)              def __init__(self, dev):
15735: (8)                  super().__init__()
15736: (8)                  self.use_virtual_moveto()
15737: (8)                  self.use_virtual_lineto()
15738: (8)                  self.use_virtual_curveto()
15739: (8)                  self.use_virtual_closepath()
15740: (8)                  self.dev = dev
15741: (4)              def closepath(self, ctx):    # trace_close().
15742: (8)                  try:
15743: (12)                     if self.dev.linecount == 3:
15744: (16)                         if jm_checkrect(self.dev):
15745: (20)                             return
15746: (12)                     self.dev.linecount = 0   # reset # of consec. lines
15747: (12)                     if self.dev.havemove:
15748: (16)                         if self.dev.lastpoint != self.dev.firstpoint:
15749: (20)                             item = ("l", JM_py_from_point(self.dev.lastpoint),
15750: (33)                                          JM_py_from_point(self.dev.firstpoint))
15751: (20)                             self.dev.pathdict[dictkey_items].append(item)
15752: (20)                             self.dev.lastpoint = self.dev.firstpoint
15753: (16)                         self.dev.pathdict["closePath"] = False
15754: (12)                     else:
15755: (16)                         self.dev.pathdict[ "closePath"] = True
15756: (12)                     self.dev.havemove = 0
15757: (8)                  except Exception:
15758: (12)                     if g_exceptions_verbose:    exception_info()
15759: (12)                     raise
15760: (4)              def curveto(self, ctx, x1, y1, x2, y2, x3, y3):   # trace_curveto().
15761: (8)                  try:
15762: (12)                     self.dev.linecount = 0  # reset # of consec. lines
15763: (12)                     p1 = mupdf.fz_make_point(x1, y1)
15764: (12)                     p2 = mupdf.fz_make_point(x2, y2)
15765: (12)                     p3 = mupdf.fz_make_point(x3, y3)
15766: (12)                     p1 = mupdf.fz_transform_point(p1, self.dev.ctm)
15767: (12)                     p2 = mupdf.fz_transform_point(p2, self.dev.ctm)
15768: (12)                     p3 = mupdf.fz_transform_point(p3, self.dev.ctm)
15769: (12)                     self.dev.pathrect =
mupdf.fz_include_point_in_rect(self.dev.pathrect, p1)
15770: (12)                     self.dev.pathrect =
mupdf.fz_include_point_in_rect(self.dev.pathrect, p2)
15771: (12)                     self.dev.pathrect =
mupdf.fz_include_point_in_rect(self.dev.pathrect, p3)
15772: (12)                     list_ = (
15773: (20)                             "c",
15774: (20)                             JM_py_from_point(self.dev.lastpoint),
15775: (20)                             JM_py_from_point(p1),
15776: (20)                             JM_py_from_point(p2),
15777: (20)                             JM_py_from_point(p3),
15778: (20)                             )
15779: (12)                     self.dev.lastpoint = p3
15780: (12)                     self.dev.pathdict[ dictkey_items].append( list_)
15781: (8)                  except Exception:
```

```
15782: (12)                     if g_exceptions_verbose:    exception_info()
15783: (12)                     raise
15784: (4)              def lineto(self, ctx, x, y):   # trace_lineto().
15785: (8)                  try:
15786: (12)                     p1 = mupdf.fz_transform_point( mupdf.fz_make_point(x, y),
self.dev.ctm)
15787: (12)                     self.dev.pathrect = mupdf.fz_include_point_in_rect(
self.dev.pathrect, p1)
15788: (12)                     list_ = (
15789: (20)                             'l',
15790: (20)                             JM_py_from_point( self.dev.lastpoint),
15791: (20)                             JM_py_from_point(p1),
15792: (20)                             )
15793: (12)                     self.dev.lastpoint = p1
15794: (12)                     items = self.dev.pathdict[ dictkey_items]
15795: (12)                     items.append( list_)
15796: (12)                     self.dev.linecount += 1 # counts consecutive lines
15797: (12)                     if self.dev.linecount == 4 and self.dev.path_type !=
trace_device_FILL_PATH:
15798: (16)                         jm_checkquad(self.dev)
15799: (8)                  except Exception:
15800: (12)                     if g_exceptions_verbose:    exception_info()
15801: (12)                     raise
15802: (4)              def moveto(self, ctx, x, y):   # trace_moveto().
15803: (8)                  if 0 and isinstance(self.dev.pathdict, dict):
15804: (12)                     log(f'self.dev.pathdict:')
15805: (12)                     for n, v in self.dev.pathdict.items():
15806: (16)                         log( '    {type(n)=} {len(n)=} {n!r} {n}: {v!r}: {v}')
15807: (8)                  try:
15808: (12)                     self.dev.lastpoint = mupdf.fz_transform_point(
15809: (20)                             mupdf.fz_make_point(x, y),
15810: (20)                             self.dev.ctm,
15811: (20)                             )
15812: (12)                     if mupdf.fz_is_infinite_rect( self.dev.pathrect):
15813: (16)                         self.dev.pathrect = mupdf.fz_make_rect(
15814: (24)                                 self.dev.lastpoint.x,
15815: (24)                                 self.dev.lastpoint.y,
15816: (24)                                 self.dev.lastpoint.x,
15817: (24)                                 self.dev.lastpoint.y,
15818: (24)                                 )
15819: (12)                     self.dev.firstpoint = self.dev.lastpoint
15820: (12)                     self.dev.havemove = 1
15821: (12)                     self.dev.linecount = 0  # reset # of consec. lines
15822: (8)                  except Exception:
15823: (12)                     if g_exceptions_verbose:    exception_info()
15824: (12)                     raise
15825: (0)          def jm_lineart_path(dev, ctx, path):
15826: (4)              '''
15827: (4)              Create the "items" list of the path dictionary
15828: (4)              * either create or empty the path dictionary
15829: (4)              * reset the end point of the path
15830: (4)              * reset count of consecutive lines
15831: (4)              * invoke fz_walk_path(), which create the single items
15832: (4)              * if no items detected, empty path dict again
15833: (4)              '''
15834: (4)              try:
15835: (8)                  dev.pathrect = mupdf.FzRect( mupdf.FzRect.Fixed_INFINITE)
15836: (8)                  dev.linecount = 0
15837: (8)                  dev.lastpoint = mupdf.FzPoint( 0, 0)
15838: (8)                  dev.pathdict = dict()
15839: (8)                  dev.pathdict[ dictkey_items] = []
15840: (8)                  walker = Walker(dev)
15841: (8)                  mupdf.fz_walk_path( mupdf.FzPath(mupdf.ll_fz_keep_path(path)), walker,
walker.m_internal)
15842: (8)                  if not dev.pathdict[ dictkey_items]:
15843: (12)                     dev.pathdict = None
15844: (4)              except Exception:
15845: (8)                  if g_exceptions_verbose:    exception_info()
15846: (8)                  raise
```

```
15847: (0)          def jm_lineart_stroke_path( dev, ctx, path, stroke, ctm, colorspace, color,
alpha, color_params):
15848: (4)              try:
15849: (8)                  assert isinstance( ctm, mupdf.fz_matrix)
15850: (8)                  dev.pathfactor = 1
15851: (8)                  if ctm.a != 0 and abs(ctm.a) == abs(ctm.d):
15852: (12)                     dev.pathfactor = abs(ctm.a)
15853: (8)                  elif ctm.b != 0 and abs(ctm.b) == abs(ctm.c):
15854: (12)                     dev.pathfactor = abs(ctm.b)
15855: (8)                  dev.ctm = mupdf.FzMatrix( ctm)  # fz_concat(ctm, dev_ptm);
15856: (8)                  dev.path_type = trace_device_STROKE_PATH
15857: (8)                  jm_lineart_path( dev, ctx, path)
15858: (8)                  if dev.pathdict is None:
15859: (12)                     return
15860: (8)                  dev.pathdict[ dictkey_type] = 's'
15861: (8)                  dev.pathdict[ 'stroke_opacity'] = alpha
15862: (8)                  dev.pathdict[ 'color'] = jm_lineart_color( colorspace, color)
15863: (8)                  dev.pathdict[ dictkey_width] = dev.pathfactor * stroke.linewidth
15864: (8)                  dev.pathdict[ 'lineCap'] = (
15865: (16)                         stroke.start_cap,
15866: (16)                         stroke.dash_cap,
15867: (16)                         stroke.end_cap,
15868: (16)                         )
15869: (8)                  dev.pathdict[ 'lineJoin'] = dev.pathfactor * stroke.linejoin
15870: (8)                  if 'closePath' not in dev.pathdict:
15871: (12)                     dev.pathdict['closePath'] = False
15872: (8)                  if stroke.dash_len:
15873: (12)                     buff = mupdf.fz_new_buffer( 256)
15874: (12)                     mupdf.fz_append_string( buff, "[ ") # left bracket
15875: (12)                     for i in range( stroke.dash_len):
15876: (16)                         value = mupdf.floats_getitem( stroke.dash_list, i)  #
stroke.dash_list[i].
15877: (16)                         mupdf.fz_append_string( buff, f'{_format_g(dev.pathfactor *
value)} ')
15878: (12)                     mupdf.fz_append_string( buff, f'] {_format_g(dev.pathfactor *
stroke.dash_phase)}')
15879: (12)                     dev.pathdict[ 'dashes'] = buff
15880: (8)                  else:
15881: (12)                     dev.pathdict[ 'dashes'] = '[] 0'
15882: (8)                  dev.pathdict[ dictkey_rect] = JM_py_from_rect(dev.pathrect)
15883: (8)                  dev.pathdict['layer'] = dev.layer_name
15884: (8)                  dev.pathdict[ 'seqno'] = dev.seqno
15885: (8)                  if dev.clips:
15886: (12)                     dev.pathdict[ 'level'] = dev.depth
15887: (8)                  jm_append_merge(dev)
15888: (8)                  dev.seqno += 1
15889: (4)              except Exception:
15890: (8)                  if g_exceptions_verbose:    exception_info()
15891: (8)                  raise
15892: (0)          def jm_lineart_clip_path(dev, ctx, path, even_odd, ctm, scissor):
15893: (4)              if not dev.clips:
15894: (8)                  return
15895: (4)              dev.ctm = mupdf.FzMatrix(ctm)    # fz_concat(ctm, trace_device_ptm);
15896: (4)              dev.path_type = trace_device_CLIP_PATH
15897: (4)              jm_lineart_path(dev, ctx, path)
15898: (4)              if dev.pathdict is None:
15899: (8)                  return
15900: (4)              dev.pathdict[ dictkey_type] = 'clip'
15901: (4)              dev.pathdict[ 'even_odd'] = bool(even_odd)
15902: (4)              if 'closePath' not in dev.pathdict:
15903: (8)                  dev.pathdict['closePath'] = False
15904: (4)              dev.pathdict['scissor'] = JM_py_from_rect(compute_scissor(dev))
15905: (4)              dev.pathdict['level'] = dev.depth
15906: (4)              dev.pathdict['layer'] = dev.layer_name
15907: (4)              jm_append_merge(dev)
15908: (4)              dev.depth += 1
15909: (0)          def jm_lineart_clip_stroke_path(dev, ctx, path, stroke, ctm, scissor):
15910: (4)              if not dev.clips:
15911: (8)                  return
```

```
15912: (4)              dev.ctm = mupdf.FzMatrix(ctm)    # fz_concat(ctm, trace_device_ptm);
15913: (4)              dev.path_type = trace_device_CLIP_STROKE_PATH
15914: (4)              jm_lineart_path(dev, ctx, path)
15915: (4)              if dev.pathdict is None:
15916: (8)                  return
15917: (4)              dev.pathdict['dictkey_type'] = 'clip'
15918: (4)              dev.pathdict['even_odd'] = None
15919: (4)              if 'closePath' not in dev.pathdict:
15920: (8)                  dev.pathdict['closePath'] = False
15921: (4)              dev.pathdict['scissor'] = JM_py_from_rect(compute_scissor(dev))
15922: (4)              dev.pathdict['level'] = dev.depth
15923: (4)              dev.pathdict['layer'] = dev.layer_name
15924: (4)              jm_append_merge(dev)
15925: (4)              dev.depth += 1
15926: (0)          def jm_lineart_clip_stroke_text(dev, ctx, text, stroke, ctm, scissor):
15927: (4)              if not dev.clips:
15928: (8)                  return
15929: (4)              compute_scissor(dev)
15930: (4)              dev.depth += 1
15931: (0)          def jm_lineart_clip_text(dev, ctx, text, ctm, scissor):
15932: (4)              if not dev.clips:
15933: (8)                  return
15934: (4)              compute_scissor(dev)
15935: (4)              dev.depth += 1
15936: (0)          def jm_lineart_clip_image_mask( dev, ctx, image, ctm, scissor):
15937: (4)              if not dev.clips:
15938: (8)                  return
15939: (4)              compute_scissor(dev)
15940: (4)              dev.depth += 1
15941: (0)          def jm_lineart_pop_clip(dev, ctx):
15942: (4)              if not dev.clips or not dev.scissors:
15943: (8)                  return
15944: (4)              len_ = len(dev.scissors)
15945: (4)              if len_ < 1:
15946: (8)                  return
15947: (4)              del dev.scissors[-1]
15948: (4)              dev.depth -= 1
15949: (0)          def jm_lineart_begin_layer(dev, ctx, name):
15950: (4)              if name:
15951: (8)                  dev.layer_name = name
15952: (4)              else:
15953: (8)                  dev.layer_name = ""
15954: (0)          def jm_lineart_end_layer(dev, ctx):
15955: (4)              dev.layer_name = ""
15956: (0)          def jm_lineart_begin_group(dev, ctx, bbox, cs, isolated, knockout, blendmode,
alpha):
15957: (4)              if not dev.clips:
15958: (8)                  return
15959: (4)              dev.pathdict = { # Py_BuildValue("{s:s,s:N,s:N,s:N,s:s,s:f,s:i,s:N}",
15960: (12)                     "type": "group",
15961: (12)                     "rect": JM_py_from_rect(bbox),
15962: (12)                     "isolated": bool(isolated),
15963: (12)                     "knockout": bool(knockout),
15964: (12)                     "blendmode": mupdf.fz_blendmode_name(blendmode),
15965: (12)                     "opacity": alpha,
15966: (12)                     "level": dev.depth,
15967: (12)                     "layer": dev.layer_name
15968: (12)                     }
15969: (4)              jm_append_merge(dev)
15970: (4)              dev.depth += 1
15971: (0)          def jm_lineart_end_group(dev, ctx):
15972: (4)              if not dev.clips:
15973: (8)                  return
15974: (4)              dev.depth -= 1
15975: (0)          def jm_lineart_stroke_text(dev, ctx, text, stroke, ctm, colorspace, color,
alpha, color_params):
15976: (4)              jm_trace_text(dev, text, 1, ctm, colorspace, color, alpha, dev.seqno)
15977: (4)              dev.seqno += 1
15978: (0)          def jm_dev_linewidth( dev, ctx, path, stroke, matrix, colorspace, color,
```

```
alpha, color_params):
15979: (4)              dev.linewidth = stroke.linewidth
15980: (4)              jm_increase_seqno( dev, ctx)
15981: (0)          def jm_increase_seqno( dev, ctx, *vargs):
15982: (4)              try:
15983: (8)                  dev.seqno += 1
15984: (4)              except Exception:
15985: (8)                  if g_exceptions_verbose:    exception_info()
15986: (8)                  raise
15987: (0)          def planish_line(p1: point_like, p2: point_like) -> Matrix:
15988: (4)              """Compute matrix which maps line from p1 to p2 to the x-axis, such that
it
15989: (4)              maintains its length and p1 * matrix = Point(0, 0).
15990: (4)              Args:
15991: (8)                  p1, p2: point_like
15992: (4)              Returns:
15993: (8)                  Matrix which maps p1 to Point(0, 0) and p2 to a point on the x axis at
15994: (8)                  the same distance to Point(0,0). Will always combine a rotation and a
15995: (8)                  transformation.
15996: (4)              """
15997: (4)              p1 = Point(p1)
15998: (4)              p2 = Point(p2)
15999: (4)              return Matrix(util_hor_matrix(p1, p2))
16000: (0)          class JM_image_reporter_Filter(mupdf.PdfFilterOptions2):
16001: (4)              def __init__(self):
16002: (8)                  super().__init__()
16003: (8)                  self.use_virtual_image_filter()
16004: (4)              def image_filter( self, ctx, ctm, name, image):
16005: (8)                  assert isinstance(ctm, mupdf.fz_matrix)
16006: (8)                  JM_image_filter(self, mupdf.FzMatrix(ctm), name, image)
16007: (8)                  if mupdf_cppyy:
16008: (12)                     return 0
16009: (0)          class JM_new_bbox_device_Device(mupdf.FzDevice2):
16010: (4)              def __init__(self, result, layers):
16011: (8)                  super().__init__()
16012: (8)                  self.result = result
16013: (8)                  self.layers = layers
16014: (8)                  self.use_virtual_fill_path()
16015: (8)                  self.use_virtual_stroke_path()
16016: (8)                  self.use_virtual_fill_text()
16017: (8)                  self.use_virtual_stroke_text()
16018: (8)                  self.use_virtual_ignore_text()
16019: (8)                  self.use_virtual_fill_shade()
16020: (8)                  self.use_virtual_fill_image()
16021: (8)                  self.use_virtual_fill_image_mask()
16022: (8)                  self.use_virtual_begin_layer()
16023: (8)                  self.use_virtual_end_layer()
16024: (4)              begin_layer = jm_lineart_begin_layer
16025: (4)              end_layer = jm_lineart_end_layer
16026: (4)              fill_path = jm_bbox_fill_path
16027: (4)              stroke_path = jm_bbox_stroke_path
16028: (4)              fill_text = jm_bbox_fill_text
16029: (4)              stroke_text = jm_bbox_stroke_text
16030: (4)              ignore_text = jm_bbox_ignore_text
16031: (4)              fill_shade = jm_bbox_fill_shade
16032: (4)              fill_image = jm_bbox_fill_image
16033: (4)              fill_image_mask = jm_bbox_fill_image_mask
16034: (0)          class JM_new_output_fileptr_Output(mupdf.FzOutput2):
16035: (4)              def __init__(self, bio):
16036: (8)                  super().__init__()
16037: (8)                  self.bio = bio
16038: (8)                  self.use_virtual_write()
16039: (8)                  self.use_virtual_seek()
16040: (8)                  self.use_virtual_tell()
16041: (8)                  self.use_virtual_truncate()
16042: (4)              def seek( self, ctx, offset, whence):
16043: (8)                  return self.bio.seek( offset, whence)
16044: (4)              def tell( self, ctx):
16045: (8)                  ret = self.bio.tell()
```

```
16046: (8)                 return ret
16047: (4)             def truncate( self, ctx):
16048: (8)                 return self.bio.truncate()
16049: (4)             def write(self, ctx, data_raw, data_length):
16050: (8)                 data = mupdf.raw_to_python_bytes(data_raw, data_length)
16051: (8)                 return self.bio.write(data)
16052: (0)         def compute_scissor(dev):
16053: (4)             '''
16054: (4)             Every scissor of a clip is a sub rectangle of the preceding clip scissor
16055: (4)             if the clip level is larger.
16056: (4)             '''
16057: (4)             if dev.scissors is None:
16058: (8)                 dev.scissors = list()
16059: (4)             num_scissors = len(dev.scissors)
16060: (4)             if num_scissors > 0:
16061: (8)                 last_scissor = dev.scissors[num_scissors-1]
16062: (8)                 scissor = JM_rect_from_py(last_scissor)
16063: (8)                 scissor = mupdf.fz_intersect_rect(scissor, dev.pathrect)
16064: (4)             else:
16065: (8)                 scissor = dev.pathrect
16066: (4)             dev.scissors.append(JM_py_from_rect(scissor))
16067: (4)             return scissor
16068: (0)         class JM_new_lineart_device_Device(mupdf.FzDevice2):
16069: (4)             '''
16070: (4)             LINEART device for Python method Page.get_cdrawings()
16071: (4)             '''
16072: (4)             def __init__(self, out, clips, method):
16073: (8)                 super().__init__()
16074: (8)                 self.use_virtual_fill_path()
16075: (8)                 self.use_virtual_stroke_path()
16076: (8)                 self.use_virtual_clip_path()
16077: (8)                 self.use_virtual_clip_image_mask()
16078: (8)                 self.use_virtual_clip_stroke_path()
16079: (8)                 self.use_virtual_clip_stroke_text()
16080: (8)                 self.use_virtual_clip_text()
16081: (8)                 self.use_virtual_fill_text
16082: (8)                 self.use_virtual_stroke_text
16083: (8)                 self.use_virtual_ignore_text
16084: (8)                 self.use_virtual_fill_shade()
16085: (8)                 self.use_virtual_fill_image()
16086: (8)                 self.use_virtual_fill_image_mask()
16087: (8)                 self.use_virtual_pop_clip()
16088: (8)                 self.use_virtual_begin_group()
16089: (8)                 self.use_virtual_end_group()
16090: (8)                 self.use_virtual_begin_layer()
16091: (8)                 self.use_virtual_end_layer()
16092: (8)                 self.out = out
16093: (8)                 self.seqno = 0
16094: (8)                 self.depth = 0
16095: (8)                 self.clips = clips
16096: (8)                 self.method = method
16097: (8)                 self.scissors = None
16098: (8)                 self.layer_name = ""  # optional content name
16099: (8)                 self.pathrect = None
16100: (8)                 self.linewidth = 0
16101: (8)                 self.ptm = mupdf.FzMatrix()
16102: (8)                 self.ctm = mupdf.FzMatrix()
16103: (8)                 self.rot = mupdf.FzMatrix()
16104: (8)                 self.lastpoint = mupdf.FzPoint()
16105: (8)                 self.firstpoint = mupdf.FzPoint()
16106: (8)                 self.havemove = 0
16107: (8)                 self.pathrect = mupdf.FzRect()
16108: (8)                 self.pathfactor = 0
16109: (8)                 self.linecount = 0
16110: (8)                 self.path_type = 0
16111: (4)             fill_path           = jm_lineart_fill_path
16112: (4)             stroke_path         = jm_lineart_stroke_path
16113: (4)             clip_image_mask     = jm_lineart_clip_image_mask
16114: (4)             clip_path           = jm_lineart_clip_path
```

```
16115: (4)              clip_stroke_path    = jm_lineart_clip_stroke_path
16116: (4)              clip_text           = jm_lineart_clip_text
16117: (4)              clip_stroke_text    = jm_lineart_clip_stroke_text
16118: (4)              fill_text           = jm_increase_seqno
16119: (4)              stroke_text         = jm_increase_seqno
16120: (4)              ignore_text         = jm_increase_seqno
16121: (4)              fill_shade          = jm_increase_seqno
16122: (4)              fill_image          = jm_increase_seqno
16123: (4)              fill_image_mask     = jm_increase_seqno
16124: (4)              pop_clip            = jm_lineart_pop_clip
16125: (4)              begin_group         = jm_lineart_begin_group
16126: (4)              end_group           = jm_lineart_end_group
16127: (4)              begin_layer         = jm_lineart_begin_layer
16128: (4)              end_layer           = jm_lineart_end_layer
16129: (0)          class JM_new_texttrace_device(mupdf.FzDevice2):
16130: (4)              '''
16131: (4)              Trace TEXT device for Python method Page.get_texttrace()
16132: (4)              '''
16133: (4)              def __init__(self, out):
16134: (8)                  super().__init__()
16135: (8)                  self.use_virtual_fill_path()
16136: (8)                  self.use_virtual_stroke_path()
16137: (8)                  self.use_virtual_fill_text()
16138: (8)                  self.use_virtual_stroke_text()
16139: (8)                  self.use_virtual_ignore_text()
16140: (8)                  self.use_virtual_fill_shade()
16141: (8)                  self.use_virtual_fill_image()
16142: (8)                  self.use_virtual_fill_image_mask()
16143: (8)                  self.use_virtual_begin_layer()
16144: (8)                  self.use_virtual_end_layer()
16145: (8)                  self.out = out
16146: (8)                  self.seqno = 0
16147: (8)                  self.depth = 0
16148: (8)                  self.clips = 0
16149: (8)                  self.method = None
16150: (8)                  self.seqno = 0
16151: (8)                  self.pathdict = dict()
16152: (8)                  self.scissors = list()
16153: (8)                  self.linewidth = 0
16154: (8)                  self.ptm = mupdf.FzMatrix()
16155: (8)                  self.ctm = mupdf.FzMatrix()
16156: (8)                  self.rot = mupdf.FzMatrix()
16157: (8)                  self.lastpoint = mupdf.FzPoint()
16158: (8)                  self.pathrect = mupdf.FzRect()
16159: (8)                  self.pathfactor = 0
16160: (8)                  self.linecount = 0
16161: (8)                  self.path_type = 0
16162: (8)                  self.layer_name = ""
16163: (4)              fill_path = jm_increase_seqno
16164: (4)              stroke_path = jm_dev_linewidth
16165: (4)              fill_text = jm_lineart_fill_text
16166: (4)              stroke_text = jm_lineart_stroke_text
16167: (4)              ignore_text = jm_lineart_ignore_text
16168: (4)              fill_shade = jm_increase_seqno
16169: (4)              fill_image = jm_increase_seqno
16170: (4)              fill_image_mask = jm_increase_seqno
16171: (4)              begin_layer = jm_lineart_begin_layer
16172: (4)              end_layer = jm_lineart_end_layer
16173: (0)          def _get_glyph_text() -> bytes:
16174: (4)              '''
16175: (4)              Adobe Glyph List function
16176: (4)              '''
16177: (4)              import base64
16178: (4)              import gzip
16179: (4)              return gzip.decompress(base64.b64decode(
16180: (12)
b'H4sIABmRaF8C/7W9SZfjRpI1useviPP15utzqroJgBjYWhEkKGWVlKnOoapVO0YQEYSCJE'
16181: (12)
b'IcMhT569+9Ppibg8xevHdeSpmEXfPBfDZ3N3f/t7u//r//k/zb3WJ4eTv2T9vzXTaZZH/N'
```

```
16182: (12)
b'Junsbr4Z7ru7/7s9n1/+6z//8/X19T/WRP7jYdj/57//R/Jv8Pax2/Sn87G/v5z74XC3Pm'
16183: (12)
b'zuLqfurj/cnYbL8aEzyH1/WB/f7h6H4/70l7vX/ry9G47wzK/hcr7bD5v+sX9YM4i/3K2P'
16184: (12)
b'3d1Ld9z353O3uXs5Dl/7DT7O2/UZ/3Tw9zjsdsNrf3i6exgOm57eTsbbvjv/1w2xTnfDo5'
16185: (12)
b'fnYdjA3eV0vjt25zXkRJB36/vhKwN+kEw4DOf+ofsLuP3pboewGISO7bAxPkUU+EaUD7t1'
16186: (12)
b'v++O/3FTCESmcsILgQRuLhDs/w857lz6NsPDZd8dzmtfSP85HO8GcI53+/W5O/br3QkeJa'
16187: (12)
b'9NERmPKgE2Ue+73vgj97Ded5TH1pPDEFCT4/35RFFtAMORMezXb3dwiioCsYe77rABjjCO'
16188: (12)
b'jHs/nLs7mx3wuYFYX+HsEQyTfHg/DY/nVxa0rzmnl+6BVQfeegTyemSlOdjqczqJ0J9/ev'
16189: (12)
b'fp7tOH1ed/zj+2d/j+9eOHf7xbtsu75jcw27vFh19/+/jux58+3/304edl+/HT3fz9kq3i'
16190: (12)
b'w/vPH981Xz5/APR/5p/g9/+Qhb+/3bX/8+vH9tOnuw8f79798uvP7xAcwv84f//5XfvpL/'
16191: (12)
b'D97v3i5y/Ld+9//Msdgrh7/+Hz3c/vfnn3GQ4/f/iLifja492HFbz+0n5c/ARg3rz7+d3n'
16192: (12)
b'30ycq3ef3zO+FSKc3/06//j53eLLz/OPd79++fjrh0/tHRIHr8t3nxY/z9/90i7/AxIg1r'
16193: (12)
b'v2H+37z3effpr//PPN1CIF47Q2LUSdNz+3NjakdvnuY7v4/BcEGb4WyEPI+DMT++nXdvEO'
16194: (12)
b'n8iWFomaf/ztL8wZhPqp/e8vcAbm3XL+y/xHpPH/xlnDejXKHJTQ4svH9hdK/mF19+lL8+'
16195: (12)
b'nzu89fPrd3P374sDSZ/qn9+I93i/bTD/D+8wcWxOruy6f2L4jl89xEjkCQaZ9+4Hfz5dM7'
16196: (12)
b'k33v3n9uP3788uvndx/e/zu8/vThn8ggSDqH56XJ6Q/vTZKRVx8+/sZgmRemIP5y98+fWu'
16197: (12)
b'Ao8vc+z+bMjE/Iu8Vn7RBxIis/q7TevW9//Pndj+37RWuz/AND+ue7T+2/o+zefaKTdzbq'
16198: (12)
b'f84R7xeTdJYYJLOf7z4xq11N/osp2bt3q7v58h/vKLxzjtrw6Z2rOSbzFj+5rEd7+P84UL'
16199: (12)
b'xH8/6vO/lj2/6Pu7eX7d3P6C3Y2tb3u+7ua3dkA/yvu+w/JqyV6GeUt0/dy7nb36MjySZ/'
16200: (12)
b'MUMO3Hz5+LNycsdx54SB5wmN/XJvRh0z/vz1/PaCf4Zhd/rP9dPur/j7eDDtfIV+dX3+r7'
16201: (12)
b'vz63B36vb9w7AbDn/ddLseown7kr7bbU4YIhD6/03//e7JiM0O669/vbyg1/hPdKLd8WGN'
16202: (12)
b'PmnXoSs52h5200OGk/WW/fvdl0NvhpHTw3q3Pt59Xe8uCOARA8ydCcX433Z/rjfonfbrnf'
16203: (12)
b'hP5j9MJtM0mbf4XZT4XT9czt0Pk3S1ALFfPxyHA6g2A3WCz90Pq6qFO+dsskjdtzAB3B+7'
16204: (12)
b'rwwDeWi/reu0nbcOeMBostv1Dz9MpsuJwzbD+b5DcuGuKR32dFx/pcfGO9oOw7MZlAj64M'
16205: (12)
b'/9bmOAaTJ/WFuJF0t898eHXfdDNmV4JC77x133J8XONCDiTTWq5JkvNMMLNY9C1ZLNa82R'
16206: (12)
b'rIki9ULP50AZ/6pczOyn92DSE3IqRSZs7nc2+gmqKMi+O3an/sQkTQOpszcLsBTnsg2gSE'
16207: (12)
b'f/KskTQ4YaANrFPFn4b/ELIEo/Iu2jQkbg/QEtEJXe1Y6MtWP3sl3/MMlnqf08D4cBaclr'
16208: (12)
b'5KzEzHTuyXhZPyCXVhkcD0/DoXsmEwEfoWVQqsJ+Sg2eW9qniOGQFqHh3n+XCNMWCMLJ3b'
16209: (12)
b'c4BPB2vz5CYenXkKjI06Rhu8mSJlSxKmmQX+uHB6g1jC0ztEQ+TRqdISmC6A46TLiH/sfM'
16210: (12)
b'wBczE0mo4WrXHzoJpUyaKCvglLnpJC1XiEWSBN55eIHcDChLFpQ4TxZrHWkL2mUXwl6Yto'
16211: (12)
b'N6OLefEmyRLHy7mizwDT1yt1szryqhfCOa1AJJBtKVZFRtCd8WU3pATvFrbr5cHlo6Dome'
16212: (12)
b'tzoF0xmAbn3/vF2fgKgcbhbkKCCrCKBYETp0uZt+2siJ5pSGc92+kOVgbLVIOREE/rw+jc'
16213: (12)
b'JfNGSxGWBysYMmOzxrCU3qelSBOUV1VQCf456kXEGaqB4gykGJUKTJQupBnixZ9NNk+S+2'
16214: (12)
b'ihS/0kkCjOoD6ccjhCO3niVLKfYW367Y0xY90TIU6MwSVkRfVdMM6HFYsxzpPGobc0NLrV'
16215: (12)
b'4ky6htQIoOA9rLmWTeIupuh6aRZaij5vPp2LH15zO49PmEMH1niBrcCCWd60KgH00/Bmgp'
16216: (12)
```

```
b'kM8t9NzL/mm930scS/j7XYuHlr2MGiXkiwoDQvnESoFVyfKEarx1uSGFA7ehkULobywiRP'
16217: (12)
b'BNiqgAcbOCo9MFRwtGp1GVn6wSDuzTImllwJ65b2mcAPyAjZxvfcTpHN+2xC0bZboApKt6'
16218: (12)
b'joBDPZhbIgyyEeD7B7Sx9kZ1qTWqKgeUkvZ66MUI1N4eejGytzeG3kgUP/QumFyVWyD1+E'
16219: (12)
b'pSja9NICVYYqbrSkvzJV2Xo0WhQfIedV+EsGU0rd23hAogyuUKtNZ7kBjOxTEPBT9LS/Cv'
16220: (12)
b'BlfE32OqDgVzo+JFfWt3uqkhATv4OEhYCFtGXrRhR/jCY7Is4kuCVWavQ0QdiVoDqoiute'
16221: (12)
b'kS9K0eFjpDy3E8nc75EdVjKGbtgVmg+1KkWtQAVp/hpaPQM1SNl1O/YwryWeEJUS3gUkeb'
16222: (12)
b'wTnzDLP+DdtgG0jtClLrXh86SHu6mQoIb1r5HM1KWjmksEN7xQ9VsjVpEQ1ezvA7gUqMD+'
16223: (12)
b'97RcpruAv3Le0G8V2Oww/ZBDpq+40xQxPBh2/G6D1BqRSiKq7YJ5TJKjTdJlnpDjptk1U0'
16224: (12)
b'phVwrbvkabJy/S5Ut1UPnyELqgwIovM1Cm6jCoGgMDERdp6sJJ/K5EeKViU/Nqc/Lutj90'
16225: (12)
b'OeYwD8UVS6Kb7RNzMrc/sZhqsZmYenfh3EnCc/StfWJj9KniAe0WFSKFE/hpxYWEK0k5TA'
16226: (12)
b'wIh806Z72+hRd37UjZ50NJBBxu16o3UD+N1iHrjZ7LpRfab42+5KJ5gZH5eX8+WomxFq+Y'
16227: (12)
b'++BBALJnWqVgGIRywArlFjJgefUXkgf/142NpPKQ84le/KfdtYs1kD2gjLDJ0mP7Hg6uSn'
16228: (12)
b'tEb8P2TFYmW+p/xGo+B3kfK7SX7CQF4ZPE1++lUKGh3sT+tbAx3G5J/WN5WyDIzj5tQ/ae'
16229: (12)
b'cZYrMDKqraT6b8fWshK2gxGcINBb+0hBQ8uuifpPuHY4SlmwhqwU+qg6frKFcRttbIphPQ'
16230: (12)
b'R9WCwJesxfcF85bjZb9bX84siFWEiBYBh98kv1AF3jHTZ8k7PUvMVsm7v0F+TCjefdF4m7'
16231: (12)
b'wTJWDpvmXIAeBbSrZI3on2gcBCFrWWCAN8BEhYRFXlK5N3elStQapRdRVIP8hQ0huaNirZ'
16232: (12)
b'u6sBmN5NW8wn5kvaoqNFjZgn77qrpQeIFrXXInn3eFw/o62hZ8IU7Z2M0Qv3LREDiNQOJK'
16233: (12)
b'vXQZEej8mQoT9th+NZO0TxyYCL+ukInW4UZFS14AO1SrX3Jnk36ByH4DIyMjMHO/jMzJfq'
16234: (12)
b'MEsDhNLI0VCJyIAEUiopfEt7xzj2zk2XU9T0d9GQxPrzbdufT9GgMPWgrwuaWSZ/Y02eJ3'
16235: (12)
b'+L5nZp8rdQ+VaWkPaJucrfok6uTv42mog1yd+ijEP4kpx58ndG2SR/V0NNkfz976E/WiZ/'
16236: (12)
b'X99DZ3/uoxF+AtjV1Nx8q8JEqDd7qhkZYwUmB/byYoqG7OuuvwX63cnibJH8XQa0Gt8yoO'
16237: (12)
b'UlKJ9v0JT/Ho9fZKuWgX7i7/FYPwUQLU2skr9vdTKh0/19q9UBhOgHI0gSjz0QU8+WUGx/'
16238: (12)
b'jwoFJTAgF5SXemIhmYEhH066cZUEfEE2yc8syEXyM3s9aIU//4yuEtXlZ6815DN87+83Jq'
16239: (12)
b'fh3OdavsR3yDVyJNdSS8STlByRjPISnlz/szJfgWNp8VoGUoZiqH8/969RViOG35kMcOJs'
16240: (12)
b'RBqibJwnP0fZCI9+gol2Y79l3IBnya9F8gvza5n8oip+mfxihVqVUD7tt0yJVwRchW+TX0'
16241: (12)
b'ImZckvekjEGPeLSjJ0nV+iejSdJr9EMkMGEQvfVHGMioqq/cuFhbVI3lPWNnlvynaevPdl'
16242: (12)
b'Os2T974coS++D+WIye77IGJuibgc0dG8j8uRnqKkTA0tHsrkPSv4rnuk69kyeY+yEBW2Tt'
16243: (12)
b'6bQmvwGxUa4tGFBv3ofZQBSNjwqnMI8UiOgOmXJJep+5Y5AQCTQ8vkA3NolXzARD8tMvxK'
16244: (12)
b'qc+TD37AX+buWwIAACXpGM1y0I048Nbwi+C8ioAS+eBzH7J9YK7Bw8aPCTPIE8pgaglRG5'
16245: (12)
b'YR4KsW6t2HmysAy1oz/LxzmWlUD8Vx8JLgCPXzKWgAH3T/jXRhfPKVrJgYUlSXBcigutDv'
16246: (12)
b'rXxSsEROTCkjCMiMz1JUDQCnajBhkaqxAhD1zwXoPeodVNIPkQ7Skj6yUDBImU/J3LmllR'
16247: (12)
b'BtZiHJ0IWlo6x0IfrsahmsVlVtHvWMEcFdKTzwLroNeugP8WICa2u8mMDA9t3T2iWOn7rb'
16248: (12)
b'd1w/LmCKbejjcDnoalzNLX7uzzutF1ULh3v1BrV031vx8pkQwqZz3VrhQjV6CCNKFtuGJc'
16249: (12)
b'J+CXy7FQn0rh9c3zxhZTbfMqVtHSDFTRe+D0CUduDXzrX6WJH2vUThvn0GM8sNoOYxU+9B'
16250: (12)
b'4iuSX+EZWf+rFMw0+TU0X/B111iUya+R0rwCHaldcwA3p7hzeLXr2/ywCsMccRkI8fevR1'
```

```
16251: (12)
b'3P8+RXnf9Qtn49Gac1P3QmkOOSg+//ZnLS5L9DEsrkv6OQwBT3afKR7rPkY6R7LkD7bmCa'
16252: (12)
b'fPS9XVHjW8Ya5MXHEEsFIhpVyFb9RzoBqXOyNrRvkMU8kKIiFJAj1s4QiJqjgL0dmCdIRt'
16253: (12)
b'jbKlcLknFrTJFEPRoVbfIxyhXwJVf8tw8E/ut0hJ0uLx2tXMBryuQTczFPPq24YzeZYHqP'
16254: (12)
b'/hJU5qh0Sir31ITU1FM1qcJRufFXOiozVOV5JpTa+zO8mXdJnoncxM4YUpElI+VdlimozL'
16255: (12)
b'ssycu8SxQaKC81OltQXuqS6cu81IUJxUtdVKS81MWSlJe6oJyZl7poQOXisiUlLlekxOWc'
16256: (12)
b'lJe6YPqmIvWMlJe6pNRTL3XJtE+91IWhvNQlZZl6qUtKPfWylCyHqZelNPF5WUrmxFRkYe'
16257: (12)
b'yFl6Wgv0JykPlZSA4yzwrJQaa9EFmQPmll/ls3EYqw3r/0vsvHAPTJN8XSf0ceSgdKS0BB'
16258: (12)
b'qAaLzH7YvvITvb/51OsBtYVubaNDutDSa0vIXJTlGzX9jDU6kmtiaN/2WOU8GTmDt7gzhf'
16259: (12)
b'jR+jzSF2+AVgT05AxBbB9iCIUVzdcQ+zZy0SB5236vlk6Rov7JrLTOUYD9nyIAqkHUa4A7'
16260: (12)
b'PJ7Ha3DwLn0JXJwZlszn5slndhbT5POaSiyGgM92wQ6p+yzFCzQUHDLsc8j/mSVirR49/+'
16261: (12)
b'e4/6WnKHfnhpZCWCSfow1iOL+5+Tunw1AEiL07n6KNW8i6dbv3NT7d0LbgJ/WxCRQp8ymD'
16262: (12)
b'Lmlkh4SJqNWgXJIfzwyh4n/WvTemB5+jcoAIesERk97PUEgee6OwNwtDnXrW1npqiPPrQC'
16263: (12)
b'Gr5POxg47h1WhiCDtKH5Sxz6d4Z7EB4gsY4b12O7XkD+brIFSafGFxF8kXmY7M3bfkBwA/'
16264: (12)
b'uUCxfJHJRY5vKfa5JcJEotGA1INSoxID3aoUIWCl6aPufNEj9RSk0vQXgfQ+llXAJOYsYJ'
16265: (12)
b'KCmcKU2cAkwC7WlMm5NtUpAihpoTxKk4e0MnuYuW9xC0Cr9JiefPGThJX99Gofpn9fRpME'
16266: (12)
b'iqknCVB0v4wnCegqvkSThBZ0PElg9mpIZwTy7EpTgYxab6wgmGQIGvGX6zXS1oNK1a3oUj'
16267: (12)
b'cRZKWo7Cwr2SacF55I2T8Jy+QM03p6298PO+nAcnEgi6lN6jG9ntqMwRuBTb2bwIuEkPkI'
16268: (12)
b'0mhNnVI0/i/jheQJMd8ikR7MG9bcJdb9WBvga+MTlJGfv2MY+hLNJCoPSFWfJv9goy6Tf4'
16269: (12)
b'T22ST/UHUHU5N/RBOFDHS02gEHrsdpwIuKCuFG2yd18g9JHHi+rmFK90+KUSX/9KLWWfLP'
16270: (12)
b'INLCEjJSQ+5/qipSk1QjBKZq/1RJqOvkn77q15Pkn5GIiFNEqpL/oRh18j8h6mXyPzqmBU'
16271: (12)
b'gd0zz5n2ikz+Ges5tZm/xPFA8ClXjq5DfGM0t+k6506b6lwRPQpY6x5bcgVWuJkCFl8luo'
16272: (12)
b'sSljuOpuVsC06K2hpY+YJr9hHqA714bI5Va3h+B9hqLl/+aLP7efvktZQSi9wzEtQOu6Xo'
16273: (12)
b'GOhkfonL9FuYYsklzDt68wFOByuu+fdAbNHXbLYGJB3q4/n3e6LkNREfiWrzr5F8tpnvwr'
16274: (12)
b'Mq8qQfsRZ5aIGVa1dN8y/K8ASJE5whVZ2s4myb/sonPVmC9ReBztS2aWJf+KWmAF+ub2RE'
16275: (12)
b'3GDa23BW7VGoi+7XRa5gTGO2qLlKiO0vi7Gafl3Ih0kfxLazqzafKvqGgRsxQtv/2uVFMk'
16276: (12)
b'tEmEvrFe33cYbXZoTzM06bVvLC1Zm+4rnM0mxJ8uv6+P6zPczWtLH/eXZ65RzA1/v0Z3qc'
16277: (12)
b'C8BXi8yML5JAf9dYD2QwU4RNq0Gncx5hGooqbre2Zlb87D7NfHZ121VxFXBYhhVScUyb8f'
16278: (12)
b'Xob98Dj8kNN+ay2G2Ln7FkvnlQN0vqcO03ZLlcPEENs7igySfPBipgJRZAsZiZO6vJxYQl'
16279: (12)
b'Q4TEXWNwyxC41qq+SlZoghdqXRyBB5pjlict0kvkZAczefJoKH/T2qelpZyFKT1FFDRLoS'
16280: (12)
b'KJx3LtkMXCRBYzUABm0XwJQ+Qi7nyAG9pgzuZrN+VnWsIuTqKPJB6aFQ9G7OTfMAB70Rgu'
16281: (12)
b'iMSw0ZlidBmxaBWh4WF5G73fNw7FDvcq7srrvgAZE89v2EO/g/QOzCkvVsmtL4aGrIdII+'
16282: (12)
b'yFqqe7K2xs6enFlFwJHZxFrJeDK11p+ezOyevCdzu7ftyantXjxZ2A7Ok6XdhPdkZbfaPV'
16283: (12)
b'nbzVpPzqwpnCPzibVj82RqzdY8mdmNAk/mdg3Uk1NrU+bJwhqLebK000xPVnYm4snaWgZ6'
16284: (12)
b'cma3Wh05ndiJmCdTa9LsycxO/T2Z22m/J6fWLsaThR2kPVnaGbsnK2vw5snaGo94cmZtTB'
16285: (12)
```

```
b'xZTKwxkidTayDrycxaH3kyt1aWnpxao1VPFtZaxJOlHeg9Wdk9fk/WdlPUkzO73ebIcmKn'
16286: (12)
b'qJ5M7Ua0JzOrLnsyp8WNSFVOSYpUZeEarSMpVS4FWlKqXNJbUqpc0ltSqlxCrihVLiFXlK'
16287: (12)
b'qQoCpKlUvyK+ZVLsmvmFe5JL8yUknyKyOVJL8yUknyKyOVJL8yUkn51kYqyY2aUuVSvjWl'
16288: (12)
b'mkrya0o1FZlrSjWV5NeUairJrynVVJJfU6qpJL+mVFNJb02pppLeGaWaSnpnlGoq6Z0ZqS'
16289: (12)
b'S9MyOVpHdmpJL0zoxUkt6ZkUrSOzNSSXpnlGomCZxRqsInEADJXEhTglMhKVVRCEmpilJI'
16290: (12)
b'SlVUQlKqohaSUhUzISlVMReSUhWNkEYqn8A0NVL5FKWmdU9WQpZ2DuDJyppoerK2xjmORM'
16291: (12)
b'ai8ovMJmMLCcpkbCnJNxlbBZIRVT75NbpNBFUJaUL26a2NVEub3gy5nE1cg8y5MDxx4mO4'
16292: (12)
b'JWHLrqhyVs6ynAsJ4UvXrkGyVpTlRMicZCrklGQmZEEyF7IkORWyIlkIyYjKUsgZycqRU9'
16293: (12)
b'aKsqyFNELOhKQYbnAhyZDdeEGSQWVeyCmLsswyIRlUlgvJBGZTIRlyVgjJBGalkExgJkKm'
16294: (12)
b'TGAmQnKYLjMRksN0mc2FNFKJzJmRaiGkkWoppJGqFdJIJQnkMF3mEyEpVS7p5TBd5pJeDt'
16295: (12)
b'NlLunlMF3mkl4O02Uu6eUwXeaSXg7TZS7p5TBd5pJeDtNlLunNjVSSXo6t5VSE5NhaTkVI'
16296: (12)
b'jq3lVITk2FpORUiOreVUhGTrK6ciJOt5ORUh2dzKqUjFwbScilSFEUOkKowYUgqFEUNKoT'
16297: (12)
b'BiSCkURgwphcKIIaXAwbQsJIEcTMtCEsjBtCwkgZURw+dkwZ6qnE+FZFBVKySDqkshGdSs'
16298: (12)
b'FpIJnHsxClOfq5mQTFEtjk19nqVCMkXNXEgGtfRCFqYElz6fUQ+ohXrHJUuhaLyQJRNYLH'
16299: (12)
b'yRoZ2DXE6EpONlKmRJMhOyIhn8MqjlVMgZSRGDWVcsSyFTkpWQGclayJzkTEgjlSShMlI1'
16300: (12)
b'QhqpFkIaqZZCGqkkvZWRymd7ySG+aCW97EWLVtLLIb5oJb0c4otW0sshvmglvRzii1bSyy'
16301: (12)
b'G+aCW9HOKLVtLL/rloJb0c4otW0jszUkl60T+vmiyQBUmf/Ap97KqZBpJc6UUrdm7FaiIk'
16302: (12)
b'xVilQlKMlU9ghQ5q1Ug3UnGYKJqpkExvE7imIpVCMqJGxOAwUTS1kIyoqYRkehsvVc1hom'
16303: (12)
b'gyIVkKTSokS6HJhaRUi+CYUi2CYyPGTEgjhq8bdW7i9XWjnpqIVkIyooWXasZONXN+yzRD'
16304: (12)
b'B5WlTicHiSLLUjdBK9McXVCWujlXmRY04p9kCyGnJJdCFiRbR7LRYSh3jvO0NCOsczydcS'
16305: (12)
b'qUUWa/kcHqqldniiRanAG57Y/rp/Vh/UPOk7jraNoPifuwMsL5Sa+XRiBU76bYnKrGR5UR'
16306: (12)
b'dK9iNp5V1MbDeF2IXTpvUlnfMwwz0PSHRyA7h61ogQ4M/517jTZE990mAhcER7ZUTNKNlS'
16307: (12)
b'aqVP14pWkagSoxdP28PuOvybd5Fsjtevf42m/O2x9WKy5ByDoAR5Fd9+i6THxJMqldgN6s'
16308: (12)
b'n7rT1iwGvrJpWVdx6uvWgNv1/tvalFIIJB9xRh6ngW0WM4LHYsQZeawt24olwu/WyGyR1a'
16309: (12)
b'VtzzWYkVjZiDMK3bOfT5fjWnxxLA9w7GU10bxxRVjlmjuqECubCS8oqpDPmc3SP7hIeQqo'
16310: (12)
b'SdHLFg2Vfdxu1/1xWe9+yDJqDu64PXsdfdx+DlY4bg+mXm6lHrR/6Y6n9WHzAxdWAqmdTR'
16311: (12)
b'TuV2eN22BPjyw7qFbIHD48aWBK4Hm7PjxvL+ftGhWWRlHAuHaYcVWFn/fH9cNzdza2uJgt'
16312: (12)
b'1FeoN5lHxnEiq7jmCiN6ml3DytfUxWSiyPLMuba+QRuZuOxsrDDRgg/DGY575m2NNnG4bN'
16313: (12)
b'bns1/Eo2J1uJy+sjTDYm0A/VpfQHS/BzRcdoACfVmj2ML684TIsTv8kPFAwPploFgv0Uo9'
16314: (12)
b's1Bwu0rJ/v7lBbm6qlcrfh6H9cO2OyGXqSSS/lPqTa2B4Yi+74nFwWQZnJ1ht3sT9xDyuO'
16315: (12)
b'7UQiLbPpEAoJ8/PiAnuRJocpWdj9nbTNvZnJi50YF6RnSjQ2NpOXmNqnk8Dq/3w5n1fTa1'
16316: (12)
b'5GZ92m6GV9oeUI/xkC1NXmQhkCtRXm8i2OWFgAt5c79zgS+ngriwl7kgLujlRBAf8jITyA'
16317: (12)
b'S89AHbMGZ5IF0gs1mAfChUqD32uu2RGRDRuUNZb4i79ecioAzQoVlATZgOzgN8eXGYS+cW'
16318: (12)
b'Jf2t+xM1hPocES/fJJBIlUq2Q9x+TMYrWARHB3r0qeH6gsclNQ6TFGeKjgJdKQYE//r2Q1'
16319: (12)
b'bNWgUyKierT4zBJSqXmWfeCmSrxFQQqREuH02hzVJPbEyhFYG8PzHIeS0ISuJ+PQJ9zpUa'
```

```
16320: (12)
b'GB5dHVhIcJL4yiMis0OMTmAKBWGdHvrebm5wr7HVQLRf5jjeTLjStHZogzj2LzRg4+zQEv'
16321: (12)
b'5Yhmnx9gio0rxSh2mtYoxp1YLLJife8HZ65mgyF2q9456JjKRUDT3nBoY+B60yS0No0WAU'
16322: (12)
b'gnVjUcuFIAuh0zYKo5ivrkq2pdPb/uU8mCFAdWZoIWcesEAV9/nHPuUcGYaTKfGgjwo5Bs'
16323: (12)
b'5F6aFTkmrAI9vroeRptdPSQe0kvUNQ5y33B0OgnF5ervRRdPCXW9pihHttMQK1tgjGV2rk'
16324: (12)
b'Wz9Icdk4ugqH2frWH9wM8o0KD4sxqCMTg4oWBlf33KPFjxoNoYDcYyT2RvKFIqOaTNxJkv'
16325: (12)
b'FbyTq3tOSA4auKWk1In51aAb3gXivCS3KPbBz0doxaBRBVZhiD78N2ZprcRxeb5IaW8Qlu'
16326: (12)
b'O+pyp/7PcwcnWyoKGGXLEoF2D+sLO4ospzO9RYhQaRriNdGaZKxLohMGNtYhZ8ajSvOM9E'
16327: (12)
b'iXRM9qwG4/8r6YrYRzGnYY1DfCmhgZDsMQT2oWaJH3nc5HxqjtMljQ3dmur9xbU4LGQOuR'
16328: (12)
b'FRQTdLYzCc4h0kCGiYUBg0JvSGjZobahJt9vdb1akvY1xhC6yjgg1BkC9nh7gZLsdVaS1g'
16329: (12)
b'klvUMurHcPKDVzIh551B82eq4Ine6+V+YCTMEONdtXIJ6SNwBKCHVuQ6R0CAaHl6E/nKHv'
16330: (12)
b'QEF1SjBn+YbNEcSzzW93pOfpNVd5xqzfscF5uKAYY106/d/4WqtuvuPO69dp+r850CH55P'
16331: (12)
b'CWO8aipEU/G3jGo2ZmlnnsHs4em7vAjNvrzGnmN9g6a13Om57cFZm5u8Ch/Q7uH9kpZKXP'
16332: (12)
b'geDMZd3pjG4kK9nySZrb98bpmireVbqCRyehEUeLOR270EyTLYdn9E0Zs09fU1SBHlBTsw'
16333: (12)
b'JT4/toigdfwz1XNXrXP6ZI9aCrP7J20NUftMw70Gr+CLM8RIuy7oyWgnmrIey5yUnVBPL+'
16334: (12)
b'TH4egH2/IZIpRPfCyqsfajV2fqHnNAC6klUWtrUTYiwVbeVoFeIE0Y4iSTRDRFko0MqiES'
16335: (12)
b'1MnehGh8Gu0YAVZ6Ihq++tNBQNipF/E3fbJlGDRCTLCLGxNBFmC2weYVE8cRA2keju3frU'
16336: (12)
b'sk7CVRvW8iVrLeQMaUpLycKWcriKWc4OJ43RzXCBwm55JXn95imKbu6wGzHk5GECcbCj/B'
16337: (12)
b'yyiNlYjdzWuiCchiu5UEEvuh3A40W3A9KY/p251Jm5bxM/R3au9VtoQPCYtx+pss4Mdure'
16338: (12)
b'TJfcJg/Uh/LkQVsKloDVOIY58YPc01fh2yuNxLXSaOmgNJLehWPeNcjDhoP3YaP00jrVuM'
16339: (12)
b'v9icb8GkXkUC9TkPFysv0Lj0M+IMbh0a4lO0uwbFHZT11mCwu5KmIo9GZP3bGjEg3/Dfzr'
16340: (12)
b'pVskQe6kW+JbriLEFOlhfBXhDJDoapklwr2D5F6OO472iMRdQdiYr3AFIenQucGdRNjUnn'
16341: (12)
b'BpgQDGE5dV+dU/cXGHeZBb+vDoK9lyZRDdvtqJgYbd5nR+49JM5YLRdRNuotM/0PAetMIz'
16342: (12)
b'a0j72mEIXT0cEOoHAZ27U9C3b1NckvPwzLkHJtxpbsjAn1YE/vfLFVeRE82xnm+YCxdkaC'
16343: (12)
b'vpykR8+3LFBVnfv1yRWUUDa1bDbd9deEbKVA6/LpVVgWMGN2Gkwhj5KGeeEZbL5x6Kw2B1'
16344: (12)
b'2w4ImlM4M8hO5h7xQG2BPjhxnobOA0yku/EQrhnPVSpKh4/S4OBxClwoQX4HjKR36GUUKM'
16345: (12)
b'QRXbZx3/vL7ty/7N7Q2c0qh6FxgZo56mV34VrjrPD0AL1pZ+pWjs7dobxTnWMalw+MysMe'
16346: (12)
b'daKYsnQo3DTRTTxblMnofJBrqkuFu74HjW3XUXkzDZk6/Xr3tcM8iOPAIrPQhnfW7whMLM'
16347: (12)
b'Bp0tEiqUXkMBUx1Nbd5Z4TPvt1uvRnJ6yG3DIPbUoe9g/omUOXM0eTjHQ1+HJr6soRpNHH'
16348: (12)
b'JdgdD+ZoywQjn/nc88TX+vjGbfJUIAk2dc64AqCciH5TWNqqmlTome12xXCZjnkOp1Dmsj'
16349: (12)
b'buEdqTedxIceNLriBTkA4vEn2Ib1UuvEM/H574wNQS99JCqodtUwtFy0LOp78NT4szjVlu'
16350: (12)
b'ndyFK9ngkqS75MxCds1HhxgxXHgNsRd0XZxDUJrD0/HCdJp1c75NMFyOnLA8Hc36E1Qo82'
16351: (12)
b'DBAILG5o6YL3h5ETQqRzct78ChZuBoHsZmk7XkYs5rVNJA88Q7R09LLhcp2WmgM9JZoHPS'
16352: (12)
b'eaCnpKdCm9irldA/89JRKhCWbnnhDNQeT77nAf1JIfQHngadSHDtJ15VzKHJ0Z952XJaBZ'
16353: (12)
b'pnbUJmrHidoSlaSzLtqZA/GlLS+pOJS2T52fide/L9nPmaimgfjWcpg0+8b20i6fzEq1cm'
16354: (12)
```

b'gWvTIdn2ycop2frpi0mHRPbpN1MqUohfTGQS+j9MaMwF9/QGFYtZIE/rw4m6voZQKR+pXR'
16355: (12)
b'BDrRtN700ejeBoaTa75utdsTRmy2ba8gYehZvfcKADNvG+DEd7vsF3aqZCBdWL5Q9Pz08B'
16356: (12)
b'QtbJJBTFcLx863p7FyZChALQnalWcGkGnqHpvXELM6ONvqGMOk4F/HJEIA9vzGDUwrejuV'
16357: (12)
b'Ob+ZiSWrEvX9H0CMS9ZxmHj45VJNwaLafJJlLiSavFqBLkJtgIGNItTZnveImvaYmNl/ig'
16358: (12)
b'RAEd2wtMErdyZsxAomUzjzxxDWSSTdy32bmZZClJtSJWGjosiJFW05+S3tX0x0S8CyuVFG'
16359: (12)
b'5nl/ty+xlW9CIgrOk5eItA7f628XxnLGVGnLDyd8U/dU88Nek46Zgz8un5AXVAf+z/EFdT'
16360: (12)
b'BY4C8CxoB3sBZwocuXesOH2VAkfuHctu7Qtaa3Tkw/Mu9xflo9HoyIfjxTlXKnDk3rO2ps'
16361: (12)
b'o6cKLAkXvHYqfUCVgocOTesOImMJ8D00P/dGUBbQbisfP6MNpCmi4CJ8IOvApuZprn8SnI'
16362: (12)
b'Pa8sYPrFCMRM4+XQcZdFjvKYQX5aQ+r7nb8/lfWIy2/XRgrzWwy9KrQcO5DetbnJ0X5b4+'
16363: (12)
b'LIecP10or1rvZv0XN5RG1Sc1vb54tJ05NPUymUU5RXBLSOsiCAGLnayKNBlaLd8ovJGLMx'
16364: (12)
b'GzATzsux33ujBJNJPmFcf8k4OiqMnpWGNWHC1c4MWtl9GBzQImShAFGpy+vR/MOqQG6J0W'
16365: (12)
b'3kRP3l9XAedeOG9h23IXQP6oDQhRog9JGYtW3GFb2pIfpmIxP3Ajm6ifYxskSxM0vpWD0S'
16366: (12)
b'oiWid6YaQ8tiMOqbfQrm1L2szdJU2GVtrni06zFjmmOqvSrUpo6bOFwQQZPvtn1oOktDh9'
16367: (12)
b'EDFUPfQoJS0XtHC7LROYjZTeNosbspCdg9pKn9lCsDa8Z1GPbIVsiLn8sJXcHhsrfrbiEr'
16368: (12)
b'V8j/jvdkZxjr40yuEpXHhtBZ7ICQwwTcZhE+MR6/nblD5E/rFyPMnQacJrLXwxMFjogmgS'
16369: (12)
b'i6cOZvXifx1RNoklUS3TzhWvpUUNc8gk9pzAGK5NSFxNh1qZA+nwc3OYfaven5JhtEW1Xu'
16370: (12)
b'm3P5zDL4wpLdxs0y6NGb6D7EAmE9n7ZmUayYwUO0P4HqEJYqobFtwj30aEPRHBhJPchmBg'
16371: (12)
b'guomzWfokE3cKAmuW3MsjXCURb01sZC9I7M82fMA/Nt55I5g6LZpLeoVquE89iCuBD1tNF'
16372: (12)
b'Ojo8UUdF9R7U3iBrd1h4zJazQLryrBLfgl2J5wEYFKISt2IkGGxOvDgtzVNP/c4rUluh7G'
16373: (12)
b'KZq80mQ8/OwGJRkOCavCzzoHMyK/Fvw8YqNMYSO8ZEvzOc1wMS8qyP2LaCurUCRCOqPLzo'
16374: (12)
b'HEMSzuveLNMii8LSPOTQS/MctvTSPCU3r2kgT75ZzYCNnpQcTS5J2CXgOZ3ffmcjJUdXYz'
16375: (12)
b'qNVj+LVcIGARE6OWo+w/eReciTJJ1abIdbveS6SDq5ox7+7fq6X29fekCvtQt4ZchRXHG0'
16376: (12)
b'NYfhuhbV4Hv0uAeD1UutTM3D9i2+Z6GuAMrgObVEOM0914C8+LHSqIyxM43q2zErzZAXP1'
16377: (12)
b'KNRtde5pojb3tQelVCEFUfuwbX5zGk02eskTPuSY8q6aInPSwtR+Mhf6f3+hFOd2WHAz/6'
16378: (12)
b'3Q/0XJ1YuNf4VsUK/1H2w2u0No/y0YZX8B2dwYfckY07gnOrBnltP8MI74BQKdvWIlK0jD'
16379: (12)
b'0AbkeLSw52jSGrZql14HKxdAF0mEj7MKpUMN+2MdoIxAa+YXufWUzlhRdH5aSPYIs+4yoh'
16380: (12)
b'XFT/th0uyJfMQzS1sdY3HFMbi2KwGpD/L9verRzkWeZSKl1+NqldGNECqcNUh+/z1Seucp'
16381: (12)
b'FIyuqVAE59Wjkv/m6sykUu/V02qZwTbwBNcnwWgL5u3DqCzNVmeHUgI+N+1MHn4YBc1JcO'
16382: (12)
b'GNCf/AehX4nJkbBdt7frlFArOvNkTKgrc4dIRrQekDLOHCIJp59d/8JGl9Go3FMyscky1o'
16383: (12)
b'KgA+SekLdoKo/IWzTIAP0WTY6+db8xygiXK+23njmhgkZ6Bf2/cAA4je/gaMg5v506kwVw'
16384: (12)
b'F1myQzY9YmA21x18vLn71vFmxG5dNEfH5g2chh86CkY5ehSH0PhOeRTOwSbHPGHZhRdy0M'
16385: (12)
b'qGUMKIyN5OmzFp/HzYDSe7WDa3QHgzBoN+DInboo0ZXiFGBvjKMJ/g21+0hVl+F99qhUmC'
16386: (12)
b'NbZEP+U+o2bnMNGpSkerBrMg1H/FvP3AdGclivWo8w5+dC5PIZFOXB1I7Qox671IjuK3n/'
16387: (12)
b'xBBnLpLatzfjh9oi5JDEffQUIrtfTVoG0cegF2w/DCq9nmBKkbnpWk7D2vDHArh+mWP8ai'
16388: (12)
b'1VgGfTZG+xseX6BcSttCZtoZVsUPNRzVpKXU4Ms8VbRCXsqtL0v3LUM8cuaM2M/rxwH9jE'

```
16389: (12)
b'wMOXYoPFpvCbwb0LVLP/9bIu6LVG/WAHkVqbtlB1sp2BeExrTeBPzPB7PSxwVT+637hoXD'
16390: (12)
b'7JpqLiTNuyfcSgu03KnvwWhS4UE5P0MAUzXaDpgeEbMvO3dlf6reeFoZyla8mXGjH3yaEb'
16391: (12)
b'AqdNrMk0dqqmXyKKsNLb7VUGBoBHDYdj1XhyYz0OetWoVrLRCtwjksWmtrkke9PlMnj0F1'
16392: (12)
b'LJLH6MWpVfKobF7R2B4jbQjN6XFsBLvMiI1XyJc50dEKOTTVR730gNgxdlASHvt+fMRMZc'
16393: (12)
b'Lfnh8I4HHHD3gyAITpHyPVBtqIg0SzyQSRQQ8y0xq080MBnex2GMeHP63JoCVpw2jNF036'
16394: (12)
b'nteP9iCwp8Ia+hgLy+iBE5ZVAxYWkud2sThmKC8xWxZ753ZFN8JHvhx33+3tyWRPBWcOO1'
16395: (12)
b'wO9nSyp4ILh7109giyI4LxuIP4ikxvzyEHOrgiejydzRVMqB7diToTpvmPPeS2Vlck4kfL'
16396: (12)
b'GLRRy/PCfAUd09JKV24MEOrCVNE3NOW6NXyvKFvfVkeF7pMWSwNo7bdxSFB+LRLrvoXDgu'
16397: (12)
b'prkVs6rhVRq7jWbTTUWkgruBYRta62pKi3C0977da6Fx3PxqqHauvAq7agTDtDu+DBMvMm'
16398: (12)
b'Eb4jlQxtKBwhxFThcXgUexl2GsOjX/eBqvAIXXAv7CnZR3alvM474XPYLN+p+Qr5aGlVvn'
16399: (12)
b'MDhPLNFX2rfJeG78vX+tbF6ZFQnBaJi3PqsFCcFrlVnFYiXZzWbVScFrq1BFoZji5o61YK'
16400: (12)
b'2joIBd142he0dS8FbeXRBW0dxH3mUjDpNNMASa9ZWMzVERfQdtSaIZEomAjkuH7g3jFP9k'
16401: (12)
b'xJHR449ucJTxFiKvukTeRI+gOFBb69tRzxcLZ5viIZL9NjaH3iod5owGlmU6LxgNPMGLI2'
16402: (12)
b'vasMHSzvSGs1bgFaq3Ck7UuHTW4/dwjJKRCYMDlQ3cHfTgDF7x82iZ5DTJYg/VITkifqA2'
16403: (12)
b'RRzyEi5DBMl5YIzyEijNFziHDvnkNMzVfggI72CuBSL2EUGWiV5ob0sOcOV3QIq2A4x45v'
16404: (12)
b'ZjDkoAAuHC7IKnfI/vLHRu3CzpbEUVl5kpCXpq5II8A33nkeB9oGVggXRQzt162BY0r3FB'
16405: (12)
b'ld1qT1M49VZhBXsQxb1wUHhMpgAH1/wNwCoxsEWote3SGwsvhY50F9+N5bkwVZ10+KMWE3'
16406: (12)
b'3ppE/m/D5tTcUFphJGInfiXjVE8UIkC9uQAt8UlvLsxJa12a1brfdzt7A4v5DNpPBATVx8'
16407: (12)
b'FBiwAQbzsg0N1wxvRBXq6QK0NbzzqdOfHK2JgDoF6/gDKnGO6s7ERjaqLG/L1mOE/pLZ5u'
16408: (12)
b'x5EIXtRsnl7DKso5Uh3e+ITbaBRFC9d7IOhVn/QeSANautOM38G0EI3syOsl7eJPlfjlSx'
16409: (12)
b'Y1P/WyfpnojWLnwN+c6UhfjXJLhpszWwtEcjs/6jZNIh2NLjmUt57wXQWUIo0MR25vAF82'
16410: (12)
b'Ho+GSPE/HGUJgcms8sBwIVSVQF9VfILKAgUkkEO0mIc+hUdSwdEbFgWScuEEYD/4syDzJk'
16411: (12)
b'De5qux2Kk/PLlz5pN8FiC3OUo7zye9/dEw9ON6HzaY2Mu8hf3xWcL5O6b129uPrs7IiA0q'
16412: (12)
b'UHV1v9fQyU177jwJJ0bpSN91a+lwoy5pddhxSXJkBpIRG/d689ygYf9nRXrUB86nAPuz2m'
16413: (12)
b'WbJ9vIgmmlaL1MUtPhDrqkXs2ncLymRKRNLRBbqWTpnTFLCSw9K7bcheXGE2vLahXr2mNj'
16414: (12)
b'udFFKKlgz+vTcRQeqlnEvQ7Spep0eb6MWAVznja9ZqJ65MoKM/Tqyd0pM+v4MgzmEoP79f'
16415: (12)
b'HenJtvFh62p448vqBIoSbSs7L+ajJFm5udIiTLr5DHMRJs3zR6cJcd3OJRGLTi20zUie6K'
16416: (12)
b'I3NqU9sFSO+voKy+gvLpFRQiiOCx0BHzSuqIG4vtWN7eq0kVbS7MipBsOkbyyRgJYWt0LL'
16417: (12)
b'DmXcmrmbG44LhHnKtEb4NN0K7iN53RItSbzuhOgvZaWSK86VwkW/2mM/jRm865oSVkuO7s'
16418: (12)
b'bW+8UOXMfaTCfkZ2/AoTGw6I3wXNZSpUUFuIbW90sHoVrCIpeo3xYbtG7W3VzCvNOb8O0v'
16419: (12)
b'9h7rkdL5tZ7Dv3LTXzIuaOj4I3cyOG741HgtSaJxE2Bg2H6Iwr11OPApgplvhHNwI5OhRc'
16420: (12)
b'6DUqBqpP4tWKjjryJRmXc3Rve14CPIjWyvw7XtQwwVHJ2rGSpSxFQXpPpf3Ur6Ch+Prucn'
16421: (12)
b'2uqHH46PCMg8cncpYWDidyWguMTuTQmc5V9EvRCXVNRxnCaK2hK/Q+85lOFZGlmtgoIrRO'
16422: (12)
b'B4zbuoOvmrnD4xYOMLrmH/kZ6X4oUH2mpcKgAR32xS0MsNlHJ5RJ6+RrOko+ctPZ7VIX4W'
16423: (12)
```

b'c6U0RWKiLPFBFEd8A4+Q6+Sr7D4+QTPAzP24s3VMoomNvQ9zrzzEAPmnjhQgAUsG+xnWdq'
16424: (12)
b'mHL4SLMysoJd/ZS0fop+ZuhvA482ObPLgpA7lclqOpxPL7x5ydxdwYIxN1fw0NRW5g3oPH'
16425: (12)
b'VbQHHJPSjsIqNjtKT7Xl1klcN3dLC2UHRUfOgMoseFsuUyQlxmQeivXE9EOG8vW+508mpC'
16426: (12)
b'+62tuzw/2ojxDkWpzz2gdspKh/EdrYzHXXrq07OkFxOgJb+VlrRK1KWEdZVoe42MpFucga'
16427: (12)
b'C9vB+FcMOAVid9bHDTJvpdlKJMem3lAmH86qExRnIB5Vm9CpzH/tgFRpOoBUea3GJW0PmF'
16428: (12)
b'x3yluWQLZx5xkCsqUIwpmsnNY5oSlhFqjorlPC8zRs2sZ7WC6hlxuO1/vuzMoRERo4rdHL'
16429: (12)
b'm3EuTINdfkiCypRikzzxmjwp9CypcR/8+Hbse5ogQ9i/iP3GHFbNL7xqxVczHgHh54c4j4'
16430: (12)
b'Lm/yJfIR+yhiZVFxbddfg8BZxIH+HbIhysieBxj9syMsgKiwduiOjkHO+oon8cUsFFmILy'
16431: (12)
b'oU9kvCiRLGYf+B9uHCnsXsc8gSdJaaNYQqkEU18bDehyyJ0u0WnHOaSWiYx+9CgqNoMPI+'
16432: (12)
b'SI2Z5jHrBVolaoRENovZJ24hBFHicJXpFVId5eSpe+A5JhFoFjN3jyJPlIzT8NB35zeJLx'
16433: (12)
b'LW9nN8kjNGu6jSRfXgdB4enoWVxqzLJkQUVcjTJbTMOC72o191+1po9itXVKRAY9YwbIQT'
16434: (12)
b'Nbpv3XFgolRtM1Um9G0q01ljAkNVGVaYkNuqxiAtAVeJMbKGoJSwFDUwjKzWFIQSKovDVS'
16435: (12)
b'C9bVOmMG2KyjJRlpLI7KsnmKCiRvfZshw7jo9jpdTjI6XUwWOltLJwUEodMFJKgYp9I7JC'
16436: (12)
b'2zeSpcwlQeqVYeR0ZNSJeq4HS7QJPdCxt5Hs5LeOyNIhJtJXhpkowSuzOmRnP35Wj+345r'
16437: (12)
b'27E417E5II1DYkYPxOC2y0Q73+PU1uqujQ5ftgzAI/5ua5bIkc3V3ewgEL0GIgx6Hg+l3E'
16438: (12)
b'PDH3dQ7Hm3d1FoY9euIKVS/Sw5EBB/RB3vwPXfbB7IHxfH+KJnXQL7WVkEIdDQrU/cBDBD'
16439: (12)
b'zFkQbsHNP2CppCaC7Jw8EkAIo+ome0e35ZRhHPfbgVlUF89Rez8BYWkGLAvqTrr7zPqQu3'
16440: (12)
b'OfX6ofgCIonhHJviYE2iZuZLve+4mEeIt45i9wDYbNhR+7X+xHYKAYrSjApw1JWVJX9l4p'
16441: (12)
b'U7TNecMRaZeCHBp9N2rfd8IalsJRi+0mTRNXklQEU7U7A+UkDYvRPJjI8svtgjRzccwsFF'
16442: (12)
b'q8CoL7eeS1slV20p15heQAb+bdufT5H5RuFBOaymmFXyO1XzefJ7dHdKClrt4i1A+i07fu'
16443: (12)
b'sdO0uHDTvQ2tZ6kvzu9fUVv0Vfn1lCFqDQGf+OJno6df5MA3L5d3cMQ8qnWCXxBlYNutuH'
16444: (12)
b'tdmFoUdXArYGvLoTcGXg8bo4pFQLTTNGsB2dSWuS36NdziVpn0GG0DnkgJBFBOKrWxAgWk'
16445: (12)
b'3Oo/6/Rz0MCkYaBDJIzyKzhNeEolfByLA+bZ/7yPIyJRwkLEC6ATQnS3fjc9A3nyFsDMOm'
16446: (12)
b'igE82mcXnpUtABpgZIbVJDcssAw4MlBjpMogyzi5slcz6HjvdkEwvttwCUjneGHokOGkda'
16447: (12)
b'/BcMfmwVNguhdpFB0NQCUYLy+m15vbz/i+RlRzoG/dcDnsoQfsZbSqUmG8cNXqJaxj1dPA'
16448: (12)
b'Iif4qYVxOq2hU8TcGbjH4dirDp55cdr2mzUm/EMop4mGUcF69kz2CunYzag3XTHvwjVZlF'
16449: (12)
b'PvoxST5GrrxBTH9Q76KmGwLAYMtztjjnR8jnKWYX33kiI0o2e92N0mz9EFXjPSzmqD32K1'
16450: (12)
b'gYnvc+h2UGSxkQbZSnGEGvIcm1dOCai9SZRiZJqh6Sg5kCK+8BM5cGWQvEJ1Ys057NaHDR'
16451: (12)
b'OaQoF7jnqXkrQeKQoCvmEarq78Dgi13wBqH7E19Ggj0Tq62kmsDDzuIimhthmlq2AFMTOU'
16452: (12)
b'toIggor7fL38WwtnpGsLY6xtzz0j6NuNh0YaN50Oz1u5uhHTWQMMcqtUYYHL2p8pmeQWeQ'
16453: (12)
b'2epkT2Fzl1wtjsNVMzpgv647O+uYoZqcw8UDsiZR61OFJzNR3VHuRpfxzGG9WFQfddd9YH'
16454: (12)
b'JFnEgAMNmXt0Gs/j/C5bzxhllcfH7icOl8zm6GGQUQDe4akfTsExcjMertF565VtDPrP6m'
16455: (12)
b'QrCn18xxNSFg2IyP3rO55QrpENR05aPa8A4ZBkKdHUkKEF54qOygAVaECXE/IV2TSgw1cp'
16456: (12)
b'qhkYk3s685KA48Y9U466vSJnOPhDxxwqZSwv+R0SgIhOehLHruIc5CflF4yhzDzrBeMpmH'
16457: (12)
b'p5eK7pKDXI3a8SZgPqNVBtwmMm5SLZaSuGDKSzB4SWsBPDBeJa77R0mCeRfjat4m09eJPT'

```
16458: (12)
b'IuHhgKvnT1YLj3/vnZNVfe1ivPfWrqrI0Y1XT1bzaxfXwcy8o2tW41nfe/kEffmVi+tgbD'
16459: (12)
b'7IYDkleb8x+kTjvsUwZmYQljsfuDKfQdeKgKBtOTjoVh7wV7Is7L0rAZQbchzrztyMM+ar'
16460: (12)
b'AG+6GvPJGil9LbHrYWaxMEVzpf6tiN7Q3BcLE/jzrZBMhhlptuOsX65YL8f6fjuxYHdDsG'
16461: (12)
b'Vde+ZVRAvPuTW1WK7uEPL0zkwnnLtb46tyx5iOT2I7X7RIvd3mnyF3UFuN1RRi1UoQSK/0'
16462: (12)
b'5MhcpfSQI0pPY4n4lHG+BBqrQvBk7VWhCu60vaqjxWsVSLGsy1Eo3aO9clpf9jY38PiYO5'
16463: (12)
b'JL67EJDwXxS8zGpoEcjt6gLcuWc4NHNmrW59hALXNo8AuV3UDaOs1CsovFWM3xIYyQvDTR'
16464: (12)
b'XaCAGKK9QzpAtqH3tS877+Ij4CwermWxfsbjHgC+Xo+RaBe60ZyE7kcJ6NER5aacI7rd1w'
16465: (12)
b'FKb/+gTPLTgHo7ewXdWFFo8xts7xU8axbr1jEyzC+jU4dTJDGMrEukZ3jYcqvJ7dSCPTxR'
16466: (12)
b'gbcXimWVpw+DMeNbKFpsNDPeqetwc/VYhuox7MJlnxk6zYF7rJMUw6q/QMfsRZmrdVbttE'
16467: (12)
b'3ie3UyT/OIEeKAE5Tc8A35YM65oD7JaAwh3QML6RT+/NXlPFm706tBiOMsl3Qgl/1TTBlq'
16468: (12)
b'01XJsPLEBTMJyK1yyZLvFgtYf4ZMzxMeuENF3Os7WtrEL3hSB7Df+p7n1GFuF3jqyGBlun'
16469: (12)
b'RIdPVuTtAtHDBUfwkMY9N3wFg6XAFDmkq9Ots4nwoW3yNlcLUFTr/cskOn8UrjPNN/MKdX'
16470: (12)
b'Nab2Me8oB8LBnGqm1zsaDYZb550Xpq/vnuNYUHQe1eHXjYV9yLUlx2HWc+LQfrh+oPGpwv'
16471: (12)
b'1rGyyV/rzuMQnRTmcB9rFVBsJQG4u6CnAka+tw733m6Ctpl4aBrirO6CzAUR6nDvfhzh19'
16472: (12)
b'lbMTMt7W+0HyqwSiDRlaRUeGDEyTPYFIKQ6nN22jwXz4Q60dNQzmePKu0fO7WU+oYAwvrB'
16473: (12)
b'SgyPUYivDC3VhLlFEYN1ENRtMRVD9tFjdNDe07bKj4e70aCZ13f7UaiXZ+Q6FoW+t3rJ1M'
16474: (12)
b'HXqtgSzTwBo/SsKqOZojovfb63WMmt77b7HlGLJSr220qaJ1CbF22NOM9LEPOqkig0ZqwK'
16475: (12)
b'AektSjZsU0cikoFFjhkOfuEWNLwMsIj3sRz4tRhOSs0iokRs/MkQQz0qlrgaKdgsLwzajV'
16476: (12)
b'oI5wKe9q+SJz+GjxwsHjyfQ0iRcEWXsIvKCK62lzNfF4NMV23uMlQOgrBo0CwPRxHxnAkd'
16477: (12)
b'YtT9NRuTLmg7mB2iQCn9pcynF9A6FxhgHcTUWVpdwV1hg8SdLoE17xfezvI0tDdh0AA40u'
16478: (12)
b'iqP8rnuS2S6zQi0QIL5xi0QskX6Can61QDBDevUCQZ2RVgsEKAi9IsAmenNFgMPFEORZQp'
16479: (12)
b'5hL7oPQ6FGE4SrIkRJjfYp2of5DiwMMiEEqIR7rYEgIcF0DMSFtRM19ZL6D9XRIRWXh23Q'
16480: (12)
b'g6HLEXDHNkpk/+UxuEZnd/Fr2I0hAg+ZqtccapSKXnNoNR3lF7LkosqPArob0CcT1peLOs'
16481: (12)
b'FK6Q7KQp1FSyBu0ARPToE09sRzDZiLBkqTUGCP6BXttd18IM1A3Pt78RgzUOU180utkKBw'
16482: (12)
b'L2qJBFnydd89hfzFFHevnCM1rzEfwSv/y4SqGdrrQWttNUlM2cwBooNfbZlO8e1VLTrRqp'
16483: (12)
b'alg6pFWp/2mCeH6ByHpqNhtgBDnr9krDMAodDTRN/kMmlA2lYGBXOSHPzEE2PNIUw8MciH'
16484: (12)
b'c63LpSXiiSc0skM88aSnaFgtDC0ekDPRbYkINroeUdNRCiFa9wr1/w+rTtuH0A+q0kOU6A'
16485: (12)
b'TsjLRfWjeEXlp3QFhaJ4Aey+toLEK9TZwn5hYae4SJo8VhPJus4ITGIlcLtSuHj8YAB8fv'
16486: (12)
b'EuSFR+MwUgvHJtN5adEATC0wHoXK2uORBC7Q2GllwXP/3F3OAWZUutyQ29EFipqOyo0ezX'
16487: (12)
b'qJ1p+Z/Q71GiUKntO/Cc998SucGbe0ml2tDBCOXNeKvnWJV2b4fgJmfeuj6x4JR9ctEh9d'
16488: (12)
b'nzksHF23yK2j61YifXTduo3WPCykD6hbRA6oLywpZ8YnnvYH1K17OaBuY9UH1K2D+L6yTD'
16489: (12)
b'A5oF4GSCKbW8ztlCAgsxoCkeLVEDjTW2B5IKPBA6ULXcDMPqgXcCkMvadeIWGPFY3+4KsR'
16490: (12)
b'BfFEnW1O2nerhtD9qgNCx0oguEdU0WWZiCq6LFPTUWWmxwOGr/UzzcRVD8prWP0NDTlJ34'
16491: (12)
b'+wlIdB7aiWydUDg21rwaftBUKK02au0NEZ/ZVh3TqGUt2ZsyRkX/MMfGsZdpkF1tUMpDG8'
16492: (12)
```

b'8XSmduiNwIrAugqsNbzrRxahmGDU57MA6/5ApWbCRJzVlWwzRfPVJY/4dUAWw1mpSCtFHw'
16493: (12)
b'ZZL8TkIcL90VcTWL8xj/nZAJknZ69itZ7QQZkoeX3wbtcZU7DSAEdeO2kujK2Ni9Pl3t6p'
16494: (12)
b'Vk8tidERKiSB1AJs1NYF8+5VT6kQpOiXkFEpOfCrGzvS619vXYF1ofKHTI2uD0WeRteHaj'
16495: (12)
b'qq6RUZZ72DtLCIX8J0pF7zFChsHxHa37PHejKHE3JFR4cRNEMeIlkl9mIPax3lFFrMMRVq'
16496: (12)
b'3k0UVmFZAxf8kG/mDh5otPiQee1UkcHsxIDhch2QSh1EqEr5Q2t403pGS9rrGYbQeoYDgp'
16497: (12)
b'7RJgN1x1Uy+BMU6DSHsOucLZPhfn082jlT4Qlt7jjz4C3j2QbMIByC1iZcZLrjF1NIEF3D'
16498: (12)
b'mqYe0PILeGUFOrviaFNQw3WHOzJ8ix7ZWkIOd6ymGvALlMtUo0qBXM40w9+JuMw1qk1s0R'
16499: (12)
b'cN1/emYr6iTSFzCMXr4p3KXqSGlAMmKBGfR4hHGTWvykDqMkDo2oAZ/k2w8Kyun5wn3vqS'
16500: (12)
b'B/ftt5uc18ng7YtXyDxdHggjMmlB8vQOMgKNDIxXpI8shXlqPyWHG0srQdvcQpKrS0tH+e'
16501: (12)
b'lC9DnZMtjoqJLJPl7EjFF4uLI+hne9wz1Pbm/XI1khp5CdegkQgos9MNTGIb4wk7kcX5hJ'
16502: (12)
b'efbeomWCb8zsaNY6s58pH+Yt7bfet08tZOxb5SrIqrLocUAfoq0vG4ufoebqmlUtHe7MYq'
16503: (12)
b'FaDHtVnkvK09vEcJbpCHG+AKKVIriwSnKaRO+IG1KpyBXpoCFPAnnrbqc52V4/Nl5RKzpo'
16504: (12)
b'bOgbzIMqU2L2Ni9e5tWQfOx5YzbvW1+Q1Ap1ZYGgTxsgVqdTC+14UR+GqSFWrQ33lmZtUq'
16505: (12)
b'IVa+My0qsNcutGKJMKrW8bl6JuG3a4Dqp2pFe2jWN36pEym1SL7m3kCjadk2ZGwKvPqSX6'
16506: (12)
b'Iy+jZA0Vw2v215aQOt0uCakhg+6vTPvpz91tCsFFQ0BRAhWrcGiWNO2iAXmeoVEdN49GXz'
16507: (12)
b'OViI6Pm/369HDZWaQhct5SIKPgpKhv+n7PNHP01WgAj/5h81XtvuUCKoYyNveeOUz3BmMs'
16508: (12)
b'WsRFgq0xRRRsWFBboQj0mQboQ4PoQ4X79r0E+w0DqIPybFyRWTdKzT3mwXXPVqh4t3KexE'
16509: (12)
b'9+TAoBwn7lLGD3u9f11zeCCwE90hjk9DAcO7v3N9w6lNEo2Oe/xvQ43CQvfLZskrys1/uX'
16510: (12)
b'oDzWBuFZrmATlcGxnmPNQfpetcC3nz4Rf+rMzZ9ZigGBlLnyAoP7SzQPMy7VNIy0XsxOQf'
16511: (12)
b'dva0wH/CZUxuD0+jaduLPAxkh/9DTNlOzhYRvZQS+YuNFCPMNFxOxOWNHLRKvtTN2xO7gL'
16512: (12)
b'ajD+Chkf3V/mbWCZ94XRWAWwbxgvAqD7KeUuUnxVXKL3zhSmFHwVhH0BuQmAvnjZpcbfrZ'
16513: (12)
b'PNFD1Oz0rx7IPJtULsWZVKITpJrcKjNOkIJVFzDapU6VDse8ulQnS6DM6Z5qZ/NPO/DMCp'
16514: (12)
b'Cyf2Tbmfolt1KUpYkCfl7l+p7GeaamKjiGytiLBF6YDxqXgHX52Kd3h8Kp7gN+UKutmLXp'
16515: (12)
b'9FQoPCjBLSC6rQhuzNoaj50Qk4uAuXcUynQoVJDrHuW9ilyVF/rN3b2GUORjAzZhHFhxzm'
16516: (12)
b'ib6wlOGOzlUYKceLE01RGzS0fxPO6FJB1v7ozgs6unnB25yRxMcHKOnRPVDMVm2JoHXMPR'
16517: (12)
b'TVV3EoRkTGHRUBBNO6b612zxxmhwKqhtxZtFg0aqUO1KfxvcNIBh+LtJfMA2rPqDbYCTUF'
16518: (12)
b'kphZrzNINY4x8G/6B75NisYxN4milcDJ2O9gYAJw4r3XGe/OflFL50ht9EZQQ9r39obQnb'
16519: (12)
b'oDQq9OwLw5XPLD6NNF4s5FXO2zzoUz2mkVxnjte5GMz1hg9HbQaEXbOPUn0qqa1OEsdhe5'
16520: (12)
b'iSI+4mEktTbgc/P5El4qxlzdABeZnKeMYDiteX++N8eASvpiUs9fyHSV4tzho/Q6OF7/r0'
16521: (12)
b'qPxnlQWHhkwV1lSbyFPHXAKFucbzMgjkKYKpaEosDRPkDlgjoz+8+hRDAvsvjIOROpGzxD'
16522: (12)
b'1m2b9KhAmAOvR93YEAj3odEUG/OljQ9XBgnb2IWh7c73hCc6DGk3tUtHqFZnA5Rmn1lSjU'
16523: (12)
b'6oMtoD5o8vymYONSy6ngX1cuAhzcNTD83sT6pI/rIkSqp5HLSFt4h5ZuQTZhszLy/CYXQ6'
16524: (12)
b'N0m/iAFfisTpJ6ehvAf60R6OZ+WVuQPch5VLphyasbnkz8wfUgqiHrKbWSpY/vFS6ZfjsL'
16525: (12)
b'k8mOXaFYnfeXz1q7lFxTC5+N9t/G7BgtBLtzOWgjQkNeQxLJdmgoQF0txgmIPYY7F5pWg7'
16526: (12)
b'aUE2nEyLrPmhpwQpgV3/nWcOUT/U6ipyJrrNBfFEd7eAVmuEqMhqjXCe/EGtO03+kKM0Nb'

```
16527: (12)
b'/3ygCGgDp9l5EcGVmXxK4MjSui46N0DM1f1ea/00lErSPqQVNZFVEzTeW5pjidClRQaTwy'
16528: (12)
b'1os8/gfPlX0H/l/9XGlUETfWq4T1PT/Xzo+Hjtc6KI1xlfyhl0xRhqKLtZPkD2eCNMdn1D'
16529: (12)
b'HA3cBTlRjd8REUMUUGNcWA0X2AbWVfe43woGKNuP5+O4unMT7yZbkBM6S7Gsu6mAo08moZ'
16530: (12)
b'7rCBhWYCjdwaRpyaSqCRW8OQ+mqxOmAj15bj33y1WBOwkWvDifOnFGjk1jLc9f8Wmgg0cm'
16531: (12)
b'sY/p1XCxUCjdyCIZ3qInG10Ru5IKN8Wiis+U5rTWWFpvJUU6H2emTcejx+1Qg8I24ERHmR'
16532: (12)
b'j7E2xiTCU9IzpRoL74G0gronQJpVhPjnPRQs2zTBb7RwF1x6z0YeZwuE4T8T6n59Mq+wto'
16533: (12)
b'K4W2PThSDRQB+8mlGLw2EbQzKQ5XxJ3bP8zbMe8tHUgVQjYNpY+BbkA5op+mBNdQxgLrr1'
16534: (12)
b'6ZorjEtBWaWBKGVVwvVGqILH6Nz/ArTavZuA9NsbRSKbPjnxjdvwRKyOsCsZxt3IDK4dYc'
16535: (12)
b'oQbkVWIJcJp2asYqtETdIcrfcNJ0l8NwdpbaI2A61N1DQdWRkgK9ZmQxBjo1nCVIu/KXjO'
16536: (12)
b'SvSayRj3J7tTQuNOcx8ElYsy0W8spSD9rhamqcdgK4X5bnhLoUVcsVUU2WpHCYPKMZrTzw'
16537: (12)
b'zt92GKJpByJqdAfnaYQ/L5J6PQQd9qCKGwgsJUChIUJsTdPfGBHTtPZRE6mpsALOg6IGZL'
16538: (12)
b'YFVi0n1UKwB5asmgk08IjA4eM2BdbgvSb52x49UH5fL0btWucvxTt3fm3NwxMlVeKDoqXw'
16539: (12)
b'plTrcZiU/b8bBq0Xhcre3IGTNCfz1my8hR27EzZoz8OXYALe0H19qOoYKNfDuOH15rO4oK'
16540: (12)
b'NnJtOXGyqoCNXFtOGGJrO5AGcOTesWSQre1QGsCRe8uKM6sM2Mi14/iBtrbjqWAj15YjQ2'
16541: (12)
b'1tR1TBRq7JsZ2tXezPeIsdoF6pdJUFaBS7VuVlcXWoyRxeOvIFHW9o3gZSXUNfoQfTCyaY'
16542: (12)
b'eB3DoXkSA6cfKT9sOEv7GYyhGw3ou0AKMkbXUJiAzv0Dfbi5LATDfHt3tdiQOny02ODg8b'
16543: (12)
b'JCbuHRTawTi46Pi881HBsNzhxL3DogNpJnf0X0yjxx4fFo1cIJN178gU5g8WjlI18oNA7d'
16544: (12)
b'xRofZ19acLyOkbt8HZs/urQj5cd+ZIVZMiiurJuh2uyZ2bXs0THJmYOPvXfJgVCvjtSMRX'
16545: (12)
b'eEmo46QjTXnlZ0PEvJL23ZXxjE7UVZNv06y1UTZ0C0RjeLOFr0RcQJa57ZMheO223ImjaG'
16546: (12)
b'9Lm1WczSAWVkxbYCKQM/RydfMMs6aqPBAqlx5wzYqBZChYaGHIjmaYgoOj+A0ovOC2g6yn'
16547: (12)
b'NUI4giJwQgnOj48KOVreWCtNewUhL6Cg1y9bVEqaFH9xIxyOsTopOA+u16BekteAXf2kKc'
16548: (12)
b'3mD7rcRbPL2lCL7edoX4Z3/KdoZoQ9bPPKH7N/iOzh8gW6PzB5qO8h+hIRij+yjNLbNonL'
16549: (12)
b'xVTrTnq90l+2Y53InIrw93NskoTycB0TfuBfRWjubJdzP0BkvnZ55wqbLCj1bY6+QkCnvj'
16550: (12)
b'vrXOWBYAN0GnMqSrcvS7iZWzZk5svJbUMOTNaC2pWQDU+nlt6KCfk9Z3dDBqfQmHpiOrHs'
16551: (12)
b'YGfRn/b4cLYnzbdq9rA+3DyX4Kuu+ejZaTuu+wnBIjQfXzeNAOiGBK5Btsnlna22RMHb/f'
16552: (12)
b'8/+dXCmC6h/wS3hmLbfw3gfnaE9ODCmBW7Lv9enM0mHeS2Fp7cRB3oUVRc592hRcuk57qT'
16553: (12)
b'3oPVUO0I485t1YUWRfxIUh9Cw56VkPSD/rKVP3HVVFBK+mQitQ29c1LVNm9lNf3OmgG2Zz'
16554: (12)
b'y8ay/PO6qAhhSpVZQu6Yg5Z1iuZYGcWMpEoN7YcK6DpCRs7grUP13u30SIUm0D0Mdt8sd9'
16555: (12)
b'+jx9nmib+bccL9tFPXqaetckOPmmBmwKs2aN2OGyHK3j9iUdrPNNfEoyKyB0WEebYDxgtE'
16556: (12)
b'Dr5aH3K43j3PkhuPVtBdtBu8JKD6A5RjdK2WpqP+oAVj3z8MO7v41AQyrD4pMFosUrhsmU'
16557: (12)
b'4N9nXoURs5TjgBZosbeDS2oMp2+m7NLEtGpjEspK/mgnU2MH6GTWUHqHF6aZFggFdq4NYZ'
16558: (12)
b'lYl14Ed1F4B6QLO1iB7jlx4KhnYOik3tKg8G+zoH3bKwc6JqQw/nOsp/h2lzOgeJQd3c0W'
16559: (12)
b'JS1wrgjeqcFzGjc5HrHTjnJD7EMgmgnGKZKkyOsdQOdIZ4COzxLHflQ3E7baNVs4qAGoVL'
16560: (12)
b'0vrCtpoAbwSSa/NSh+jnkVaLMoLDnXqrBUvScPSzSPAw0bC+hK9wTyJZtr60D74yDUfRrB'
16561: (12)
```

b'K538I64ikMo6TlltzZFUlef2Fo9kCXvXJvlQmTBVodcEDQBwyww1R+px4RMbHoUQRj2/Yh'
16562: (12)
b'zkx0vduo25xaYNRvlha96jgri497ThaRvtKOgvDYoD0yaL+dmB4x6xLNxH5CVE1pIss00S'
16563: (12)
b'kidI8OGPe6Dr7qdR0ed7EEo6xiH7rlzceSKlbd3pxvmJmvoCJpOihIGjVfwxlwtriGxU/M'
16564: (12)
b'FC/LKzT4cLwh1INFaqCgl1lBlAhzDYSgHCzOGkUHV0StvlCj1vZP5jFRqtT8pCnKwsGmTi'
16565: (12)
b'l6dzmsz91ooYU8PZKhhukJeaPpaCRDTvW7i3o7ZmmB6MCzAfe9tc+hijHKKcY+nK6WdKYW'
16566: (12)
b'Hq3oWHRkPdI6MF7lKZNblh/zJDb6KAwdHyilxt6zz48WZmx4o/tLl8ktcxEmkqc82Ef0f4'
16567: (12)
b'YhyZBqwDTuwnBZBPKWvfqKbD9UGq96WHRAGBQNEA+JpYXCgGiAW8OhEUUPhsZlNBQaRA+E'
16568: (12)
b'BpBhcGYoGQSXjvRDoHEsA6CJTg9/hh0/MbwS6HLkfsDbBuPwHvU7NnefeWcyQuaCyPhYGc'
16569: (12)
b'iNjojL2XBnK/sZ7TQRs4c3K/epFekZ6oq+bhz1K1p4QeTcDT6pVrIwWDwec0d19O4eyi+6'
16570: (12)
b'E5KudKvUdNQqIeWw6zcXI6uxtV6/OQW/9ixjzh7zkCdcdBKTZGQk2l+4GIt+T35WNmlIhX'
16571: (12)
b'UhJNudC80m9lPXPAduzE6w+4yeWVOYPLM2TU6y1IQWbnRSPVlpHPbwwAswpp7a89zs0lF+'
16572: (12)
b'08vcyw394mHL1w4x2M9nzkV4HslzfEjPTzQSXHnKhNsK9bB+6eGJUXtwd6BxVOqpgf6XmS'
16573: (12)
b'P3JjTvFDWGzMKTJvCFp5zs3E70oYXzCddJKZ2bcIHRYLYDzWqjd1RpR3ZJ1rqiB++odo68'
16574: (12)
b'+bHHvZymbF5RQ8zcw5Ueb7Q4HYN1GMolWtKpSHu1yhBarTIAn6TQPTqHbaLxkjPXCYjGj1'
16575: (12)
b'XUE4uO1+0zC8c9e+mCGNkP5haNR4bSgqO+nU1IrwMiGnsqgs+RMyccFd1BhlI0ZziuG2Tp'
16576: (12)
b'ODfaI0RVFmH2Wx38recOCwdz2UmHQ7YcxS4PW6rVNEwjpbsTZHH0pqymo+5kmcSvhxYUht'
16577: (12)
b'q9tURLkbgLLyPh0B4ZrHlKC90IqsRGHQg2ZUsE8zZcXtfRvU6LhLbNUAr04dw5yYdneyQj'
16578: (12)
b'c5Q1VeB7UHJqNyNH2/JaOpjyklbbvhXJ0fvcGbGr17nz5BytCa5IjzTzBUPvmaYoRcvkHC'
16579: (12)
b'0frhQdnUmegHF+7bqdvuf8vOZBZxP0V6qXc34Y5ZRab6C2IzJoxgYM+ilIe1kn5s1nbZUP'
16580: (12)
b'hiyDFfjG6Mu3DdBXnMPqV4mMeNDPW6IqGiBe30eVNOjYQp7F+3D1OGTDPLLw1Wl7eDEXjy'
16581: (12)
b'bnsFiWWyK+q6VKgUZWCZRVnX+CLnCOVsYaQ8sCGmTQBw6mqAjdrccG5nSoLimfkxw941AS'
16582: (12)
b'u3Hp6zzzjPHFAZMFOVcPP1QGDQfcTcC3bjjAAOI5V0E3ZO35cO9ZvSs8U+hI/KlhxbV7Vl'
16583: (12)
b'vwRtRT4VxF3ZJ1fRtChaKJ7sUpFR01CjrcdS9bngvNeGZNSK9TmDh2PSft3WbQd7BNPOOP'
16584: (12)
b'jksHgcGkK4XTkLeUY8MQRXdpKFEtKUpY2aFTqpZ8KO1sXx1lhp3DhXOKDBfOGTBcOGfIk6'
16585: (12)
b'6GDZpi97UPM+pZY4Fo6kUwOuJQkPa9oiF0t+iA0C8aIPQ7+cTQI/uXBUEuNT1jpBndwViP'
16586: (12)
b'eNFFjJVm+tX+KLSrKxlRH3QvkzWGHlXTuQGv2ox1O66+jA99Qfdnfzqb+zdyCzzyMGLGd+'
16587: (12)
b'VA2ieCavtpTnqk9ntkxE/U7KxfzWZnwhlNaIUxnr42yXiX3uSNgUYzU+P0GM+WFoLJPGgS'
16588: (12)
b'IKmtTB60SqOvhLs2UybEHQ9Z8vPFnCYRdkaMVmOTVZtYb+r8SOUgASYWGMKBktoi6ogJS9'
16589: (12)
b'Ye2tF302eCnsx7cpzrhens4gY3TDENGyXDeXhuP4NXB6i5+MwiIQczDdyaj7vw/YzcBaAW'
16590: (12)
b'r50DPUufeSjM0x0Uz9RzD4a5uoNudUhOVD1fd66jGbvDbh0SLy1LT+eda+nnnJMwpZ8L4C'
16591: (12)
b'f1zotb7TNHUdoY4t2aJ7NB7RjSU7o06MPkLjg/Tyeprr9E1Y3u5kKdje7m0nQ0dhgGmtFV'
16592: (12)
b'I514xqiNenzcRLNkPDmoHDJqoHQoz7yFR7Wcoj+xkLNdyR01RORmuNzvnJPSeeARERajXV'
16593: (12)
b'azUDSDmFrQz+Yciozv9506PEShedIxDBulQ+LBxKAv0YtmlERd/eBOlFDm6FrxCsqtNmAp'
16594: (12)
b'QUerJJBUvwfNNhFdVYX+IrqqStNR2TIgxIPs//NMc9qnrbUca4uIIXdGs0FaXLktPRac1R'
16595: (12)
b'7a9xsHVQZ67M29Ms3SUGbZjxNVEnw8GB2o8WrutbDShd01hkAzRn+/8ATZwmlgj45m22GC'

```
16596: (12)
b'fUSf0Jkb5GiePf0uV7YCl991ok8Uz266sqZMOR+I/i5bImq/70bHhC4CqrWMGwjZHWv3o0'
16597: (12)
b'uTnGWRB6mn/ZA1803ZqXnSW+zOFeRNdhGC3Efo18SR5cd+/bRBsHziwRC7R16aPrXEkTtA'
16598: (12)
b'zdwSPMRPa1jagPLZWr4013NO5D7DRCoCwlTKwWEyRSCaNBjAGHZSceNnmmlCc7J7RYRVdA'
16599: (12)
b'eMN1gcfLXB4vB4g4XgNrrIDrmnVzPQcvUEe7Yi7W/BMIS+lccB4coOAvoE9czQ8RyQ88vr'
16600: (12)
b'KU3DJn41u2jYEcQa7MQAXoW1lNZhPRKUWCLeOKtG5NHNYKgP0c1gmo46FlSPy/g2D47Sl/'
16601: (12)
b'F1HosrMDoZjSx67XZflZ7ROEQGWu8kaGm5Q2SwNH4O57ewNZw7RDSGIp9OHSYaYOUBCZkB'
16602: (12)
b'8WauPONH0D8MqbSjmnSQOQ3kLc3IhOr1IuN1dLNO4bDvIboPmZCjdajaAkGDMkCsP2UWCt'
16603: (12)
b'qTAW7pTiYpWnMyLiO9ySC3tCYjtNaZjEspSMMO+tLMkV5bMo6lSI0c8m5OY7JQK0PGtVeF'
16604: (12)
b'HNEfN0bRnCa8RhnxXeR2tXlyMes5GaK9KLM/Uuqylxqkuxqt XCYXubwMIYaFFUeEy8saDc'
16605: (12)
b'hKS5VEz4HmyWWzDt1HkYIOt41VlpSzIZDd2yFCRH3b2CKQ3jMmxIJJ9HnAJBlzhQXRVmmA'
16606: (12)
b'nQDpUkUjdxItS4DqpjAIKTeUQUptJmnI8C4xSH3tD8LR14lBd7i4C8qaif30V860M0uraC'
16607: (12)
b'muvqCsbSwdhbi0mFxQtgIdX1DGHNeQzhDk3ZUdMmTUtxSVye3lYXjVt1Ogz7+EO8yQqZKZ'
16608: (12)
b'6Ogu148YrzyoluQq43J08xOkj1RGlAVX4PytQcVK0eYS7QlTIJD2m2u3uqvJFe4vJ6Jb9x'
16609: (12)
b'TxnJ/s7cyy9QQlJxdaMRt8u2eRvsgLPCTQiqMtbzQonsg2158tCk/ox4ebMeh1SBO44fgL'
16610: (12)
b'HzAPc4jcn4bK8DI2xPeYO0kBEaL8ZQKsdT0v37+Mn8qGwnc1/E2L5Gr0m4+xaPBD3UAPtz'
16611: (12)
b'ZW8GrldBXgq1czG5S7f5KY/qP7rCoPSCeA6HVvh6yRboXfusVaOjRZ0le1LgN4y+45wr3F'
16612: (12)
b'cwRqW2cwbgWSJtdhaEwHkSZf2cWXyVfZSyvwrbfSLB0MlEjrW4or0NwsWJIRtgdyRZbFCA'
16613: (12)
b'hLkgYMS5KWNKe4oAE3QgWt2GDaz2pC5G0IL7uhZ/sahhkEqXo9qEHRS88YW78q3XI+JTlS'
16614: (12)
b'LRtiV5rlguhYsVwC1JkzA23ejeDuiu8TzAg6qRYCcBKrngabLCOOPo8yizjhjaI4LAfWAK'
16615: (12)
b'Pbb9vkq5/LIE16WWMFt2iC+uEkNHcL+TrkaV1/iJ3WR31XPObpDvNNRADdTgBGHS+qoJ6r'
16616: (12)
b'VxDImJjefGe8HTN1UjxTG602yf9isEoPOoB58lU6XVQlP/hVSGxQ+ZHjeiyeoeLogW01TV'
16617: (12)
b'5ZyFXy6rsVJPl1re4snYHUhzdWoPXhDU1H8i7IkGBqUOM+tG49qAMkeFZ2uAWF+2ou1uME'
16618: (12)
b'ncF+fbs9hCE169ewU8g4R89ImtBfw0uUYTV9GjNib3WZvKpnhpbJa2i5pSXETB3d8Ksaz2'
16619: (12)
b'uSaosN85BX1dKhO73q3axZChq+OSbwFuo0RSqixkoHIV+Rnk7dmwrJvKZUwyFNFvTFkAaQ'
16620: (12)
b'Rwox0CrAzWWAL2cOh07VHeOFmEn7HZ4qB2i/1278Cstk9T2mDmFqHaHb2huT/GJRRYi7NJ'
16621: (12)
b'zn4LjlZSqRclw7x8PrwV+kY5yEk3g8kn7lRrOXls2kfS+IRX7tRrNTz+b94ryja7SmVX6H'
16622: (12)
b'L4tRLs2G/m46Zjccab4LxPjzb+PxRl2H9jTYCAZcFhVnLgmnMw0Yy4mTWG0/lr48/7fFu/'
16623: (12)
b'r7TiStLhnQF7+X0GLsQjNRFHpBfDYBrVuNoaWZQOaoW0ce6SXXWQZa+9Z0pNQhQwbzMMmM'
16624: (12)
b'H5HdC1noSf1GUIY4pL9GeEbfTLmF/KrPysFV6L1RB98OZqK0Sjj3xHDzpxqB82Xypza3zp'
16625: (12)
b'JgT4lZ1p+6F4LTqBdqkj+jEx3QCf7kBUpNm0SWjui4xawRmfynkrXNEz4EBD30bb3ehA57'
16626: (12)
b'2ib6tnRouG8yM18mcnF6Rlz1ZFkSXaNuvOmlLNJ68JiC1uOGpqOByDAkmhTUfs3h1e+6Ut'
16627: (12)
b'yroSn3oI7iCozqwgJcrdqXcB7Ko7ZEGCaq5E3P9JG8qIAsLdPgInlTCuB0TtLcCB+GsGUW'
16628: (12)
b'wFg3ZF6Od4pXxvWtkbCMGaORcB5zxzvNqFgRf7TlDIXk7Xp7GlPwt6vdaegmb7eNKzD+vn'
16629: (12)
b'3HuALV9e2WccXMBGa3LIezXTcJGYc6oSoi029MU5nncZsmokZbQ16dDq8ZwHG9RRN4Q9sM'
16630: (12)
```

```
b'JhbzCI8fxjI8fXHZlBl5vLmCgwYHKDYETAUbH7VnVXasGGcFOPdhijKDDF55YIm4bYpmaj'
16631: (12)
b'/9agumUm+91oGRC1rwgvxgdIhY+sMb+mmMFWzD8eYYhYi6G6RtMA9mm48wT1NkmJYZMEzL'
16632: (12)
b'DBlNsTKH6PsyVk0KMaID4ag0QxC5Zji62deKjnqWkgypDSiwqzuvoe29XV163V6BUT+C/s'
16633: (12)
b'g8VmLPJ6AgBt1PGmFVh2ZieJNttIxJfgtv72KWJkvgLMmX4alDIe9ZAryXaR5D+oJRlCtt'
16634: (12)
b'4uZIpR+skDN6sIIoftrBShkGLiQhOvGNIC4qg9EJRAfAS0VHGVyQIVVpAup03z/pPrZxWD'
16635: (12)
b'+c+8c+ejQDQxp4u/4MPUTDVYBv+ZqRPS7GwoNa7CswKkbGrroVdowX3XuwJ9Xj5HJF2i8Y'
16636: (12)
b'r5JvHFvnyTd9WA36xjdZRCbPO2/wrS8cIK2MOmuSI6NOBnVt1FkZNBh1Gldjo04G16szXJ'
16637: (12)
b'mhR0e4JgC1jSdD+qN7xIRbHVhFCRs0visQvfW39fEPtSnPGN/M2adlaT9D1xABoXNwcOge'
16638: (12)
b'AGhtCSn1S+VVi28ZqWeWcCM1an0KwBp+8tO+sV4tzJcYVjraj9ezPPkWLeAgtpuWk2hS37'
16639: (12)
b'pbJ6NRAaITtgg/OmFL+mh2rybmK2z/WFrtX5UG8FtSltJ7Sh4Jm0oWiXeVbLB6s8gi0W6R'
16640: (12)
b'hfSukEXUzo8F9HkXi/jtHUuZZvT7wLfOqAusAngYDg7PJpNFwK0MwFD3ndEakhGdR0ShbD'
16641: (12)
b'vdnOYEzKK/vko+I6oLj+HcLr3KcG4U3zL5Fh0rQwWOjpWRPgzqPnBUQW0lwoYRDYwQNToR'
16642: (12)                    b'A/fRiRjQ0s/D79gsABOib2GDDQmK7OEReGQPP0/+7a59v0z+H+SUGTTsMAEA'
16643: (12)                    )).decode().splitlines()
16644: (0)          def ConversionHeader(i: str, filename: OptStr ="unknown"):
16645: (4)              t = i.lower()
16646: (4)              import textwrap
16647: (4)              html = textwrap.dedent("""
16648: (12)                    <!DOCTYPE html>
16649: (12)                    <html>
16650: (12)                    <head>
16651: (12)                    <style>
16652: (12)                    body{background-color:gray}
16653: (12)                    div{position:relative;background-color:white;margin:1em auto}
16654: (12)                    p{position:absolute;margin:0}
16655: (12)                    img{position:absolute}
16656: (12)                    </style>
16657: (12)                    </head>
16658: (12)                    <body>
16659: (12)                    """)
16660: (4)              xml = textwrap.dedent("""
16661: (12)                    <?xml version="1.0"?>
16662: (12)                    <document name="%s">
16663: (12)                    """
16664: (12)                    % filename
16665: (12)                    )
16666: (4)              xhtml = textwrap.dedent("""
16667: (12)                    <?xml version="1.0"?>
16668: (12)                    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
16669: (12)                    <html xmlns="http://www.w3.org/1999/xhtml">
16670: (12)                    <head>
16671: (12)                    <style>
16672: (12)                    body{background-color:gray}
16673: (12)                    div{background-color:white;margin:1em;padding:1em}
16674: (12)                    p{white-space:pre-wrap}
16675: (12)                    </style>
16676: (12)                    </head>
16677: (12)                    <body>
16678: (12)                    """)
16679: (4)              text = ""
16680: (4)              json = '{"document": "%s", "pages": [\n' % filename
16681: (4)              if t == "html":
16682: (8)                  r = html
16683: (4)              elif t == "json":
16684: (8)                  r = json
16685: (4)              elif t == "xml":
16686: (8)                  r = xml
```

```
16687: (4)             elif t == "xhtml":
16688: (8)                 r = xhtml
16689: (4)             else:
16690: (8)                 r = text
16691: (4)             return r
16692: (0)         def ConversionTrailer(i: str):
16693: (4)             t = i.lower()
16694: (4)             text = ""
16695: (4)             json = "]\n}"
16696: (4)             html = "</body>\n</html>\n"
16697: (4)             xml = "</document>\n"
16698: (4)             xhtml = html
16699: (4)             if t == "html":
16700: (8)                 r = html
16701: (4)             elif t == "json":
16702: (8)                 r = json
16703: (4)             elif t == "xml":
16704: (8)                 r = xml
16705: (4)             elif t == "xhtml":
16706: (8)                 r = xhtml
16707: (4)             else:
16708: (8)                 r = text
16709: (4)             return r
16710: (0)         def adobe_glyph_names() -> tuple:
16711: (4)             '''
16712: (4)             Adobe Glyph List function
16713: (4)             '''
16714: (4)             if _adobe_unicodes == {}:
16715: (8)                 for line in _get_glyph_text():
16716: (12)                    if line.startswith("#"):
16717: (16)                        continue
16718: (12)                    gname, unc = line.split(";")
16719: (12)                    c = int("0x" + unc[:4], base=16)
16720: (12)                    _adobe_unicodes[gname] = c
16721: (4)             return tuple(_adobe_unicodes.keys())
16722: (0)         def adobe_glyph_unicodes() -> tuple:
16723: (4)             '''
16724: (4)             Adobe Glyph List function
16725: (4)             '''
16726: (4)             if _adobe_unicodes == {}:
16727: (8)                 for line in _get_glyph_text():
16728: (12)                    if line.startswith("#"):
16729: (16)                        continue
16730: (12)                    gname, unc = line.split(";")
16731: (12)                    c = int("0x" + unc[:4], base=16)
16732: (12)                    _adobe_unicodes[gname] = c
16733: (4)             return tuple(_adobe_unicodes.values())
16734: (0)         def annot_preprocess(page: "Page") -> int:
16735: (4)             """Prepare for annotation insertion on the page.
16736: (4)             Returns:
16737: (8)                 Old page rotation value. Temporarily sets rotation to 0 when required.
16738: (4)             """
16739: (4)             CheckParent(page)
16740: (4)             if not page.parent.is_pdf:
16741: (8)                 raise ValueError("is no PDF")
16742: (4)             old_rotation = page.rotation
16743: (4)             if old_rotation != 0:
16744: (8)                 page.set_rotation(0)
16745: (4)             return old_rotation
16746: (0)         def annot_postprocess(page: "Page", annot: "Annot") -> None:
16747: (4)             """Clean up after annotation insertion.
16748: (4)             Set ownership flag and store annotation in page annotation dictionary.
16749: (4)             """
16750: (4)             assert isinstance( page, Page)
16751: (4)             assert isinstance( annot, Annot)
16752: (4)             annot.parent = page
16753: (4)             page._annot_refs[id(annot)] = annot
16754: (4)             annot.thisown = True
16755: (0)         def canon(c):
```

```
16756: (4)              assert isinstance(c, int)
16757: (4)              if c == 0xA0 or c == 0x2028 or c == 0x2029:
16758: (8)                  return ord(' ')
16759: (4)              if c == ord('\r') or c == ord('\n') or c == ord('\t'):
16760: (8)                  return ord(' ')
16761: (4)              if c >= ord('A') and c <= ord('Z'):
16762: (8)                  return c - ord('A') + ord('a')
16763: (4)              return c
16764: (0)          def chartocanon(s):
16765: (4)              assert isinstance(s, str)
16766: (4)              n, c = mupdf.fz_chartorune(s)
16767: (4)              c = canon(c)
16768: (4)              return n, c
16769: (0)          def dest_is_valid(o, page_count, page_object_nums, names_list):
16770: (4)              p = mupdf.pdf_dict_get( o, PDF_NAME('A'))
16771: (4)              if (
16772: (12)                     mupdf.pdf_name_eq(
16773: (16)                         mupdf.pdf_dict_get( p, PDF_NAME('S')),
16774: (16)                         PDF_NAME('GoTo')
16775: (16)                         )
16776: (12)                     and not string_in_names_list(
16777: (16)                         mupdf.pdf_dict_get( p, PDF_NAME('D')),
16778: (16)                         names_list
16779: (16)                         )
16780: (12)                     ):
16781: (8)                  return 0
16782: (4)              p = mupdf.pdf_dict_get( o, PDF_NAME('Dest'))
16783: (4)              if not p.m_internal:
16784: (8)                  pass
16785: (4)              elif mupdf.pdf_is_string( p):
16786: (8)                  return string_in_names_list( p, names_list)
16787: (4)              elif not dest_is_valid_page(
16788: (12)                     mupdf.pdf_array_get( p, 0),
16789: (12)                     page_object_nums,
16790: (12)                     page_count,
16791: (12)                     ):
16792: (8)                  return 0
16793: (4)              return 1
16794: (0)          def dest_is_valid_page(obj, page_object_nums, pagecount):
16795: (4)              num = mupdf.pdf_to_num(obj)
16796: (4)              if num == 0:
16797: (8)                  return 0
16798: (4)              for i in range(pagecount):
16799: (8)                  if page_object_nums[i] == num:
16800: (12)                     return 1
16801: (4)              return 0
16802: (0)          def find_string(s, needle):
16803: (4)              assert isinstance(s, str)
16804: (4)              for i in range(len(s)):
16805: (8)                  end = match_string(s[i:], needle)
16806: (8)                  if end is not None:
16807: (12)                     end += i
16808: (12)                     return i, end
16809: (4)              return None, None
16810: (0)          def get_pdf_now() -> str:
16811: (4)              '''
16812: (4)              "Now" timestamp in PDF Format
16813: (4)              '''
16814: (4)              import time
16815: (4)              tz = "%s'%s'" % (
16816: (8)                  str(abs(time.altzone // 3600)).rjust(2, "0"),
16817: (8)                  str((abs(time.altzone // 60) % 60)).rjust(2, "0"),
16818: (4)              )
16819: (4)              tstamp = time.strftime("D:%Y%m%d%H%M%S", time.localtime())
16820: (4)              if time.altzone > 0:
16821: (8)                  tstamp += "-" + tz
16822: (4)              elif time.altzone < 0:
16823: (8)                  tstamp += "+" + tz
16824: (4)              else:
```

```
16825: (8)                  pass
16826: (4)              return tstamp
16827: (0)          class ElementPosition(object):
16828: (4)              """Convert a dictionary with element position information to an object."""
16829: (4)              def __init__(self):
16830: (8)                  pass
16831: (0)          def make_story_elpos():
16832: (4)              return ElementPosition()
16833: (0)          def get_highlight_selection(page, start: point_like =None, stop: point_like
=None, clip: rect_like =None) -> list:
16834: (4)              """Return rectangles of text lines between two points.
16835: (4)              Notes:
16836: (8)                  The default of 'start' is top-left of 'clip'. The default of 'stop'
16837: (8)                  is bottom-reight of 'clip'.
16838: (4)              Args:
16839: (8)                  start: start point_like
16840: (8)                  stop: end point_like, must be 'below' start
16841: (8)                  clip: consider this rect_like only, default is page rectangle
16842: (4)              Returns:
16843: (8)                  List of line bbox intersections with the area established by the
16844: (8)                  parameters.
16845: (4)              """
16846: (4)              if clip is None:
16847: (8)                  clip = page.rect
16848: (4)              clip = Rect(clip)
16849: (4)              if start is None:
16850: (8)                  start = clip.tl
16851: (4)              if stop is None:
16852: (8)                  stop = clip.br
16853: (4)              clip.y0 = start.y
16854: (4)              clip.y1 = stop.y
16855: (4)              if clip.is_empty or clip.is_infinite:
16856: (8)                  return []
16857: (4)              blocks = page.get_text(
16858: (8)                  "dict", flags=0, clip=clip,
16859: (4)              )["blocks"]
16860: (4)              lines = []  # will return this list of rectangles
16861: (4)              for b in blocks:
16862: (8)                  bbox = Rect(b["bbox"])
16863: (8)                  if bbox.is_infinite or bbox.is_empty:
16864: (12)                     continue
16865: (8)                  for line in b["lines"]:
16866: (12)                     bbox = Rect(line["bbox"])
16867: (12)                     if bbox.is_infinite or bbox.is_empty:
16868: (16)                         continue
16869: (12)                     lines.append(bbox)
16870: (4)              if lines == []:  # did not select anything
16871: (8)                  return lines
16872: (4)              lines.sort(key=lambda bbox: bbox.y1)  # sort by vertical positions
16873: (4)              bboxf = lines.pop(0)
16874: (4)              if bboxf.y0 - start.y <= 0.1 * bboxf.height:  # close enough?
16875: (8)                  r = Rect(start.x, bboxf.y0, bboxf.br)  # intersection rectangle
16876: (8)                  if not (r.is_empty or r.is_infinite):
16877: (12)                     lines.insert(0, r)  # insert again if not empty
16878: (4)              else:
16879: (8)                  lines.insert(0, bboxf)  # insert again
16880: (4)              if lines == []:  # the list might have been emptied
16881: (8)                  return lines
16882: (4)              bboxl = lines.pop()
16883: (4)              if stop.y - bboxl.y1 <= 0.1 * bboxl.height:  # close enough?
16884: (8)                  r = Rect(bboxl.tl, stop.x, bboxl.y1)  # intersection rectangle
16885: (8)                  if not (r.is_empty or r.is_infinite):
16886: (12)                     lines.append(r)  # append if not empty
16887: (4)              else:
16888: (8)                  lines.append(bboxl)  # append again
16889: (4)              return lines
16890: (0)          def glyph_name_to_unicode(name: str) -> int:
16891: (4)              '''
16892: (4)              Adobe Glyph List function
```

```
16893: (4)              '''
16894: (4)              if _adobe_unicodes == {}:
16895: (8)                  for line in _get_glyph_text():
16896: (12)                     if line.startswith("#"):
16897: (16)                         continue
16898: (12)                     gname, unc = line.split(";")
16899: (12)                     c = int(unc[:4], base=16)
16900: (12)                     _adobe_unicodes[gname] = c
16901: (4)              return _adobe_unicodes.get(name, 65533)
16902: (0)          def hdist(dir, a, b):
16903: (4)              dx = b.x - a.x
16904: (4)              dy = b.y - a.y
16905: (4)              return mupdf.fz_abs(dx * dir.x + dy * dir.y)
16906: (0)          def make_table(rect: rect_like =(0, 0, 1, 1), cols: int =1, rows: int =1) ->
list:
16907: (4)              """Return a list of (rows x cols) equal sized rectangles.
16908: (4)              Notes:
16909: (8)                  A utility to fill a given area with table cells of equal size.
16910: (4)              Args:
16911: (8)                  rect: rect_like to use as the table area
16912: (8)                  rows: number of rows
16913: (8)                  cols: number of columns
16914: (4)              Returns:
16915: (8)                  A list with <rows> items, where each item is a list of <cols>
16916: (8)                  PyMuPDF Rect objects of equal sizes.
16917: (4)              """
16918: (4)              rect = Rect(rect)  # ensure this is a Rect
16919: (4)              if rect.is_empty or rect.is_infinite:
16920: (8)                  raise ValueError("rect must be finite and not empty")
16921: (4)              tl = rect.tl
16922: (4)              height = rect.height / rows  # height of one table cell
16923: (4)              width = rect.width / cols  # width of one table cell
16924: (4)              delta_h = (width, 0, width, 0)  # diff to next right rect
16925: (4)              delta_v = (0, height, 0, height)  # diff to next lower rect
16926: (4)              r = Rect(tl, tl.x + width, tl.y + height)  # first rectangle
16927: (4)              row = [r]
16928: (4)              for i in range(1, cols):
16929: (8)                  r += delta_h  # build next rect to the right
16930: (8)                  row.append(r)
16931: (4)              rects = [row]
16932: (4)              for i in range(1, rows):
16933: (8)                  row = rects[i - 1]  # take previously appended row
16934: (8)                  nrow = []  # the new row to append
16935: (8)                  for r in row:  # for each previous cell add its downward copy
16936: (12)                     nrow.append(r + delta_v)
16937: (8)                  rects.append(nrow)  # append new row to result
16938: (4)              return rects
16939: (0)          def util_ensure_widget_calc(annot):
16940: (4)              '''
16941: (4)              Ensure that widgets with /AA/C JavaScript are in array AcroForm/CO
16942: (4)              '''
16943: (4)              annot_obj = mupdf.pdf_annot_obj(annot.this)
16944: (4)              pdf = mupdf.pdf_get_bound_document(annot_obj)
16945: (4)              PDFNAME_CO = mupdf.pdf_new_name("CO")    # = PDF_NAME(CO)
16946: (4)              acro = mupdf.pdf_dict_getl(  # get AcroForm dict
16947: (12)                     mupdf.pdf_trailer(pdf),
16948: (12)                     PDF_NAME('Root'),
16949: (12)                     PDF_NAME('AcroForm'),
16950: (12)                     )
16951: (4)              CO = mupdf.pdf_dict_get(acro, PDFNAME_CO)  # = AcroForm/CO
16952: (4)              if not mupdf.pdf_is_array(CO):
16953: (8)                  CO = mupdf.pdf_dict_put_array(acro, PDFNAME_CO, 2)
16954: (4)              n = mupdf.pdf_array_len(CO)
16955: (4)              found = 0
16956: (4)              xref = mupdf.pdf_to_num(annot_obj)
16957: (4)              for i in range(n):
16958: (8)                  nxref = mupdf.pdf_to_num(mupdf.pdf_array_get(CO, i))
16959: (8)                  if xref == nxref:
16960: (12)                     found = 1
```

```
16961: (12)                     break
16962: (4)              if not found:
16963: (8)                  mupdf.pdf_array_push(CO, mupdf.pdf_new_indirect(pdf, xref, 0))
16964: (0)          def util_make_rect( *args, p0=None, p1=None, x0=None, y0=None, x1=None,
y1=None):
16965: (4)              '''
16966: (4)              Helper for initialising rectangle classes.
16967: (4)              2022-09-02: This is quite different from PyMuPDF's util_make_rect(), which
16968: (4)              uses `goto` in ways that don't easily translate to Python.
16969: (4)              Returns (x0, y0, x1, y1) derived from <args>, then override with p0, p1,
16970: (4)              x0, y0, x1, y1 if they are not None.
16971: (4)              Accepts following forms for <args>:
16972: (8)                  () returns all zeros.
16973: (8)                  (top-left, bottom-right)
16974: (8)                  (top-left, x1, y1)
16975: (8)                  (x0, y0, bottom-right)
16976: (8)                  (x0, y0, x1, y1)
16977: (8)                  (rect)
16978: (4)              Where top-left and bottom-right are (x, y) or something with .x, .y
16979: (4)              members; rect is something with .x0, .y0, .x1, and .y1 members.
16980: (4)              2023-11-18: we now override with p0, p1, x0, y0, x1, y1 if not None.
16981: (4)              '''
16982: (4)              def get_xy( arg):
16983: (8)                  if isinstance( arg, (list, tuple)) and len( arg) == 2:
16984: (12)                     return arg[0], arg[1]
16985: (8)                  if isinstance( arg, (Point, mupdf.FzPoint, mupdf.fz_point)):
16986: (12)                     return arg.x, arg.y
16987: (8)                  return None, None
16988: (4)              def make_tuple( a):
16989: (8)                  if isinstance( a, tuple):
16990: (12)                     return a
16991: (8)                  if isinstance( a, Point):
16992: (12)                     return a.x, a.y
16993: (8)                  elif isinstance( a, (Rect, IRect, mupdf.FzRect, mupdf.fz_rect)):
16994: (12)                     return a.x0, a.y0, a.x1, a.y1
16995: (8)                  if not isinstance( a, (list, tuple)):
16996: (12)                     a = a,
16997: (8)                  return a
16998: (4)              def handle_args():
16999: (8)                  if len(args) == 0:
17000: (12)                     return 0, 0, 0, 0
17001: (8)                  elif len(args) == 1:
17002: (12)                     arg = args[0]
17003: (12)                     if isinstance( arg, (list, tuple)) and len( arg) == 2:
17004: (16)                         p1, p2 = arg
17005: (16)                         return *p1, *p2
17006: (12)                     if isinstance( arg, (list, tuple)) and len( arg) == 3:
17007: (16)                         a, b, c = arg
17008: (16)                         a = make_tuple(a)
17009: (16)                         b = make_tuple(b)
17010: (16)                         c = make_tuple(c)
17011: (16)                         ret = *a, *b, *c
17012: (16)                         return ret
17013: (12)                     arg = make_tuple( arg)
17014: (12)                     return arg
17015: (8)                  elif len(args) == 2:
17016: (12)                     return get_xy( args[0]) + get_xy( args[1])
17017: (8)                  elif len(args) == 3:
17018: (12)                     x0, y0 = get_xy( args[0])
17019: (12)                     if (x0, y0) != (None, None):
17020: (16)                         return x0, y0, args[1], args[2]
17021: (12)                     x1, y1 = get_xy( args[2])
17022: (12)                     if (x1, y1) != (None, None):
17023: (16)                         return args[0], args[1], x1, y1
17024: (8)                  elif len(args) == 4:
17025: (12)                     return args[0], args[1], args[2], args[3]
17026: (8)                  raise Exception( f'Unrecognised args: {args}')
17027: (4)              ret_x0, ret_y0, ret_x1, ret_y1 = handle_args()
17028: (4)              if p0 is not None:  ret_x0, ret_y0 = get_xy(p0)
```

```
17029: (4)              if p1 is not None:  ret_x1, ret_y1 = get_xy(p1)
17030: (4)              if x0 is not None:  ret_x0 = x0
17031: (4)              if y0 is not None:  ret_y0 = y0
17032: (4)              if x1 is not None:  ret_x1 = x1
17033: (4)              if y1 is not None:  ret_y1 = y1
17034: (4)              return ret_x0, ret_y0, ret_x1, ret_y1
17035: (0)          def util_make_irect( *args, p0=None, p1=None, x0=None, y0=None, x1=None,
y1=None):
17036: (4)              a, b, c, d = util_make_rect( *args, p0=p0, p1=p1, x0=x0, y0=y0, x1=x1,
y1=y1)
17037: (4)              def convert(x):
17038: (8)                  ret = int(x)
17039: (8)                  return ret
17040: (4)              a = convert(a)
17041: (4)              b = convert(b)
17042: (4)              c = convert(c)
17043: (4)              d = convert(d)
17044: (4)              return a, b, c, d
17045: (0)          def util_round_rect( rect):
17046: (4)              return JM_py_from_irect(mupdf.fz_round_rect(JM_rect_from_py(rect)))
17047: (0)          def util_transform_rect( rect, matrix):
17048: (4)              if g_use_extra:
17049: (8)                  return extra.util_transform_rect( rect, matrix)
17050: (4)              return JM_py_from_rect(mupdf.fz_transform_rect(JM_rect_from_py(rect),
JM_matrix_from_py(matrix)))
17051: (0)          def util_intersect_rect( r1, r2):
17052: (4)              return JM_py_from_rect(
17053: (12)                     mupdf.fz_intersect_rect(
17054: (16)                         JM_rect_from_py(r1),
17055: (16)                         JM_rect_from_py(r2),
17056: (16)                         )
17057: (12)                     )
17058: (0)          def util_is_point_in_rect( p, r):
17059: (4)              return mupdf.fz_is_point_inside_rect(
17060: (16)                         JM_point_from_py(p),
17061: (16)                         JM_rect_from_py(r),
17062: (16)                         )
17063: (0)          def util_include_point_in_rect( r, p):
17064: (4)              return JM_py_from_rect(
17065: (12)                     mupdf.fz_include_point_in_rect(
17066: (16)                         JM_rect_from_py(r),
17067: (16)                         JM_point_from_py(p),
17068: (16)                         )
17069: (12)                     )
17070: (0)          def util_point_in_quad( P, Q):
17071: (4)              p = JM_point_from_py(P)
17072: (4)              q = JM_quad_from_py(Q)
17073: (4)              return mupdf.fz_is_point_inside_quad(p, q)
17074: (0)          def util_transform_point( point, matrix):
17075: (4)              return JM_py_from_point(
17076: (12)                     mupdf.fz_transform_point(
17077: (16)                         JM_point_from_py(point),
17078: (16)                         JM_matrix_from_py(matrix),
17079: (16)                         )
17080: (12)                     )
17081: (0)          def util_union_rect( r1, r2):
17082: (4)              return JM_py_from_rect(
17083: (12)                     mupdf.fz_union_rect(
17084: (16)                         JM_rect_from_py(r1),
17085: (16)                         JM_rect_from_py(r2),
17086: (16)                         )
17087: (12)                     )
17088: (0)          def util_concat_matrix( m1, m2):
17089: (4)              return JM_py_from_matrix(
17090: (12)                     mupdf.fz_concat(
17091: (16)                         JM_matrix_from_py(m1),
17092: (16)                         JM_matrix_from_py(m2),
17093: (16)                         )
17094: (12)                     )
```

```
(0)          def util_invert_matrix(matrix):
(4)              if 0:
(8)                  if isinstance( matrix, (tuple, list)):
(12)                     matrix = mupdf.FzMatrix( *matrix)
(8)                  elif isinstance( matrix, mupdf.fz_matrix):
(12)                     matrix = mupdf.FzMatrix( matrix)
(8)                  elif isinstance( matrix, Matrix):
(12)                     matrix = mupdf.FzMatrix( matrix.a, matrix.b, matrix.c, matrix.d,
matrix.e, matrix.f)
(8)                  assert isinstance( matrix, mupdf.FzMatrix), f'{type(matrix)=}:
{matrix}'
(8)                  ret = mupdf.fz_invert_matrix( matrix)
(8)                  if ret == matrix and (0
(16)                         or abs( matrix.a - 1) >= sys.float_info.epsilon
(16)                         or abs( matrix.b - 0) >= sys.float_info.epsilon
(16)                         or abs( matrix.c - 0) >= sys.float_info.epsilon
(16)                         or abs( matrix.d - 1) >= sys.float_info.epsilon
(16)                         ):
(12)                     return 1, ()
(8)                  return 0, (ret.a, ret.b, ret.c, ret.d, ret.e, ret.f)
(4)              src = JM_matrix_from_py(matrix)
(4)              a = src.a
(4)              det = a * src.d - src.b * src.c
(4)              if det < -sys.float_info.epsilon or det > sys.float_info.epsilon:
(8)                  dst = mupdf.FzMatrix()
(8)                  rdet = 1 / det
(8)                  dst.a = src.d * rdet
(8)                  dst.b = -src.b * rdet
(8)                  dst.c = -src.c * rdet
(8)                  dst.d = a * rdet
(8)                  a = -src.e * dst.a - src.f * dst.c
(8)                  dst.f = -src.e * dst.b - src.f * dst.d
(8)                  dst.e = a
(8)                  return 0, (dst.a, dst.b, dst.c, dst.d, dst.e, dst.f)
(4)              return 1, ()
(0)          def util_measure_string( text, fontname, fontsize, encoding):
(4)              font = mupdf.fz_new_base14_font(fontname)
(4)              w = 0
(4)              pos = 0
(4)              while pos < len(text):
(8)                  t, c = mupdf.fz_chartorune(text[pos:])
(8)                  pos += t
(8)                  if encoding == mupdf.PDF_SIMPLE_ENCODING_GREEK:
(12)                     c = mupdf.fz_iso8859_7_from_unicode(c)
(8)                  elif encoding == mupdf.PDF_SIMPLE_ENCODING_CYRILLIC:
(12)                     c = mupdf.fz_windows_1251_from_unicode(c)
(8)                  else:
(12)                     c = mupdf.fz_windows_1252_from_unicode(c)
(8)                  if c < 0:
(12)                     c = 0xB7
(8)                  g = mupdf.fz_encode_character(font, c)
(8)                  dw = mupdf.fz_advance_glyph(font, g, 0)
(8)                  w += dw
(4)              ret = w * fontsize
(4)              return ret
(0)          def util_sine_between(C, P, Q):
(4)              c = JM_point_from_py(C)
(4)              p = JM_point_from_py(P)
(4)              q = JM_point_from_py(Q)
(4)              s = mupdf.fz_normalize_vector(mupdf.fz_make_point(q.x - p.x, q.y - p.y))
(4)              m1 = mupdf.fz_make_matrix(1, 0, 0, 1, -p.x, -p.y)
(4)              m2 = mupdf.fz_make_matrix(s.x, -s.y, s.y, s.x, 0, 0)
(4)              m1 = mupdf.fz_concat(m1, m2)
(4)              c = mupdf.fz_transform_point(c, m1)
(4)              c = mupdf.fz_normalize_vector(c)
(4)              return c.y
(0)          def util_hor_matrix(C, P):
(4)              '''
(4)              Return the matrix that maps two points C, P to the x-axis such that
```

```
17162: (4)              C -> (0,0) and the image of P have the same distance.
17163: (4)              '''
17164: (4)              c = JM_point_from_py(C)
17165: (4)              p = JM_point_from_py(P)
17166: (4)              s = mupdf.fz_normalize_vector(mupdf.fz_make_point(p.x - c.x, p.y - c.y))
17167: (4)              m1 = mupdf.fz_make_matrix(1, 0, 0, 1, -c.x, -c.y)
17168: (4)              m2 = mupdf.fz_make_matrix(s.x, -s.y, s.y, s.x, 0, 0)
17169: (4)              return JM_py_from_matrix(mupdf.fz_concat(m1, m2))
17170: (0)          def match_string(h0, n0):
17171: (4)              h = 0
17172: (4)              n = 0
17173: (4)              e = h
17174: (4)              delta_h, hc = chartocanon(h0[h:])
17175: (4)              h += delta_h
17176: (4)              delta_n, nc = chartocanon(n0[n:])
17177: (4)              n += delta_n
17178: (4)              while hc == nc:
17179: (8)                  e = h
17180: (8)                  if hc == ord(' '):
17181: (12)                     while 1:
17182: (16)                         delta_h, hc = chartocanon(h0[h:])
17183: (16)                         h += delta_h
17184: (16)                         if hc != ord(' '):
17185: (20)                             break
17186: (8)                  else:
17187: (12)                     delta_h, hc = chartocanon(h0[h:])
17188: (12)                     h += delta_h
17189: (8)                  if nc == ord(' '):
17190: (12)                     while 1:
17191: (16)                         delta_n, nc = chartocanon(n0[n:])
17192: (16)                         n += delta_n
17193: (16)                         if nc != ord(' '):
17194: (20)                             break
17195: (8)                  else:
17196: (12)                     delta_n, nc = chartocanon(n0[n:])
17197: (12)                     n += delta_n
17198: (4)              return None if nc != 0 else e
17199: (0)          def on_highlight_char(hits, line, ch):
17200: (4)              assert hits
17201: (4)              assert isinstance(line, mupdf.FzStextLine)
17202: (4)              assert isinstance(ch, mupdf.FzStextChar)
17203: (4)              vfuzz = ch.m_internal.size * hits.vfuzz
17204: (4)              hfuzz = ch.m_internal.size * hits.hfuzz
17205: (4)              ch_quad = JM_char_quad(line, ch)
17206: (4)              if hits.len > 0:
17207: (8)                  quad = hits.quads[hits.len - 1]
17208: (8)                  end = JM_quad_from_py(quad)
17209: (8)                  if ( 1
17210: (16)                         and hdist(line.m_internal.dir, end.lr, ch_quad.ll) < hfuzz
17211: (16)                         and vdist(line.m_internal.dir, end.lr, ch_quad.ll) < vfuzz
17212: (16)                         and hdist(line.m_internal.dir, end.ur, ch_quad.ul) < hfuzz
17213: (16)                         and vdist(line.m_internal.dir, end.ur, ch_quad.ul) < vfuzz
17214: (16)                         ):
17215: (12)                     end.ur = ch_quad.ur
17216: (12)                     end.lr = ch_quad.lr
17217: (12)                     assert hits.quads[-1] == end
17218: (12)                     return
17219: (4)              hits.quads.append(ch_quad)
17220: (4)              hits.len += 1
17221: (0)          def page_merge(doc_des, doc_src, page_from, page_to, rotate, links,
copy_annots, graft_map):
17222: (4)              '''
17223: (4)              Deep-copies a source page to the target.
17224: (4)              Modified version of function of pdfmerge.c: we also copy annotations, but
17225: (4)              we skip some subtypes. In addition we rotate output.
17226: (4)              '''
17227: (4)              if g_use_extra:
17228: (8)                  return extra.page_merge( doc_des, doc_src, page_from, page_to, rotate,
links, copy_annots, graft_map)
```

```
17229: (4)              known_page_objs = [
17230: (8)                  PDF_NAME('Contents'),
17231: (8)                  PDF_NAME('Resources'),
17232: (8)                  PDF_NAME('MediaBox'),
17233: (8)                  PDF_NAME('CropBox'),
17234: (8)                  PDF_NAME('BleedBox'),
17235: (8)                  PDF_NAME('TrimBox'),
17236: (8)                  PDF_NAME('ArtBox'),
17237: (8)                  PDF_NAME('Rotate'),
17238: (8)                  PDF_NAME('UserUnit'),
17239: (8)                  ]
17240: (4)              page_ref = mupdf.pdf_lookup_page_obj(doc_src, page_from)
17241: (4)              page_dict = mupdf.pdf_new_dict(doc_des, 4)
17242: (4)              mupdf.pdf_dict_put(page_dict, PDF_NAME('Type'), PDF_NAME('Page'))
17243: (4)              for i in range( len(known_page_objs)):
17244: (8)                  obj = mupdf.pdf_dict_get_inheritable( page_ref, known_page_objs[i])
17245: (8)                  if obj.m_internal:
17246: (12)                     mupdf.pdf_dict_put( page_dict, known_page_objs[i],
mupdf.pdf_graft_mapped_object(graft_map.this, obj))
17247: (4)              if copy_annots:
17248: (8)                  old_annots = mupdf.pdf_dict_get( page_ref, PDF_NAME('Annots'))
17249: (8)                  n = mupdf.pdf_array_len( old_annots)
17250: (8)                  if n > 0:
17251: (12)                     new_annots = mupdf.pdf_dict_put_array( page_dict,
PDF_NAME('Annots'), n)
17252: (12)                     for i in range(n):
17253: (16)                         o = mupdf.pdf_array_get( old_annots, i)
17254: (16)                         if not o.m_internal or not mupdf.pdf_is_dict(o):
17255: (20)                             continue    # skip non-dict items
17256: (16)                         if mupdf.pdf_dict_gets( o, "IRT").m_internal:
17257: (20)                             continue
17258: (16)                         subtype = mupdf.pdf_dict_get( o, PDF_NAME('Subtype'))
17259: (16)                         if mupdf.pdf_name_eq( subtype, PDF_NAME('Link')):
17260: (20)                             continue
17261: (16)                         if mupdf.pdf_name_eq( subtype, PDF_NAME('Popup')):
17262: (20)                             continue
17263: (16)                         if mupdf.pdf_name_eq( subtype, PDF_NAME('Widget')):
17264: (20)                             mupdf.fz_warn( "skipping widget annotation")
17265: (20)                             continue
17266: (16)                         if mupdf.pdf_name_eq(subtype, PDF_NAME('Widget')):
17267: (20)                             continue
17268: (16)                         mupdf.pdf_dict_del( o, PDF_NAME('Popup'))
17269: (16)                         mupdf.pdf_dict_del( o, PDF_NAME('P'))
17270: (16)                         copy_o = mupdf.pdf_graft_mapped_object( graft_map.this, o)
17271: (16)                         annot = mupdf.pdf_new_indirect( doc_des, mupdf.pdf_to_num(
copy_o), 0)
17272: (16)                         mupdf.pdf_array_push( new_annots, annot)
17273: (4)              if rotate != -1:
17274: (8)                  mupdf.pdf_dict_put_int( page_dict, PDF_NAME('Rotate'), rotate)
17275: (4)              ref = mupdf.pdf_add_object( doc_des, page_dict)
17276: (4)              mupdf.pdf_insert_page( doc_des, page_to, ref)
17277: (0)          def paper_rect(s: str) -> Rect:
17278: (4)              """Return a Rect for the paper size indicated in string 's'. Must conform
to the argument of method 'PaperSize', which will be invoked.
17279: (4)              """
17280: (4)              width, height = paper_size(s)
17281: (4)              return Rect(0.0, 0.0, width, height)
17282: (0)          def paper_size(s: str) -> tuple:
17283: (4)              """Return a tuple (width, height) for a given paper format string.
17284: (4)              Notes:
17285: (8)                  'A4-L' will return (842, 595), the values for A4 landscape.
17286: (8)                  Suffix '-P' and no suffix return the portrait tuple.
17287: (4)              """
17288: (4)              size = s.lower()
17289: (4)              f = "p"
17290: (4)              if size.endswith("-l"):
17291: (8)                  f = "l"
17292: (8)                  size = size[:-2]
17293: (4)              if size.endswith("-p"):
```

```
17294: (8)                  size = size[:-2]
17295: (4)              rc = paper_sizes().get(size, (-1, -1))
17296: (4)              if f == "p":
17297: (8)                  return rc
17298: (4)              return (rc[1], rc[0])
17299: (0)          def paper_sizes():
17300: (4)              """Known paper formats @ 72 dpi as a dictionary. Key is the format string
17301: (4)              like "a4" for ISO-A4. Value is the tuple (width, height).
17302: (4)              Information taken from the following web sites:
17303: (4)              www.din-formate.de
17304: (4)              www.din-formate.info/amerikanische-formate.html
17305: (4)              www.directtools.de/wissen/normen/iso.htm
17306: (4)              """
17307: (4)              return {
17308: (8)                  "a0": (2384, 3370),
17309: (8)                  "a1": (1684, 2384),
17310: (8)                  "a10": (74, 105),
17311: (8)                  "a2": (1191, 1684),
17312: (8)                  "a3": (842, 1191),
17313: (8)                  "a4": (595, 842),
17314: (8)                  "a5": (420, 595),
17315: (8)                  "a6": (298, 420),
17316: (8)                  "a7": (210, 298),
17317: (8)                  "a8": (147, 210),
17318: (8)                  "a9": (105, 147),
17319: (8)                  "b0": (2835, 4008),
17320: (8)                  "b1": (2004, 2835),
17321: (8)                  "b10": (88, 125),
17322: (8)                  "b2": (1417, 2004),
17323: (8)                  "b3": (1001, 1417),
17324: (8)                  "b4": (709, 1001),
17325: (8)                  "b5": (499, 709),
17326: (8)                  "b6": (354, 499),
17327: (8)                  "b7": (249, 354),
17328: (8)                  "b8": (176, 249),
17329: (8)                  "b9": (125, 176),
17330: (8)                  "c0": (2599, 3677),
17331: (8)                  "c1": (1837, 2599),
17332: (8)                  "c10": (79, 113),
17333: (8)                  "c2": (1298, 1837),
17334: (8)                  "c3": (918, 1298),
17335: (8)                  "c4": (649, 918),
17336: (8)                  "c5": (459, 649),
17337: (8)                  "c6": (323, 459),
17338: (8)                  "c7": (230, 323),
17339: (8)                  "c8": (162, 230),
17340: (8)                  "c9": (113, 162),
17341: (8)                  "card-4x6": (288, 432),
17342: (8)                  "card-5x7": (360, 504),
17343: (8)                  "commercial": (297, 684),
17344: (8)                  "executive": (522, 756),
17345: (8)                  "invoice": (396, 612),
17346: (8)                  "ledger": (792, 1224),
17347: (8)                  "legal": (612, 1008),
17348: (8)                  "legal-13": (612, 936),
17349: (8)                  "letter": (612, 792),
17350: (8)                  "monarch": (279, 540),
17351: (8)                  "tabloid-extra": (864, 1296),
17352: (8)                  }
17353: (0)          if mupdf_version_tuple >= (1, 23, 8):
17354: (4)              def pdf_lookup_page_loc(doc, needle):
17355: (8)                  return mupdf.pdf_lookup_page_loc(doc, needle)
17356: (0)          else:
17357: (4)              def pdf_lookup_page_loc_imp(doc, node, skip, parentp, indexp):
17358: (8)                  assert isinstance(node, mupdf.PdfObj)
17359: (8)                  assert isinstance(skip, list) and len(skip) == 1
17360: (8)                  assert isinstance(indexp, list) and len(indexp) == 1
17361: (8)                  assert isinstance(parentp, list) and len(parentp) == 1 and
isinstance(parentp[0], mupdf.PdfObj)
```

```
17362: (8)                  hit = None
17363: (8)                  stack = []
17364: (8)                  try:
17365: (12)                     while 1:
17366: (16)                         kids = mupdf.pdf_dict_get(node, PDF_NAME('Kids'))
17367: (16)                         len_ = mupdf.pdf_array_len( kids)
17368: (16)                         if len_ == 0:
17369: (20)                             raise Exception("malformed page tree")
17370: (16)                         stack.append(node)
17371: (16)                         if mupdf.pdf_mark_obj( node):
17372: (20)                             raise Exception( "cycle in page tree")
17373: (16)                         for i in range(len_):
17374: (20)                             kid = mupdf.pdf_array_get( kids, i)
17375: (20)                             type_ = mupdf.pdf_dict_get( kid, PDF_NAME('Type'))
17376: (20)                             if type_.m_internal:
17377: (24)                                 a = mupdf.pdf_name_eq( type_, PDF_NAME('Pages'))
17378: (20)                             else:
17379: (24)                                 a = (
17380: (32)                                         mupdf.pdf_dict_get( kid,
PDF_NAME('Kids')).m_internal
17381: (32)                                         and not mupdf.pdf_dict_get( kid,
PDF_NAME('MediaBox')).m_internal
17382: (32)                                         )
17383: (20)                             if a:
17384: (24)                                 count = mupdf.pdf_dict_get_int( kid,
PDF_NAME('Count'))
17385: (24)                                 if (skip[0] < count):
17386: (28)                                     node = kid
17387: (28)                                     break
17388: (24)                                 else:
17389: (28)                                     skip[0] -= count
17390: (20)                             else:
17391: (24)                                 if type_.m_internal:
17392: (28)                                     a = not mupdf.pdf_name_eq( type_,
PDF_NAME('Page'))
17393: (24)                                 else:
17394: (28)                                     a = not mupdf.pdf_dict_get( kid,
PDF_NAME('MediaBox')).m_internal
17395: (24)                                 if a:
17396: (28)                                     mupdf.fz_warn( f"non-page object in page tree
({mupdf.pdf_to_name( type_)})")
17397: (24)                                 if skip[0] == 0:
17398: (28)                                     parentp[0] = node
17399: (28)                                     indexp[0] = i
17400: (28)                                     hit = kid
17401: (28)                                     break
17402: (24)                                 else:
17403: (28)                                     skip[0] -= 1
17404: (16)                         if not ((hit is None or hit.m_internal is None) and i < len_):
17405: (20)                             break
17406: (8)                  finally:
17407: (12)                     for i in range(len(stack), 0, -1): # (i = stack_len; i > 0; i--)
17408: (16)                         mupdf.pdf_unmark_obj( stack[i-1])
17409: (8)                  return hit
17410: (4)              def pdf_lookup_page_loc(doc, needle):
17411: (8)                  '''
17412: (8)                  Copy of MuPDF's internal pdf_lookup_page_loc().
17413: (8)                  '''
17414: (8)                  root = mupdf.pdf_dict_get( mupdf.pdf_trailer( doc), PDF_NAME('Root'))
17415: (8)                  node = mupdf.pdf_dict_get( root, PDF_NAME('Pages'))
17416: (8)                  skip = [needle]
17417: (8)                  if not node.m_internal:
17418: (12)                     raise Exception("cannot find page tree")
17419: (8)                  parentp = [mupdf.PdfObj()]
17420: (8)                  indexp = [0]
17421: (8)                  hit = pdf_lookup_page_loc_imp(doc, node, skip, parentp, indexp)
17422: (8)                  skip = skip[0]
17423: (8)                  parentp = parentp[0]
17424: (8)                  indexp = indexp[0]
```

```
17425: (8)                  if not hit.m_internal:
17426: (12)                     raise Exception("cannot find page %d in page tree" % needle+1)
17427: (8)                  return hit, parentp, indexp  # We don't seem to return skip.
17428: (0)          def pdfobj_string(o, prefix=''):
17429: (4)              '''
17430: (4)              Returns description of mupdf.PdfObj (wrapper for pdf_obj) <o>.
17431: (4)              '''
17432: (4)              assert 0, 'use mupdf.pdf_debug_obj() ?'
17433: (4)              ret = ''
17434: (4)              if mupdf.pdf_is_array(o):
17435: (8)                  l = mupdf.pdf_array_len(o)
17436: (8)                  ret += f'array {l}\n'
17437: (8)                  for i in range(l):
17438: (12)                     oo = mupdf.pdf_array_get(o, i)
17439: (12)                     ret += pdfobj_string(oo, prefix + '    ')
17440: (12)                     ret += '\n'
17441: (4)              elif mupdf.pdf_is_bool(o):
17442: (8)                  ret += f'bool: {o.array_get_bool()}\n'
17443: (4)              elif mupdf.pdf_is_dict(o):
17444: (8)                  l = mupdf.pdf_dict_len(o)
17445: (8)                  ret += f'dict {l}\n'
17446: (8)                  for i in range(l):
17447: (12)                     key = mupdf.pdf_dict_get_key(o, i)
17448: (12)                     value = mupdf.pdf_dict_get( o, key)
17449: (12)                     ret += f'{prefix} {key}: '
17450: (12)                     ret += pdfobj_string( value, prefix + '    ')
17451: (12)                     ret += '\n'
17452: (4)              elif mupdf.pdf_is_embedded_file(o):
17453: (8)                  ret += f'embedded_file: {o.embedded_file_name()}\n'
17454: (4)              elif mupdf.pdf_is_indirect(o):
17455: (8)                  ret += f'indirect: ...\n'
17456: (4)              elif mupdf.pdf_is_int(o):
17457: (8)                  ret += f'int: {mupdf.pdf_to_int(o)}\n'
17458: (4)              elif mupdf.pdf_is_jpx_image(o):
17459: (8)                  ret += f'jpx_image:\n'
17460: (4)              elif mupdf.pdf_is_name(o):
17461: (8)                  ret += f'name: {mupdf.pdf_to_name(o)}\n'
17462: (4)              elif o.pdf_is_null:
17463: (8)                  ret += f'null\n'
17464: (4)              elif o.pdf_is_real:
17465: (8)                  ret += f'real: {o.pdf_to_real()}\n'
17466: (4)              elif mupdf.pdf_is_stream(o):
17467: (8)                  ret += f'stream\n'
17468: (4)              elif mupdf.pdf_is_string(o):
17469: (8)                  ret += f'string: {mupdf.pdf_to_string(o)}\n'
17470: (4)              else:
17471: (8)                  ret += '<>\n'
17472: (4)              return ret
17473: (0)          def repair_mono_font(page: "Page", font: "Font") -> None:
17474: (4)              """Repair character spacing for mono fonts.
17475: (4)              Notes:
17476: (8)                  Some mono-spaced fonts are displayed with a too large character
17477: (8)                  distance, e.g. "a b c" instead of "abc". This utility adds an entry
17478: (8)                  "/W[0 65535 w]" to the descendent font(s) of font. The float w is
17479: (8)                  taken to be the width of 0x20 (space).
17480: (8)                  This should enforce viewers to use 'w' as the character width.
17481: (4)              Args:
17482: (8)                  page: pymupdf.Page object.
17483: (8)                  font: pymupdf.Font object.
17484: (4)              """
17485: (4)              if not font.flags["mono"]:  # font not flagged as monospaced
17486: (8)                  return None
17487: (4)              doc = page.parent  # the document
17488: (4)              fontlist = page.get_fonts()  # list of fonts on page
17489: (4)              xrefs = [  # list of objects referring to font
17490: (8)                  f[0]
17491: (8)                  for f in fontlist
17492: (8)                  if (f[3] == font.name and f[4].startswith("F") and
f[5].startswith("Identity"))
```

```
17493: (4)              ]
17494: (4)              if xrefs == []:  # our font does not occur
17495: (8)                  return
17496: (4)              xrefs = set(xrefs)  # drop any double counts
17497: (4)              width = int(round((font.glyph_advance(32) * 1000)))
17498: (4)              for xref in xrefs:
17499: (8)                  if not TOOLS.set_font_width(doc, xref, width):
17500: (12)                     log("Cannot set width for '%s' in xref %i" % (font.name, xref))
17501: (0)          def sRGB_to_pdf(srgb: int) -> tuple:
17502: (4)              """Convert sRGB color code to a PDF color triple.
17503: (4)              There is **no error checking** for performance reasons!
17504: (4)              Args:
17505: (8)                  srgb: (int) RRGGBB (red, green, blue), each color in range(255).
17506: (4)              Returns:
17507: (8)                  Tuple (red, green, blue) each item in interval 0 <= item <= 1.
17508: (4)              """
17509: (4)              t = sRGB_to_rgb(srgb)
17510: (4)              return t[0] / 255.0, t[1] / 255.0, t[2] / 255.0
17511: (0)          def sRGB_to_rgb(srgb: int) -> tuple:
17512: (4)              """Convert sRGB color code to an RGB color triple.
17513: (4)              There is **no error checking** for performance reasons!
17514: (4)              Args:
17515: (8)                  srgb: (int) RRGGBB (red, green, blue), each color in range(255).
17516: (4)              Returns:
17517: (8)                  Tuple (red, green, blue) each item in interval 0 <= item <= 255.
17518: (4)              """
17519: (4)              r = srgb >> 16
17520: (4)              g = (srgb - (r << 16)) >> 8
17521: (4)              b = srgb - (r << 16) - (g << 8)
17522: (4)              return (r, g, b)
17523: (0)          def string_in_names_list(p, names_list):
17524: (4)              n = mupdf.pdf_array_len( names_list) if names_list else 0
17525: (4)              str_ = mupdf.pdf_to_text_string( p)
17526: (4)              for i in range(0, n, 2):
17527: (8)                  if mupdf.pdf_to_text_string( mupdf.pdf_array_get( names_list, i)) ==
str_:
17528: (12)                     return 1
17529: (4)              return 0
17530: (0)          def strip_outline(doc, outlines, page_count, page_object_nums, names_list):
17531: (4)              '''
17532: (4)              Returns (count, first, prev).
17533: (4)              '''
17534: (4)              first = None
17535: (4)              count = 0
17536: (4)              current = outlines
17537: (4)              prev = None
17538: (4)              while current.m_internal:
17539: (8)                  nc = strip_outlines(doc, current, page_count, page_object_nums,
names_list)
17540: (8)                  if not dest_is_valid(current, page_count, page_object_nums,
names_list):
17541: (12)                     if nc == 0:
17542: (16)                         next = mupdf.pdf_dict_get(current, PDF_NAME('Next'))
17543: (16)                         if not next.m_internal:
17544: (20)                             if prev.m_internal:
17545: (24)                                 mupdf.pdf_dict_del(prev, PDF_NAME('Next'))
17546: (16)                         elif prev.m_internal:
17547: (20)                             mupdf.pdf_dict_put(prev, PDF_NAME('Next'), next)
17548: (20)                             mupdf.pdf_dict_put(next, PDF_NAME('Prev'), prev)
17549: (16)                         else:
17550: (20)                             mupdf.pdf_dict_del(next, PDF_NAME('Prev'))
17551: (16)                         current = next
17552: (12)                     else:
17553: (16)                         mupdf.pdf_dict_del(current, PDF_NAME('Dest'))
17554: (16)                         mupdf.pdf_dict_del(current, PDF_NAME('A'))
17555: (16)                         current = mupdf.pdf_dict_get(current, PDF_NAME('Next'))
17556: (8)                  else:
17557: (12)                     if not first or not first.m_internal:
17558: (16)                         first = current
```

```
17559: (12)                     prev = current
17560: (12)                     current = mupdf.pdf_dict_get(current, PDF_NAME('Next'))
17561: (12)                     count += 1
17562: (4)              return count, first, prev
17563: (0)          def strip_outlines(doc, outlines, page_count, page_object_nums, names_list):
17564: (4)              if not outlines.m_internal:
17565: (8)                  return 0
17566: (4)              first = mupdf.pdf_dict_get(outlines, PDF_NAME('First'))
17567: (4)              if not first.m_internal:
17568: (8)                  nc = 0
17569: (4)              else:
17570: (8)                  nc, first, last = strip_outline(doc, first, page_count,
page_object_nums, names_list)
17571: (4)              if nc == 0:
17572: (8)                  mupdf.pdf_dict_del(outlines, PDF_NAME('First'))
17573: (8)                  mupdf.pdf_dict_del(outlines, PDF_NAME('Last'))
17574: (8)                  mupdf.pdf_dict_del(outlines, PDF_NAME('Count'))
17575: (4)              else:
17576: (8)                  old_count = mupdf.pdf_to_int(mupdf.pdf_dict_get(outlines,
PDF_NAME('Count')))
17577: (8)                  mupdf.pdf_dict_put(outlines, PDF_NAME('First'), first)
17578: (8)                  mupdf.pdf_dict_put(outlines, PDF_NAME('Last'), last)
17579: (8)                  mupdf.pdf_dict_put(outlines, PDF_NAME('Count'), mupdf.pdf_new_int(nc
if old_count > 0 else -nc))
17580: (4)              return nc
17581: (0)          trace_device_FILL_PATH = 1
17582: (0)          trace_device_STROKE_PATH = 2
17583: (0)          trace_device_CLIP_PATH = 3
17584: (0)          trace_device_CLIP_STROKE_PATH = 4
17585: (0)          def unicode_to_glyph_name(ch: int) -> str:
17586: (4)              '''
17587: (4)              Adobe Glyph List function
17588: (4)              '''
17589: (4)              if _adobe_glyphs == {}:
17590: (8)                  for line in _get_glyph_text():
17591: (12)                     if line.startswith("#"):
17592: (16)                         continue
17593: (12)                     name, unc = line.split(";")
17594: (12)                     uncl = unc.split()
17595: (12)                     for unc in uncl:
17596: (16)                         c = int(unc[:4], base=16)
17597: (16)                         _adobe_glyphs[c] = name
17598: (4)              return _adobe_glyphs.get(ch, ".notdef")
17599: (0)          def vdist(dir, a, b):
17600: (4)              dx = b.x - a.x
17601: (4)              dy = b.y - a.y
17602: (4)              return mupdf.fz_abs(dx * dir.y + dy * dir.x)
17603: (0)          def apply_pages(
17604: (8)                  path,
17605: (8)                  pagefn,
17606: (8)                  *,
17607: (8)                  pagefn_args=(),
17608: (8)                  pagefn_kwargs=dict(),
17609: (8)                  initfn=None,
17610: (8)                  initfn_args=(),
17611: (8)                  initfn_kwargs=dict(),
17612: (8)                  pages=None,
17613: (8)                  method='single',
17614: (8)                  concurrency=None,
17615: (8)                  _stats=False,
17616: (8)                  ):
17617: (4)              '''
17618: (4)              Returns list of results from `pagefn()`, optionally using concurrency for
17619: (4)              speed.
17620: (4)              Args:
17621: (8)                  path:
17622: (12)                     Path of document.
17623: (8)                  pagefn:
17624: (12)                     Function to call for each page; is passed (page, *pagefn_args,
```

```
(12)                    **pagefn_kwargs). Return value is added to list that we return. If
(12)                    `method` is not 'single', must be a top-level function - nested
(12)                    functions don't work with concurrency.
(8)                 pagefn_args
(8)                 pagefn_kwargs:
(12)                    Additional args to pass to `pagefn`. Must be picklable.
(8)                 initfn:
(12)                    If true, called once in each worker process; is passed
(12)                    (*initfn_args, **initfn_kwargs).
(8)                 initfn_args
(8)                 initfn_kwargs:
(12)                    Args to pass to initfn. Must be picklable.
(8)                 pages:
(12)                    List of page numbers to process, or None to include all pages.
(8)                 method:
(12)                    'single'
(16)                        Do not use concurrency.
(12)                    'mp'
(16)                        Operate concurrently using Python's `multiprocessing` module.
(12)                    'fork'
(17)                         Operate concurrently using custom implementation with
(17)                         `os.fork()`. Does not work on Windows.
(8)                 concurrency:
(12)                    Number of worker processes to use when operating concurrently. If
(12)                    None, we use the number of available CPUs.
(8)                 _stats:
(12)                    Internal, may change or be removed. If true, we output simple
(12)                    timing diagnostics.
(4)             Note: We require a file path rather than a Document, because Document
(4)             instances do not work properly after a fork - internal file descriptor
(4)             offsets are shared between the parent and child processes.
(4)             '''
(4)             if _stats:
(8)                 t0 = time.time()
(4)             if method == 'single':
(8)                 if initfn:
(12)                    initfn(*initfn_args, **initfn_kwargs)
(8)                 ret = list()
(8)                 document = Document(path)
(8)                 for page in document:
(12)                    r = pagefn(page, *pagefn_args, **initfn_kwargs)
(12)                    ret.append(r)
(4)             else:
(8)                 from . import _apply_pages
(8)                 if pages is None:
(12)                    if _stats:
(16)                        t = time.time()
(12)                    with Document(path) as document:
(16)                        num_pages = len(document)
(16)                        pages = list(range(num_pages))
(12)                    if _stats:
(16)                        t = time.time() - t
(16)                        log(f'{t:.2f}s: count pages.')
(8)                 if _stats:
(12)                    t = time.time()
(8)                 if method == 'mp':
(12)                    ret = _apply_pages._multiprocessing(
(20)                            path,
(20)                            pages,
(20)                            pagefn,
(20)                            pagefn_args,
(20)                            pagefn_kwargs,
(20)                            initfn,
(20)                            initfn_args,
(20)                            initfn_kwargs,
(20)                            concurrency,
(20)                            _stats,
(20)                            )
(8)                 elif method == 'fork':
```

```
(12)                    ret = _apply_pages._fork(
(20)                            path,
(20)                            pages,
(20)                            pagefn,
(20)                            pagefn_args,
(20)                            pagefn_kwargs,
(20)                            initfn,
(20)                            initfn_args,
(20)                            initfn_kwargs,
(20)                            concurrency,
(20)                            _stats,
(20)                            )
(8)                 else:
(12)                    assert 0, f'Unrecognised {method=}.'
(8)                 if _stats:
(12)                    t = time.time() - t
(12)                    log(f'{t:.2f}s: work.')
(4)             if _stats:
(8)                 t = time.time() - t0
(8)                 log(f'{t:.2f}s: total.')
(4)             return ret
(0)         def get_text(
(8)                 path,
(8)                 *,
(8)                 pages=None,
(8)                 method='single',
(8)                 concurrency=None,
(8)                 option='text',
(8)                 clip=None,
(8)                 flags=None,
(8)                 textpage=None,
(8)                 sort=False,
(8)                 delimiters=None,
(8)                 _stats=False,
(8)                 ):
(4)             '''
(4)             Returns list of results from `Page.get_text()`, optionally using
(4)             concurrency for speed.
(4)             Args:
(8)                 path:
(12)                    Path of document.
(8)                 pages:
(12)                    List of page numbers to process, or None to include all pages.
(8)                 method:
(12)                    'single'
(16)                        Do not use concurrency.
(12)                    'mp'
(16)                        Operate concurrently using Python's `multiprocessing` module.
(12)                    'fork'
(17)                         Operate concurrently using custom implementation with
(17)                         `os.fork`. Does not work on Windows.
(8)                 concurrency:
(12)                    Number of worker processes to use when operating concurrently. If
(12)                    None, we use the number of available CPUs.
(8)                 option
(8)                 clip
(8)                 flags
(8)                 textpage
(8)                 sort
(8)                 delimiters:
(12)                    Passed to internal calls to `Page.get_text()`.
(4)             '''
(4)             args_dict = dict(
(12)                    option=option,
(12)                    clip=clip,
(12)                    flags=flags,
(12)                    textpage=textpage,
(12)                    sort=sort,
(12)                    delimiters=delimiters,
```

```
17763: (12)                     )
17764: (4)              return apply_pages(
17765: (12)                     path,
17766: (12)                     Page.get_text,
17767: (12)                     pagefn_kwargs=args_dict,
17768: (12)                     pages=pages,
17769: (12)                     method=method,
17770: (12)                     concurrency=concurrency,
17771: (12)                     _stats=_stats,
17772: (12)                     )
17773: (0)          class TOOLS:
17774: (4)              '''
17775: (4)              We use @staticmethod to avoid the need to create an instance of this
class.
17776: (4)              '''
17777: (4)              def _derotate_matrix(page):
17778: (8)                  if isinstance(page, mupdf.PdfPage):
17779: (12)                     return JM_py_from_matrix(JM_derotate_page_matrix(page))
17780: (8)                  else:
17781: (12)                     return JM_py_from_matrix(mupdf.FzMatrix())
17782: (4)              @staticmethod
17783: (4)              def _fill_widget(annot, widget):
17784: (8)                  val = JM_get_widget_properties(annot, widget)
17785: (8)                  widget.rect = Rect(annot.rect)
17786: (8)                  widget.xref = annot.xref
17787: (8)                  widget.parent = annot.parent
17788: (8)                  widget._annot = annot  # backpointer to annot object
17789: (8)                  if not widget.script:
17790: (12)                     widget.script = None
17791: (8)                  if not widget.script_stroke:
17792: (12)                     widget.script_stroke = None
17793: (8)                  if not widget.script_format:
17794: (12)                     widget.script_format = None
17795: (8)                  if not widget.script_change:
17796: (12)                     widget.script_change = None
17797: (8)                  if not widget.script_calc:
17798: (12)                     widget.script_calc = None
17799: (8)                  if not widget.script_blur:
17800: (12)                     widget.script_blur = None
17801: (8)                  if not widget.script_focus:
17802: (12)                     widget.script_focus = None
17803: (8)                  return val
17804: (4)              @staticmethod
17805: (4)              def _get_all_contents(page):
17806: (8)                  page = _as_pdf_page(page.this)
17807: (8)                  res = JM_read_contents(page.obj())
17808: (8)                  result = JM_BinFromBuffer( res)
17809: (8)                  return result
17810: (4)              @staticmethod
17811: (4)              def _insert_contents(page, newcont, overlay=1):
17812: (8)                  """Add bytes as a new /Contents object for a page, and return its
xref."""
17813: (8)                  pdfpage = _as_pdf_page(page, required=1)
17814: (8)                  contbuf = JM_BufferFromBytes(newcont)
17815: (8)                  xref = JM_insert_contents(pdfpage.doc(), pdfpage.obj(), contbuf,
overlay)
17816: (8)                  return xref
17817: (4)              @staticmethod
17818: (4)              def _le_annot_parms(annot, p1, p2, fill_color):
17819: (8)                  """Get common parameters for making annot line end symbols.
17820: (8)                  Returns:
17821: (12)                     m: matrix that maps p1, p2 to points L, P on the x-axis
17822: (12)                     im: its inverse
17823: (12)                     L, P: transformed p1, p2
17824: (12)                     w: line width
17825: (12)                     scol: stroke color string
17826: (12)                     fcol: fill color store_shrink
17827: (12)                     opacity: opacity string (gs command)
17828: (8)                  """
```

```
17829: (8)                  w = annot.border["width"]  # line width
17830: (8)                  sc = annot.colors["stroke"]  # stroke color
17831: (8)                  if not sc:  # black if missing
17832: (12)                     sc = (0,0,0)
17833: (8)                  scol = " ".join(map(str, sc)) + " RG\n"
17834: (8)                  if fill_color:
17835: (12)                     fc = fill_color
17836: (8)                  else:
17837: (12)                     fc = annot.colors["fill"]  # fill color
17838: (8)                  if not fc:
17839: (12)                     fc = (1,1,1)  # white if missing
17840: (8)                  fcol = " ".join(map(str, fc)) + " rg\n"
17841: (8)                  np1 = p1                   # point coord relative to annot rect
17842: (8)                  np2 = p2                   # point coord relative to annot rect
17843: (8)                  m = Matrix(util_hor_matrix(np1, np2))  # matrix makes the line
horizontal
17844: (8)                  im = ~m                            # inverted matrix
17845: (8)                  L = np1 * m                        # converted start (left) point
17846: (8)                  R = np2 * m                        # converted end (right) point
17847: (8)                  if 0 <= annot.opacity < 1:
17848: (12)                     opacity = "/H gs\n"
17849: (8)                  else:
17850: (12)                     opacity = ""
17851: (8)                  return m, im, L, R, w, scol, fcol, opacity
17852: (4)              @staticmethod
17853: (4)              def _le_butt(annot, p1, p2, lr, fill_color):
17854: (8)                  """Make stream commands for butt line end symbol. "lr" denotes left
(False) or right point.
17855: (8)                  """
17856: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17857: (8)                  shift = 3
17858: (8)                  d = shift * max(1, w)
17859: (8)                  M = R if lr else L
17860: (8)                  top = (M + (0, -d/2.)) * im
17861: (8)                  bot = (M + (0, d/2.)) * im
17862: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, top.x, top.y)
17863: (8)                  ap += "%f %f l\n" % (bot.x, bot.y)
17864: (8)                  ap += _format_g(w) + " w\n"
17865: (8)                  ap += scol + "s\nQ\n"
17866: (8)                  return ap
17867: (4)              @staticmethod
17868: (4)              def _le_circle(annot, p1, p2, lr, fill_color):
17869: (8)                  """Make stream commands for circle line end symbol. "lr" denotes left
(False) or right point.
17870: (8)                  """
17871: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17872: (8)                  shift = 2.5             # 2*shift*width = length of square edge
17873: (8)                  d = shift * max(1, w)
17874: (8)                  M = R - (d/2., 0) if lr else L + (d/2., 0)
17875: (8)                  r = Rect(M, M) + (-d, -d, d, d)         # the square
17876: (8)                  ap = "q\n" + opacity + TOOLS._oval_string(r.tl * im, r.tr * im, r.br *
im, r.bl * im)
17877: (8)                  ap += _format_g(w) + " w\n"
17878: (8)                  ap += scol + fcol + "b\nQ\n"
17879: (8)                  return ap
17880: (4)              @staticmethod
17881: (4)              def _le_closedarrow(annot, p1, p2, lr, fill_color):
17882: (8)                  """Make stream commands for closed arrow line end symbol. "lr" denotes
left (False) or right point.
17883: (8)                  """
17884: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17885: (8)                  shift = 2.5
17886: (8)                  d = shift * max(1, w)
17887: (8)                  p2 = R + (d/2., 0) if lr else L - (d/2., 0)
17888: (8)                  p1 = p2 + (-2*d, -d) if lr else p2 + (2*d, -d)
17889: (8)                  p3 = p2 + (-2*d, d) if lr else p2 + (2*d, d)
```

```
17890: (8)                  p1 *= im
17891: (8)                  p2 *= im
17892: (8)                  p3 *= im
17893: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, p1.x, p1.y)
17894: (8)                  ap += "%f %f l\n" % (p2.x, p2.y)
17895: (8)                  ap += "%f %f l\n" % (p3.x, p3.y)
17896: (8)                  ap += _format_g(w) + " w\n"
17897: (8)                  ap += scol + fcol + "b\nQ\n"
17898: (8)                  return ap
17899: (4)              @staticmethod
17900: (4)              def _le_diamond(annot, p1, p2, lr, fill_color):
17901: (8)                  """Make stream commands for diamond line end symbol. "lr" denotes left
(False) or right point.
17902: (8)                  """
17903: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17904: (8)                  shift = 2.5             # 2*shift*width = length of square edge
17905: (8)                  d = shift * max(1, w)
17906: (8)                  M = R - (d/2., 0) if lr else L + (d/2., 0)
17907: (8)                  r = Rect(M, M) + (-d, -d, d, d)         # the square
17908: (8)                  p = (r.tl + (r.bl - r.tl) * 0.5) * im
17909: (8)                  ap = "q\n%s%f %f m\n" % (opacity, p.x, p.y)
17910: (8)                  p = (r.tl + (r.tr - r.tl) * 0.5) * im
17911: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
17912: (8)                  p = (r.tr + (r.br - r.tr) * 0.5) * im
17913: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
17914: (8)                  p = (r.br + (r.bl - r.br) * 0.5) * im
17915: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
17916: (8)                  ap += _format_g(w) + " w\n"
17917: (8)                  ap += scol + fcol + "b\nQ\n"
17918: (8)                  return ap
17919: (4)              @staticmethod
17920: (4)              def _le_openarrow(annot, p1, p2, lr, fill_color):
17921: (8)                  """Make stream commands for open arrow line end symbol. "lr" denotes
left (False) or right point.
17922: (8)                  """
17923: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17924: (8)                  shift = 2.5
17925: (8)                  d = shift * max(1, w)
17926: (8)                  p2 = R + (d/2., 0) if lr else L - (d/2., 0)
17927: (8)                  p1 = p2 + (-2*d, -d) if lr else p2 + (2*d, -d)
17928: (8)                  p3 = p2 + (-2*d, d) if lr else p2 + (2*d, d)
17929: (8)                  p1 *= im
17930: (8)                  p2 *= im
17931: (8)                  p3 *= im
17932: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, p1.x, p1.y)
17933: (8)                  ap += "%f %f l\n" % (p2.x, p2.y)
17934: (8)                  ap += "%f %f l\n" % (p3.x, p3.y)
17935: (8)                  ap += _format_g(w) + " w\n"
17936: (8)                  ap += scol + "S\nQ\n"
17937: (8)                  return ap
17938: (4)              @staticmethod
17939: (4)              def _le_rclosedarrow(annot, p1, p2, lr, fill_color):
17940: (8)                  """Make stream commands for right closed arrow line end symbol. "lr"
denotes left (False) or right point.
17941: (8)                  """
17942: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17943: (8)                  shift = 2.5
17944: (8)                  d = shift * max(1, w)
17945: (8)                  p2 = R - (2*d, 0) if lr else L + (2*d, 0)
17946: (8)                  p1 = p2 + (2*d, -d) if lr else p2 + (-2*d, -d)
17947: (8)                  p3 = p2 + (2*d, d) if lr else p2 + (-2*d, d)
17948: (8)                  p1 *= im
17949: (8)                  p2 *= im
17950: (8)                  p3 *= im
17951: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, p1.x, p1.y)
17952: (8)                  ap += "%f %f l\n" % (p2.x, p2.y)
```

```
17953: (8)                  ap += "%f %f l\n" % (p3.x, p3.y)
17954: (8)                  ap += _format_g(w) + " w\n"
17955: (8)                  ap += scol + fcol + "b\nQ\n"
17956: (8)                  return ap
17957: (4)              @staticmethod
17958: (4)              def _le_ropenarrow(annot, p1, p2, lr, fill_color):
17959: (8)                  """Make stream commands for right open arrow line end symbol. "lr"
denotes left (False) or right point.
17960: (8)                  """
17961: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17962: (8)                  shift = 2.5
17963: (8)                  d = shift * max(1, w)
17964: (8)                  p2 = R - (d/3., 0) if lr else L + (d/3., 0)
17965: (8)                  p1 = p2 + (2*d, -d) if lr else p2 + (-2*d, -d)
17966: (8)                  p3 = p2 + (2*d, d) if lr else p2 + (-2*d, d)
17967: (8)                  p1 *= im
17968: (8)                  p2 *= im
17969: (8)                  p3 *= im
17970: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, p1.x, p1.y)
17971: (8)                  ap += "%f %f l\n" % (p2.x, p2.y)
17972: (8)                  ap += "%f %f l\n" % (p3.x, p3.y)
17973: (8)                  ap += _format_g(w) + " w\n"
17974: (8)                  ap += scol + fcol + "S\nQ\n"
17975: (8)                  return ap
17976: (4)              @staticmethod
17977: (4)              def _le_slash(annot, p1, p2, lr, fill_color):
17978: (8)                  """Make stream commands for slash line end symbol. "lr" denotes left
(False) or right point.
17979: (8)                  """
17980: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17981: (8)                  rw = 1.1547 * max(1, w) * 1.0         # makes rect diagonal a 30 deg
inclination
17982: (8)                  M = R if lr else L
17983: (8)                  r = Rect(M.x - rw, M.y - 2 * w, M.x + rw, M.y + 2 * w)
17984: (8)                  top = r.tl * im
17985: (8)                  bot = r.br * im
17986: (8)                  ap = "\nq\n%s%f %f m\n" % (opacity, top.x, top.y)
17987: (8)                  ap += "%f %f l\n" % (bot.x, bot.y)
17988: (8)                  ap += _format_g(w) + " w\n"
17989: (8)                  ap += scol + "s\nQ\n"
17990: (8)                  return ap
17991: (4)              @staticmethod
17992: (4)              def _le_square(annot, p1, p2, lr, fill_color):
17993: (8)                  """Make stream commands for square line end symbol. "lr" denotes left
(False) or right point.
17994: (8)                  """
17995: (8)                  m, im, L, R, w, scol, fcol, opacity = TOOLS._le_annot_parms(annot, p1,
p2, fill_color)
17996: (8)                  shift = 2.5             # 2*shift*width = length of square edge
17997: (8)                  d = shift * max(1, w)
17998: (8)                  M = R - (d/2., 0) if lr else L + (d/2., 0)
17999: (8)                  r = Rect(M, M) + (-d, -d, d, d)         # the square
18000: (8)                  p = r.tl * im
18001: (8)                  ap = "q\n%s%f %f m\n" % (opacity, p.x, p.y)
18002: (8)                  p = r.tr * im
18003: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
18004: (8)                  p = r.br * im
18005: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
18006: (8)                  p = r.bl * im
18007: (8)                  ap += "%f %f l\n"   % (p.x, p.y)
18008: (8)                  ap += _format_g(w) + " w\n"
18009: (8)                  ap += scol + fcol + "b\nQ\n"
18010: (8)                  return ap
18011: (4)              @staticmethod
18012: (4)              def _oval_string(p1, p2, p3, p4):
18013: (8)                  """Return /AP string defining an oval within a 4-polygon provided as
points
```

```
18014: (8)                  """
18015: (8)                  def bezier(p, q, r):
18016: (12)                     f = "%f %f %f %f %f %f c\n"
18017: (12)                     return f % (p.x, p.y, q.x, q.y, r.x, r.y)
18018: (8)                  kappa = 0.55228474983              # magic number
18019: (8)                  ml = p1 + (p4 - p1) * 0.5          # middle points ...
18020: (8)                  mo = p1 + (p2 - p1) * 0.5          # for each ...
18021: (8)                  mr = p2 + (p3 - p2) * 0.5          # polygon ...
18022: (8)                  mu = p4 + (p3 - p4) * 0.5          # side
18023: (8)                  ol1 = ml + (p1 - ml) * kappa       # the 8 bezier
18024: (8)                  ol2 = mo + (p1 - mo) * kappa       # helper points
18025: (8)                  or1 = mo + (p2 - mo) * kappa
18026: (8)                  or2 = mr + (p2 - mr) * kappa
18027: (8)                  ur1 = mr + (p3 - mr) * kappa
18028: (8)                  ur2 = mu + (p3 - mu) * kappa
18029: (8)                  ul1 = mu + (p4 - mu) * kappa
18030: (8)                  ul2 = ml + (p4 - ml) * kappa
18031: (8)                  ap = "%f %f m\n" % (ml.x, ml.y)
18032: (8)                  ap += bezier(ol1, ol2, mo)
18033: (8)                  ap += bezier(or1, or2, mr)
18034: (8)                  ap += bezier(ur1, ur2, mu)
18035: (8)                  ap += bezier(ul1, ul2, ml)
18036: (8)                  return ap
18037: (4)              @staticmethod
18038: (4)              def _parse_da(annot):
18039: (8)                  if g_use_extra:
18040: (12)                     val = extra.Tools_parse_da( annot.this)
18041: (8)                  else:
18042: (12)                     def Tools__parse_da(annot):
18043: (16)                         this_annot = annot.this
18044: (16)                         assert isinstance(this_annot, mupdf.PdfAnnot)
18045: (16)                         this_annot_obj = mupdf.pdf_annot_obj( this_annot)
18046: (16)                         pdf = mupdf.pdf_get_bound_document( this_annot_obj)
18047: (16)                         try:
18048: (20)                             da = mupdf.pdf_dict_get_inheritable( this_annot_obj,
PDF_NAME('DA'))
18049: (20)                             if not da.m_internal:
18050: (24)                                 trailer = mupdf.pdf_trailer(pdf)
18051: (24)                                 da = mupdf.pdf_dict_getl(trailer,
18052: (32)                                         PDF_NAME('Root'),
18053: (32)                                         PDF_NAME('AcroForm'),
18054: (32)                                         PDF_NAME('DA'),
18055: (32)                                         )
18056: (20)                             da_str = mupdf.pdf_to_text_string(da)
18057: (16)                         except Exception:
18058: (20)                             if g_exceptions_verbose:    exception_info()
18059: (20)                             return
18060: (16)                         return da_str
18061: (12)                     val = Tools__parse_da(annot)
18062: (8)                  if not val:
18063: (12)                     return ((0,), "", 0)
18064: (8)                  font = "Helv"
18065: (8)                  fsize = 12
18066: (8)                  col = (0, 0, 0)
18067: (8)                  dat = val.split()  # split on any whitespace
18068: (8)                  for i, item in enumerate(dat):
18069: (12)                     if item == "Tf":
18070: (16)                         font = dat[i - 2][1:]
18071: (16)                         fsize = float(dat[i - 1])
18072: (16)                         dat[i] = dat[i-1] = dat[i-2] = ""
18073: (16)                         continue
18074: (12)                     if item == "g":            # unicolor text
18075: (16)                         col = [(float(dat[i - 1]))]
18076: (16)                         dat[i] = dat[i-1] = ""
18077: (16)                         continue
18078: (12)                     if item == "rg":           # RGB colored text
18079: (16)                         col = [float(f) for f in dat[i - 3:i]]
18080: (16)                         dat[i] = dat[i-1] = dat[i-2] = dat[i-3] = ""
18081: (16)                         continue
```

```
18082: (12)                     if item == "k":           # CMYK colored text
18083: (16)                         col = [float(f) for f in dat[i - 4:i]]
18084: (16)                         dat[i] = dat[i-1] = dat[i-2] = dat[i-3] = dat[i-4] = ""
18085: (16)                         continue
18086: (8)                  val = (col, font, fsize)
18087: (8)                  return val
18088: (4)              @staticmethod
18089: (4)              def _reset_widget(annot):
18090: (8)                  this_annot = annot
18091: (8)                  this_annot_obj = mupdf.pdf_annot_obj(this_annot)
18092: (8)                  pdf = mupdf.pdf_get_bound_document(this_annot_obj)
18093: (8)                  mupdf.pdf_field_reset(pdf, this_annot_obj)
18094: (4)              @staticmethod
18095: (4)              def _rotate_matrix(page):
18096: (8)                  pdfpage = page._pdf_page(required=False)
18097: (8)                  if not pdfpage.m_internal:
18098: (12)                     return JM_py_from_matrix(mupdf.FzMatrix())
18099: (8)                  return JM_py_from_matrix(JM_rotate_page_matrix(pdfpage))
18100: (4)              @staticmethod
18101: (4)              def _save_widget(annot, widget):
18102: (8)                  JM_set_widget_properties(annot, widget)
18103: (4)              def _update_da(annot, da_str):
18104: (8)                  if g_use_extra:
18105: (12)                     extra.Tools_update_da( annot.this, da_str)
18106: (8)                  else:
18107: (12)                     try:
18108: (16)                         this_annot = annot.this
18109: (16)                         assert isinstance(this_annot, mupdf.PdfAnnot)
18110: (16)
mupdf.pdf_dict_put_text_string(mupdf.pdf_annot_obj(this_annot), PDF_NAME('DA'), da_str)
18111: (16)                         mupdf.pdf_dict_del(mupdf.pdf_annot_obj(this_annot),
PDF_NAME('DS'))    # /* not supported */
18112: (16)                         mupdf.pdf_dict_del(mupdf.pdf_annot_obj(this_annot),
PDF_NAME('RC'))    # /* not supported */
18113: (12)                     except Exception:
18114: (16)                         if g_exceptions_verbose:    exception_info()
18115: (16)                         return
18116: (12)                     return
18117: (4)              @staticmethod
18118: (4)              def gen_id():
18119: (8)                  global TOOLS_JM_UNIQUE_ID
18120: (8)                  TOOLS_JM_UNIQUE_ID += 1
18121: (8)                  return TOOLS_JM_UNIQUE_ID
18122: (4)              @staticmethod
18123: (4)              def glyph_cache_empty():
18124: (8)                  '''
18125: (8)                  Empty the glyph cache.
18126: (8)                  '''
18127: (8)                  mupdf.fz_purge_glyph_cache()
18128: (4)              @staticmethod
18129: (4)              def image_profile(stream, keep_image=0):
18130: (8)                  '''
18131: (8)                  Metadata of an image binary stream.
18132: (8)                  '''
18133: (8)                  return JM_image_profile(stream, keep_image)
18134: (4)              @staticmethod
18135: (4)              def mupdf_display_errors(on=None):
18136: (8)                  '''
18137: (8)                  Set MuPDF error display to True or False.
18138: (8)                  '''
18139: (8)                  global JM_mupdf_show_errors
18140: (8)                  if on is not None:
18141: (12)                     JM_mupdf_show_errors = bool(on)
18142: (8)                  return JM_mupdf_show_errors
18143: (4)              @staticmethod
18144: (4)              def mupdf_display_warnings(on=None):
18145: (8)                  '''
18146: (8)                  Set MuPDF warnings display to True or False.
18147: (8)                  '''
```

```
18148: (8)                  global JM_mupdf_show_warnings
18149: (8)                  if on is not None:
18150: (12)                     JM_mupdf_show_warnings = bool(on)
18151: (8)                  return JM_mupdf_show_warnings
18152: (4)              @staticmethod
18153: (4)              def mupdf_version():
18154: (8)                  '''Get version of MuPDF binary build.'''
18155: (8)                  return mupdf.FZ_VERSION
18156: (4)              @staticmethod
18157: (4)              def mupdf_warnings(reset=1):
18158: (8)                  '''
18159: (8)                  Get the MuPDF warnings/errors with optional reset (default).
18160: (8)                  '''
18161: (8)                  mupdf.fz_flush_warnings()
18162: (8)                  ret = '\n'.join( JM_mupdf_warnings_store)
18163: (8)                  if reset:
18164: (12)                     TOOLS.reset_mupdf_warnings()
18165: (8)                  return ret
18166: (4)              @staticmethod
18167: (4)              def reset_mupdf_warnings():
18168: (8)                  global JM_mupdf_warnings_store
18169: (8)                  JM_mupdf_warnings_store = list()
18170: (4)              @staticmethod
18171: (4)              def set_aa_level(level):
18172: (8)                  '''
18173: (8)                  Set anti-aliasing level.
18174: (8)                  '''
18175: (8)                  mupdf.fz_set_aa_level(level)
18176: (4)              @staticmethod
18177: (4)              def set_annot_stem( stem=None):
18178: (8)                  global JM_annot_id_stem
18179: (8)                  if stem is None:
18180: (12)                     return JM_annot_id_stem
18181: (8)                  len_ = len(stem) + 1
18182: (8)                  if len_ > 50:
18183: (12)                     len_ = 50
18184: (8)                  JM_annot_id_stem = stem[:50]
18185: (8)                  return JM_annot_id_stem
18186: (4)              @staticmethod
18187: (4)              def set_font_width(doc, xref, width):
18188: (8)                  pdf = _as_pdf_document(doc, required=0)
18189: (8)                  if not pdf.m_internal:
18190: (12)                     return False
18191: (8)                  font = mupdf.pdf_load_object(pdf, xref)
18192: (8)                  dfonts = mupdf.pdf_dict_get(font, PDF_NAME('DescendantFonts'))
18193: (8)                  if mupdf.pdf_is_array(dfonts):
18194: (12)                     n = mupdf.pdf_array_len(dfonts)
18195: (12)                     for i in range(n):
18196: (16)                         dfont = mupdf.pdf_array_get(dfonts, i)
18197: (16)                         warray = mupdf.pdf_new_array(pdf, 3)
18198: (16)                         mupdf.pdf_array_push(warray, mupdf.pdf_new_int(0))
18199: (16)                         mupdf.pdf_array_push(warray, mupdf.pdf_new_int(65535))
18200: (16)                         mupdf.pdf_array_push(warray, mupdf.pdf_new_int(width))
18201: (16)                         mupdf.pdf_dict_put(dfont, PDF_NAME('W'), warray)
18202: (8)                  return True
18203: (4)              @staticmethod
18204: (4)              def set_graphics_min_line_width(min_line_width):
18205: (8)                  '''
18206: (8)                  Set the graphics minimum line width.
18207: (8)                  '''
18208: (8)                  mupdf.fz_set_graphics_min_line_width(min_line_width)
18209: (4)              @staticmethod
18210: (4)              def set_icc( on=0):
18211: (8)                  """Set ICC color handling on or off."""
18212: (8)                  if on:
18213: (12)                     if mupdf.FZ_ENABLE_ICC:
18214: (16)                         mupdf.fz_enable_icc()
18215: (12)                     else:
18216: (16)                         RAISEPY( "MuPDF built w/o ICC support",PyExc_ValueError)
```

```
18217: (8)                  elif mupdf.FZ_ENABLE_ICC:
18218: (12)                     mupdf.fz_disable_icc()
18219: (4)              @staticmethod
18220: (4)              def set_low_memory( on=None):
18221: (8)                  """Set / unset MuPDF device caching."""
18222: (8)                  if on is not None:
18223: (12)                     _globals.no_device_caching = bool(on)
18224: (8)                  return _globals.no_device_caching
18225: (4)              @staticmethod
18226: (4)              def set_small_glyph_heights(on=None):
18227: (8)                  """Set / unset small glyph heights."""
18228: (8)                  if on is not None:
18229: (12)                     _globals.small_glyph_heights = bool(on)
18230: (12)                     if g_use_extra:
18231: (16)                         extra.set_small_glyph_heights(_globals.small_glyph_heights)
18232: (8)                  return _globals.small_glyph_heights
18233: (4)              @staticmethod
18234: (4)              def set_subset_fontnames(on=None):
18235: (8)                  '''
18236: (8)                  Set / unset returning fontnames with their subset prefix.
18237: (8)                  '''
18238: (8)                  if on is not None:
18239: (12)                     _globals.subset_fontnames = bool(on)
18240: (12)                     if g_use_extra:
18241: (16)                         extra.set_subset_fontnames(_globals.subset_fontnames)
18242: (8)                  return _globals.subset_fontnames
18243: (4)              @staticmethod
18244: (4)              def show_aa_level():
18245: (8)                  '''
18246: (8)                  Show anti-aliasing values.
18247: (8)                  '''
18248: (8)                  return dict(
18249: (16)                         graphics = mupdf.fz_graphics_aa_level(),
18250: (16)                         text = mupdf.fz_text_aa_level(),
18251: (16)                         graphics_min_line_width = mupdf.fz_graphics_min_line_width(),
18252: (16)                         )
18253: (4)              @staticmethod
18254: (4)              def store_maxsize():
18255: (8)                  '''
18256: (8)                  MuPDF store size limit.
18257: (8)                  '''
18258: (8)                  return None
18259: (4)              @staticmethod
18260: (4)              def store_shrink(percent):
18261: (8)                  '''
18262: (8)                  Free 'percent' of current store size.
18263: (8)                  '''
18264: (8)                  if percent >= 100:
18265: (12)                     mupdf.fz_empty_store()
18266: (12)                     return 0
18267: (8)                  if percent > 0:
18268: (12)                     mupdf.fz_shrink_store( 100 - percent)
18269: (4)              @staticmethod
18270: (4)              def store_size():
18271: (8)                  '''
18272: (8)                  MuPDF current store size.
18273: (8)                  '''
18274: (8)                  return None
18275: (4)              @staticmethod
18276: (4)              def unset_quad_corrections(on=None):
18277: (8)                  '''
18278: (8)                  Set ascender / descender corrections on or off.
18279: (8)                  '''
18280: (8)                  if on is not None:
18281: (12)                     _globals.skip_quad_corrections = bool(on)
18282: (12)                     if g_use_extra:
18283: (16)
extra.set_skip_quad_corrections(_globals.skip_quad_corrections)
18284: (8)                  return _globals.skip_quad_corrections
```

```
(4)             JM_annot_id_stem = 'fitz'
(4)             fitz_config = JM_fitz_config()
(0)         from . import utils
(0)         pdfcolor = dict(
(8)                 [
(12)                    (k, (r / 255, g / 255, b / 255))
(12)                    for k, (r, g, b) in utils.getColorInfoDict().items()
(8)                 ]
(8)                 )
(0)         if not mupdf_cppyy:
(4)             mupdf.fz_set_warning_callback(JM_mupdf_warning)
(4)             mupdf.fz_set_error_callback(JM_mupdf_error)
(0)         def _atexit():
(4)             mupdf.fz_flush_warnings()
(4)             mupdf.fz_set_warning_callback(None)
(4)             mupdf.fz_set_error_callback(None)
(0)         atexit.register( _atexit)
(0)         recover_bbox_quad           = utils.recover_bbox_quad
(0)         recover_char_quad           = utils.recover_char_quad
(0)         recover_line_quad           = utils.recover_line_quad
(0)         recover_quad                = utils.recover_quad
(0)         recover_span_quad           = utils.recover_span_quad
(0)         Annot.get_text              = utils.get_text
(0)         Annot.get_textbox           = utils.get_textbox
(0)         Document._do_links          = utils.do_links
(0)         Document.del_toc_item       = utils.del_toc_item
(0)         Document.get_char_widths    = utils.get_char_widths
(0)         Document.get_oc             = utils.get_oc
(0)         Document.get_ocmd           = utils.get_ocmd
(0)         Document.get_page_labels    = utils.get_page_labels
(0)         Document.get_page_numbers   = utils.get_page_numbers
(0)         Document.get_page_pixmap    = utils.get_page_pixmap
(0)         Document.get_page_text      = utils.get_page_text
(0)         Document.get_toc            = utils.get_toc
(0)         Document.has_annots         = utils.has_annots
(0)         Document.has_links          = utils.has_links
(0)         Document.insert_page        = utils.insert_page
(0)         Document.new_page           = utils.new_page
(0)         Document.scrub              = utils.scrub
(0)         Document.search_page_for    = utils.search_page_for
(0)         Document.set_metadata       = utils.set_metadata
(0)         Document.set_oc             = utils.set_oc
(0)         Document.set_ocmd           = utils.set_ocmd
(0)         Document.set_page_labels    = utils.set_page_labels
(0)         Document.set_toc            = utils.set_toc
(0)         Document.set_toc_item       = utils.set_toc_item
(0)         Document.subset_fonts       = utils.subset_fonts
(0)         Document.tobytes            = Document.write
(0)         Document.xref_copy          = utils.xref_copy
(0)         IRect.get_area              = utils.get_area
(0)         Page.apply_redactions       = utils.apply_redactions
(0)         Page.delete_image           = utils.delete_image
(0)         Page.delete_widget          = utils.delete_widget
(0)         Page.draw_bezier            = utils.draw_bezier
(0)         Page.draw_circle            = utils.draw_circle
(0)         Page.draw_curve             = utils.draw_curve
(0)         Page.draw_line              = utils.draw_line
(0)         Page.draw_oval              = utils.draw_oval
(0)         Page.draw_polyline          = utils.draw_polyline
(0)         Page.draw_quad              = utils.draw_quad
(0)         Page.draw_rect              = utils.draw_rect
(0)         Page.draw_sector            = utils.draw_sector
(0)         Page.draw_squiggle          = utils.draw_squiggle
(0)         Page.draw_zigzag            = utils.draw_zigzag
(0)         Page.get_image_info         = utils.get_image_info
(0)         Page.get_image_rects        = utils.get_image_rects
(0)         Page.get_label              = utils.get_label
(0)         Page.get_links              = utils.get_links
(0)         Page.get_pixmap             = utils.get_pixmap
```

```
18354: (0)          Page.get_text               = utils.get_text
18355: (0)          Page.get_text_blocks        = utils.get_text_blocks
18356: (0)          Page.get_text_selection     = utils.get_text_selection
18357: (0)          Page.get_text_words         = utils.get_text_words
18358: (0)          Page.get_textbox            = utils.get_textbox
18359: (0)          Page.get_textpage_ocr       = utils.get_textpage_ocr
18360: (0)          Page.insert_image           = utils.insert_image
18361: (0)          Page.insert_link            = utils.insert_link
18362: (0)          Page.insert_text            = utils.insert_text
18363: (0)          Page.insert_textbox         = utils.insert_textbox
18364: (0)          Page.insert_htmlbox         = utils.insert_htmlbox
18365: (0)          Page.new_shape              = lambda x: utils.Shape(x)
18366: (0)          Page.replace_image          = utils.replace_image
18367: (0)          Page.search_for             = utils.search_for
18368: (0)          Page.show_pdf_page          = utils.show_pdf_page
18369: (0)          Page.update_link            = utils.update_link
18370: (0)          Page.write_text             = utils.write_text
18371: (0)          from .table import find_tables
18372: (0)          Page.find_tables = find_tables
18373: (0)          Rect.get_area               = utils.get_area
18374: (0)          TextWriter.fill_textbox     = utils.fill_textbox
18375: (0)          class FitzDeprecation(DeprecationWarning):
18376: (4)              pass
18377: (0)          def restore_aliases():
18378: (4)              warnings.filterwarnings( "once", category=FitzDeprecation)
18379: (4)              def showthis(msg, cat, filename, lineno, file=None, line=None):
18380: (8)                  text = warnings.formatwarning(msg, cat, filename, lineno, line=line)
18381: (8)                  s = text.find("FitzDeprecation")
18382: (8)                  if s < 0:
18383: (12)                     log(text)
18384: (12)                     return
18385: (8)                  text = text[s:].splitlines()[0][4:]
18386: (8)                  log(text)
18387: (4)              warnings.showwarning = showthis
18388: (4)              def _alias(class_, new_name, legacy_name=None):
18389: (8)                  '''
18390: (8)                  Adds an alias for a class_ or module item clled <class_>.<new>.
18391: (8)                  class_:
18392: (12)                     Class/module to modify; use None for the current module.
18393: (8)                  new_name:
18394: (12)                     String name of existing item, e.g. name of method.
18395: (8)                  legacy_name:
18396: (12)                     Name of legacy object to create in <class_>. If None, we generate
18397: (12)                     from <item> by removing underscores and capitalising the next
18398: (12)                     letter.
18399: (8)                  '''
18400: (8)                  if class_ is None:
18401: (12)                     class_ = sys.modules[__name__]
18402: (8)                  if not legacy_name:
18403: (12)                     legacy_name = ''
18404: (12)                     capitalise_next = False
18405: (12)                     for c in new_name:
18406: (16)                         if c == '_':
18407: (20)                             capitalise_next = True
18408: (16)                         elif capitalise_next:
18409: (20)                             legacy_name += c.upper()
18410: (20)                             capitalise_next = False
18411: (16)                         else:
18412: (20)                             legacy_name += c
18413: (8)                  new_object = getattr( class_, new_name)
18414: (8)                  assert not getattr( class_, legacy_name, None), f'class {class_}
already has {legacy_name}'
18415: (8)                  if callable( new_object):
18416: (12)                     def deprecated_function( *args, **kwargs):
18417: (16)                         warnings.warn(
18418: (24)                                 f'"{legacy_name=}" removed from {class_} after v1.19.0
- use "{new_name}".',
18419: (24)                                 category=FitzDeprecation,
18420: (24)                                 )
```

```
18421: (16)                         return new_object( *args, **kwargs)
18422: (12)                     setattr( class_, legacy_name, deprecated_function)
18423: (12)                     deprecated_function.__doc__ = (
18424: (20)                             f'*** Deprecated and removed in version after v1.19.0 -
use "{new_name}". ***\n'
18425: (20)                             f'{new_object.__doc__}'
18426: (20)                             )
18427: (8)                  else:
18428: (12)                     setattr( class_, legacy_name, new_object)
18429: (4)              _alias( Annot, 'get_file',              'fileGet')
18430: (4)              _alias( Annot, 'get_pixmap')
18431: (4)              _alias( Annot, 'get_sound',             'soundGet')
18432: (4)              _alias( Annot, 'get_text')
18433: (4)              _alias( Annot, 'get_textbox')
18434: (4)              _alias( Annot, 'get_textpage',          'getTextPage')
18435: (4)              _alias( Annot, 'line_ends')
18436: (4)              _alias( Annot, 'set_blendmode',         'setBlendMode')
18437: (4)              _alias( Annot, 'set_border')
18438: (4)              _alias( Annot, 'set_colors')
18439: (4)              _alias( Annot, 'set_flags')
18440: (4)              _alias( Annot, 'set_info')
18441: (4)              _alias( Annot, 'set_line_ends')
18442: (4)              _alias( Annot, 'set_name')
18443: (4)              _alias( Annot, 'set_oc', 'setOC')
18444: (4)              _alias( Annot, 'set_opacity')
18445: (4)              _alias( Annot, 'set_rect')
18446: (4)              _alias( Annot, 'update_file',           'fileUpd')
18447: (4)              _alias( DisplayList, 'get_pixmap')
18448: (4)              _alias( DisplayList, 'get_textpage',    'getTextPage')
18449: (4)              _alias( Document, 'chapter_count')
18450: (4)              _alias( Document, 'chapter_page_count')
18451: (4)              _alias( Document, 'convert_to_pdf',     'convertToPDF')
18452: (4)              _alias( Document, 'copy_page')
18453: (4)              _alias( Document, 'delete_page')
18454: (4)              _alias( Document, 'delete_pages',       'deletePageRange')
18455: (4)              _alias( Document, 'embfile_add',        'embeddedFileAdd')
18456: (4)              _alias( Document, 'embfile_count',      'embeddedFileCount')
18457: (4)              _alias( Document, 'embfile_del',        'embeddedFileDel')
18458: (4)              _alias( Document, 'embfile_get',        'embeddedFileGet')
18459: (4)              _alias( Document, 'embfile_info',       'embeddedFileInfo')
18460: (4)              _alias( Document, 'embfile_names',      'embeddedFileNames')
18461: (4)              _alias( Document, 'embfile_upd',        'embeddedFileUpd')
18462: (4)              _alias( Document, 'extract_font')
18463: (4)              _alias( Document, 'extract_image')
18464: (4)              _alias( Document, 'find_bookmark')
18465: (4)              _alias( Document, 'fullcopy_page')
18466: (4)              _alias( Document, 'get_char_widths')
18467: (4)              _alias( Document, 'get_ocgs',           'getOCGs')
18468: (4)              _alias( Document, 'get_page_fonts',     'getPageFontList')
18469: (4)              _alias( Document, 'get_page_images',    'getPageImageList')
18470: (4)              _alias( Document, 'get_page_pixmap')
18471: (4)              _alias( Document, 'get_page_text')
18472: (4)              _alias( Document, 'get_page_xobjects',  'getPageXObjectList')
18473: (4)              _alias( Document, 'get_sigflags',       'getSigFlags')
18474: (4)              _alias( Document, 'get_toc', 'getToC')
18475: (4)              _alias( Document, 'get_xml_metadata')
18476: (4)              _alias( Document, 'insert_page')
18477: (4)              _alias( Document, 'insert_pdf',         'insertPDF')
18478: (4)              _alias( Document, 'is_dirty')
18479: (4)              _alias( Document, 'is_form_pdf',        'isFormPDF')
18480: (4)              _alias( Document, 'is_pdf', 'isPDF')
18481: (4)              _alias( Document, 'is_reflowable')
18482: (4)              _alias( Document, 'is_repaired')
18483: (4)              _alias( Document, 'last_location')
18484: (4)              _alias( Document, 'load_page')
18485: (4)              _alias( Document, 'make_bookmark')
18486: (4)              _alias( Document, 'move_page')
18487: (4)              _alias( Document, 'needs_pass')
18488: (4)              _alias( Document, 'new_page')
```

```
(4)             _alias( Document, 'next_location')
(4)             _alias( Document, 'page_count')
(4)             _alias( Document, 'page_cropbox',       'pageCropBox')
(4)             _alias( Document, 'page_xref')
(4)             _alias( Document, 'pdf_catalog',        'PDFCatalog')
(4)             _alias( Document, 'pdf_trailer',        'PDFTrailer')
(4)             _alias( Document, 'prev_location',      'previousLocation')
(4)             _alias( Document, 'resolve_link')
(4)             _alias( Document, 'search_page_for')
(4)             _alias( Document, 'set_language')
(4)             _alias( Document, 'set_metadata')
(4)             _alias( Document, 'set_toc', 'setToC')
(4)             _alias( Document, 'set_xml_metadata')
(4)             _alias( Document, 'update_object')
(4)             _alias( Document, 'update_stream')
(4)             _alias( Document, 'xref_is_stream',     'isStream')
(4)             _alias( Document, 'xref_length')
(4)             _alias( Document, 'xref_object')
(4)             _alias( Document, 'xref_stream')
(4)             _alias( Document, 'xref_stream_raw')
(4)             _alias( Document, 'xref_xml_metadata',  'metadataXML')
(4)             _alias( IRect, 'get_area')
(4)             _alias( IRect, 'get_area',              'getRectArea')
(4)             _alias( IRect, 'include_point')
(4)             _alias( IRect, 'include_rect')
(4)             _alias( IRect, 'is_empty')
(4)             _alias( IRect, 'is_infinite')
(4)             _alias( Link, 'is_external')
(4)             _alias( Link, 'set_border')
(4)             _alias( Link, 'set_colors')
(4)             _alias( Matrix, 'is_rectilinear')
(4)             _alias( Matrix, 'prerotate',            'preRotate')
(4)             _alias( Matrix, 'prescale',             'preScale')
(4)             _alias( Matrix, 'preshear',             'preShear')
(4)             _alias( Matrix, 'pretranslate',         'preTranslate')
(4)             _alias( None, 'get_pdf_now',            'getPDFnow')
(4)             _alias( None, 'get_pdf_str',            'getPDFstr')
(4)             _alias( None, 'get_text_length')
(4)             _alias( None, 'get_text_length',        'getTextlength')
(4)             _alias( None, 'image_profile',          'ImageProperties')
(4)             _alias( None, 'paper_rect',             'PaperRect')
(4)             _alias( None, 'paper_size',             'PaperSize')
(4)             _alias( None, 'paper_sizes')
(4)             _alias( None, 'planish_line')
(4)             _alias( Outline, 'is_external')
(4)             _alias( Outline, 'is_open')
(4)             _alias( Page, 'add_caret_annot')
(4)             _alias( Page, 'add_circle_annot')
(4)             _alias( Page, 'add_file_annot')
(4)             _alias( Page, 'add_freetext_annot')
(4)             _alias( Page, 'add_highlight_annot')
(4)             _alias( Page, 'add_ink_annot')
(4)             _alias( Page, 'add_line_annot')
(4)             _alias( Page, 'add_polygon_annot')
(4)             _alias( Page, 'add_polyline_annot')
(4)             _alias( Page, 'add_rect_annot')
(4)             _alias( Page, 'add_redact_annot')
(4)             _alias( Page, 'add_squiggly_annot')
(4)             _alias( Page, 'add_stamp_annot')
(4)             _alias( Page, 'add_strikeout_annot')
(4)             _alias( Page, 'add_text_annot')
(4)             _alias( Page, 'add_underline_annot')
(4)             _alias( Page, 'add_widget')
(4)             _alias( Page, 'clean_contents')
(4)             _alias( Page, 'cropbox',                'CropBox')
(4)             _alias( Page, 'cropbox_position',       'CropBoxPosition')
(4)             _alias( Page, 'delete_annot')
(4)             _alias( Page, 'delete_link')
(4)             _alias( Page, 'delete_widget')
```

```
(4)             _alias( Page, 'derotation_matrix')
(4)             _alias( Page, 'draw_bezier')
(4)             _alias( Page, 'draw_circle')
(4)             _alias( Page, 'draw_curve')
(4)             _alias( Page, 'draw_line')
(4)             _alias( Page, 'draw_oval')
(4)             _alias( Page, 'draw_polyline')
(4)             _alias( Page, 'draw_quad')
(4)             _alias( Page, 'draw_rect')
(4)             _alias( Page, 'draw_sector')
(4)             _alias( Page, 'draw_squiggle')
(4)             _alias( Page, 'draw_zigzag')
(4)             _alias( Page, 'first_annot')
(4)             _alias( Page, 'first_link')
(4)             _alias( Page, 'first_widget')
(4)             _alias( Page, 'get_contents')
(4)             _alias( Page, 'get_displaylist',        'getDisplayList')
(4)             _alias( Page, 'get_drawings')
(4)             _alias( Page, 'get_fonts',              'getFontList')
(4)             _alias( Page, 'get_image_bbox')
(4)             _alias( Page, 'get_images',             'getImageList')
(4)             _alias( Page, 'get_links')
(4)             _alias( Page, 'get_pixmap')
(4)             _alias( Page, 'get_svg_image',          'getSVGimage')
(4)             _alias( Page, 'get_text')
(4)             _alias( Page, 'get_text_blocks')
(4)             _alias( Page, 'get_text_words')
(4)             _alias( Page, 'get_textbox')
(4)             _alias( Page, 'get_textpage',           'getTextPage')
(4)             _alias( Page, 'insert_font')
(4)             _alias( Page, 'insert_image')
(4)             _alias( Page, 'insert_link')
(4)             _alias( Page, 'insert_text')
(4)             _alias( Page, 'insert_textbox')
(4)             _alias( Page, 'is_wrapped',             '_isWrapped')
(4)             _alias( Page, 'load_annot')
(4)             _alias( Page, 'load_links')
(4)             _alias( Page, 'mediabox',               'MediaBox')
(4)             _alias( Page, 'mediabox_size',          'MediaBoxSize')
(4)             _alias( Page, 'new_shape')
(4)             _alias( Page, 'read_contents')
(4)             _alias( Page, 'rotation_matrix')
(4)             _alias( Page, 'search_for')
(4)             _alias( Page, 'set_cropbox',            'setCropBox')
(4)             _alias( Page, 'set_mediabox',           'setMediaBox')
(4)             _alias( Page, 'set_rotation')
(4)             _alias( Page, 'show_pdf_page',          'showPDFpage')
(4)             _alias( Page, 'transformation_matrix')
(4)             _alias( Page, 'update_link')
(4)             _alias( Page, 'wrap_contents')
(4)             _alias( Page, 'write_text')
(4)             _alias( Pixmap, 'clear_with')
(4)             _alias( Pixmap, 'copy',                 'copyPixmap')
(4)             _alias( Pixmap, 'gamma_with')
(4)             _alias( Pixmap, 'invert_irect',         'invertIRect')
(4)             _alias( Pixmap, 'pil_save',             'pillowWrite')
(4)             _alias( Pixmap, 'pil_tobytes',          'pillowData')
(4)             _alias( Pixmap, 'save',                 'writeImage')
(4)             _alias( Pixmap, 'save',                 'writePNG')
(4)             _alias( Pixmap, 'set_alpha')
(4)             _alias( Pixmap, 'set_dpi',              'setResolution')
(4)             _alias( Pixmap, 'set_origin')
(4)             _alias( Pixmap, 'set_pixel')
(4)             _alias( Pixmap, 'set_rect')
(4)             _alias( Pixmap, 'tint_with')
(4)             _alias( Pixmap, 'tobytes',              'getImageData')
(4)             _alias( Pixmap, 'tobytes',              'getPNGData')
(4)             _alias( Pixmap, 'tobytes',              'getPNGdata')
(4)             _alias( Quad, 'is_convex')
```

```
18627: (4)              _alias( Quad, 'is_empty')
18628: (4)              _alias( Quad, 'is_rectangular')
18629: (4)              _alias( Rect, 'get_area')
18630: (4)              _alias( Rect, 'get_area',               'getRectArea')
18631: (4)              _alias( Rect, 'include_point')
18632: (4)              _alias( Rect, 'include_rect')
18633: (4)              _alias( Rect, 'is_empty')
18634: (4)              _alias( Rect, 'is_infinite')
18635: (4)              _alias( TextWriter, 'fill_textbox')
18636: (4)              _alias( TextWriter, 'write_text')
18637: (4)              _alias( utils.Shape, 'draw_bezier')
18638: (4)              _alias( utils.Shape, 'draw_circle')
18639: (4)              _alias( utils.Shape, 'draw_curve')
18640: (4)              _alias( utils.Shape, 'draw_line')
18641: (4)              _alias( utils.Shape, 'draw_oval')
18642: (4)              _alias( utils.Shape, 'draw_polyline')
18643: (4)              _alias( utils.Shape, 'draw_quad')
18644: (4)              _alias( utils.Shape, 'draw_rect')
18645: (4)              _alias( utils.Shape, 'draw_sector')
18646: (4)              _alias( utils.Shape, 'draw_squiggle')
18647: (4)              _alias( utils.Shape, 'draw_zigzag')
18648: (4)              _alias( utils.Shape, 'insert_text')
18649: (4)              _alias( utils.Shape, 'insert_textbox')
18650: (0)          if 0:
18651: (4)              restore_aliases()
18652: (0)          __version__ = VersionBind
18653: (0)          __doc__ = (
18654: (8)                  f'PyMuPDF {VersionBind}: Python bindings for the MuPDF {VersionFitz}
library (rebased implementation).\n'
18655: (8)                  f'Python {sys.version_info[0]}.{sys.version_info[1]} running on
{sys.platform} ({64 if sys.maxsize > 2**32 else 32}-bit).\n'
18656: (8)                  )

----------------------------------------

File 9 - __main__.py:

1: (0)              import argparse
2: (0)              import bisect
3: (0)              import os
4: (0)              import sys
5: (0)              import statistics
6: (0)              from typing import Dict, List, Set
7: (0)              from . import pymupdf
8: (0)              def mycenter(x):
9: (4)                  return (" %s " % x).center(75, "-")
10: (0)             def recoverpix(doc, item):
11: (4)                 """Return image for a given XREF."""
12: (4)                 x = item[0]  # xref of PDF image
13: (4)                 s = item[1]  # xref of its /SMask
14: (4)                 if s == 0:  # no smask: use direct image output
15: (8)                     return doc.extract_image(x)
16: (4)                 def getimage(pix):
17: (8)                     if pix.colorspace.n != 4:
18: (12)                        return pix
19: (8)                     tpix = pymupdf.Pixmap(pymupdf.csRGB, pix)
20: (8)                     return tpix
21: (4)                 pix1 = pymupdf.Pixmap(doc, x)
22: (4)                 pix2 = pymupdf.Pixmap(doc, s)  # create pixmap of the /SMask entry
23: (4)                 """Sanity check:
24: (4)                 - both pixmaps must have the same rectangle
25: (4)                 - both pixmaps must have alpha=0
26: (4)                 - pix2 must consist of 1 byte per pixel
27: (4)                 """
28: (4)                 if not (pix1.irect == pix2.irect and pix1.alpha == pix2.alpha == 0 and
pix2.n == 1):
29: (8)                     pymupdf.message("Warning: unsupported /SMask %i for %i:" % (s, x))
30: (8)                     pymupdf.message(pix2)
31: (8)                     pix2 = None
```

```
 32: (8)                     return getimage(pix1)  # return the pixmap as is
 33: (4)                 pix = pymupdf.Pixmap(pix1)  # copy of pix1, with an alpha channel added
 34: (4)                 pix.set_alpha(pix2.samples)  # treat pix2.samples as the alpha values
 35: (4)                 pix1 = pix2 = None  # free temp pixmaps
 36: (4)                 return getimage(pix)
 37: (0)             def open_file(filename, password, show=False, pdf=True):
 38: (4)                 """Open and authenticate a document."""
 39: (4)                 doc = pymupdf.open(filename)
 40: (4)                 if not doc.is_pdf and pdf is True:
 41: (8)                     sys.exit("this command supports PDF files only")
 42: (4)                 rc = -1
 43: (4)                 if not doc.needs_pass:
 44: (8)                     return doc
 45: (4)                 if password:
 46: (8)                     rc = doc.authenticate(password)
 47: (8)                     if not rc:
 48: (12)                        sys.exit("authentication unsuccessful")
 49: (8)                     if show is True:
 50: (12)                        pymupdf.message("authenticated as %s" % "owner" if rc > 2 else
"user")
 51: (4)                 else:
 52: (8)                     sys.exit("'%s' requires a password" % doc.name)
 53: (4)                 return doc
 54: (0)             def print_dict(item):
 55: (4)                 """Print a Python dictionary."""
 56: (4)                 l = max([len(k) for k in item.keys()]) + 1
 57: (4)                 for k, v in item.items():
 58: (8)                     msg = "%s: %s" % (k.rjust(l), v)
 59: (8)                     pymupdf.message(msg)
 60: (0)             def print_xref(doc, xref):
 61: (4)                 """Print an object given by XREF number.
 62: (4)                 Simulate the PDF source in "pretty" format.
 63: (4)                 For a stream also print its size.
 64: (4)                 """
 65: (4)                 pymupdf.message("%i 0 obj" % xref)
 66: (4)                 xref_str = doc.xref_object(xref)
 67: (4)                 pymupdf.message(xref_str)
 68: (4)                 if doc.xref_is_stream(xref):
 69: (8)                     temp = xref_str.split()
 70: (8)                     try:
 71: (12)                        idx = temp.index("/Length") + 1
 72: (12)                        size = temp[idx]
 73: (12)                        if size.endswith("0 R"):
 74: (16)                            size = "unknown"
 75: (8)                     except Exception:
 76: (12)                        size = "unknown"
 77: (8)                     pymupdf.message("stream\n...%s bytes" % size)
 78: (8)                     pymupdf.message("endstream")
 79: (4)                 pymupdf.message("endobj")
 80: (0)             def get_list(rlist, limit, what="page"):
 81: (4)                 """Transform a page / xref specification into a list of integers.
 82: (4)                 Args
 83: (4)                 ----
 84: (8)                     rlist: (str) the specification
 85: (8)                     limit: maximum number, i.e. number of pages, number of objects
 86: (8)                     what: a string to be used in error messages
 87: (4)                 Returns
 88: (4)                 -------
 89: (8)                     A list of integers representing the specification.
 90: (4)                 """
 91: (4)                 N = str(limit - 1)
 92: (4)                 rlist = rlist.replace("N", N).replace(" ", "")
 93: (4)                 rlist_arr = rlist.split(",")
 94: (4)                 out_list = []
 95: (4)                 for seq, item in enumerate(rlist_arr):
 96: (8)                     n = seq + 1
 97: (8)                     if item.isdecimal():  # a single integer
 98: (12)                        i = int(item)
 99: (12)                        if 1 <= i < limit:
```

```
100: (16)                            out_list.append(int(item))
101: (12)                        else:
102: (16)                            sys.exit("bad %s specification at item %i" % (what, n))
103: (12)                        continue
104: (8)                    try:  # this must be a range now, and all of the following must work:
105: (12)                        i1, i2 = item.split("-")  # will fail if not 2 items produced
106: (12)                        i1 = int(i1)  # will fail on non-integers
107: (12)                        i2 = int(i2)
108: (8)                    except Exception:
109: (12)                        sys.exit("bad %s range specification at item %i" % (what, n))
110: (8)                    if not (1 <= i1 < limit and 1 <= i2 < limit):
111: (12)                        sys.exit("bad %s range specification at item %i" % (what, n))
112: (8)                    if i1 == i2:  # just in case: a range of equal numbers
113: (12)                        out_list.append(i1)
114: (12)                        continue
115: (8)                    if i1 < i2:  # first less than second
116: (12)                        out_list += list(range(i1, i2 + 1))
117: (8)                    else:  # first larger than second
118: (12)                        out_list += list(range(i1, i2 - 1, -1))
119: (4)                return out_list
120: (0)            def show(args):
121: (4)                doc = open_file(args.input, args.password, True)
122: (4)                size = os.path.getsize(args.input) / 1024
123: (4)                flag = "KB"
124: (4)                if size > 1000:
125: (8)                    size /= 1024
126: (8)                    flag = "MB"
127: (4)                size = round(size, 1)
128: (4)                meta = doc.metadata # pylint: disable=no-member
129: (4)                pymupdf.message(
130: (8)                    "'%s', pages: %i, objects: %i, %g %s, %s, encryption: %s"
131: (8)                    % (
132: (12)                       args.input,
133: (12)                       doc.page_count,
134: (12)                       doc.xref_length() - 1,
135: (12)                       size,
136: (12)                       flag,
137: (12)                       meta["format"],
138: (12)                       meta["encryption"],
139: (8)                    )
140: (4)                )
141: (4)                n = doc.is_form_pdf
142: (4)                if n > 0:
143: (8)                    s = doc.get_sigflags()
144: (8)                    pymupdf.message(
145: (12)                       "document contains %i root form fields and is %ssigned"
146: (12)                       % (n, "not " if s != 3 else "")
147: (8)                    )
148: (4)                n = doc.embfile_count()
149: (4)                if n > 0:
150: (8)                    pymupdf.message("document contains %i embedded files" % n)
151: (4)                pymupdf.message()
152: (4)                if args.catalog:
153: (8)                    pymupdf.message(mycenter("PDF catalog"))
154: (8)                    xref = doc.pdf_catalog()
155: (8)                    print_xref(doc, xref)
156: (8)                    pymupdf.message()
157: (4)                if args.metadata:
158: (8)                    pymupdf.message(mycenter("PDF metadata"))
159: (8)                    print_dict(doc.metadata)    # pylint: disable=no-member
160: (8)                    pymupdf.message()
161: (4)                if args.xrefs:
162: (8)                    pymupdf.message(mycenter("object information"))
163: (8)                    xrefl = get_list(args.xrefs, doc.xref_length(), what="xref")
164: (8)                    for xref in xrefl:
165: (12)                       print_xref(doc, xref)
166: (12)                       pymupdf.message()
167: (4)                if args.pages:
168: (8)                    pymupdf.message(mycenter("page information"))
```

```
169: (8)                    pagel = get_list(args.pages, doc.page_count + 1)
170: (8)                    for pno in pagel:
171: (12)                       n = pno - 1
172: (12)                       xref = doc.page_xref(n)
173: (12)                       pymupdf.message("Page %i:" % pno)
174: (12)                       print_xref(doc, xref)
175: (12)                       pymupdf.message()
176: (4)                if args.trailer:
177: (8)                    pymupdf.message(mycenter("PDF trailer"))
178: (8)                    pymupdf.message(doc.pdf_trailer())
179: (8)                    pymupdf.message()
180: (4)                doc.close()
181: (0)            def clean(args):
182: (4)                doc = open_file(args.input, args.password, pdf=True)
183: (4)                encryption = args.encryption
184: (4)                encrypt = ("keep", "none", "rc4-40", "rc4-128", "aes-128", "aes-
256").index(
185: (8)                    encryption
186: (4)                )
187: (4)                if not args.pages:  # simple cleaning
188: (8)                    doc.save(
189: (12)                       args.output,
190: (12)                       garbage=args.garbage,
191: (12)                       deflate=args.compress,
192: (12)                       pretty=args.pretty,
193: (12)                       clean=args.sanitize,
194: (12)                       ascii=args.ascii,
195: (12)                       linear=args.linear,
196: (12)                       encryption=encrypt,
197: (12)                       owner_pw=args.owner,
198: (12)                       user_pw=args.user,
199: (12)                       permissions=args.permission,
200: (8)                    )
201: (8)                    return
202: (4)                pages = get_list(args.pages, doc.page_count + 1)
203: (4)                outdoc = pymupdf.open()
204: (4)                for pno in pages:
205: (8)                    n = pno - 1
206: (8)                    outdoc.insert_pdf(doc, from_page=n, to_page=n)
207: (4)                outdoc.save(
208: (8)                    args.output,
209: (8)                    garbage=args.garbage,
210: (8)                    deflate=args.compress,
211: (8)                    pretty=args.pretty,
212: (8)                    clean=args.sanitize,
213: (8)                    ascii=args.ascii,
214: (8)                    linear=args.linear,
215: (8)                    encryption=encrypt,
216: (8)                    owner_pw=args.owner,
217: (8)                    user_pw=args.user,
218: (8)                    permissions=args.permission,
219: (4)                )
220: (4)                doc.close()
221: (4)                outdoc.close()
222: (4)                return
223: (0)            def doc_join(args):
224: (4)                """Join pages from several PDF documents."""
225: (4)                doc_list = args.input  # a list of input PDFs
226: (4)                doc = pymupdf.open()  # output PDF
227: (4)                for src_item in doc_list:  # process one input PDF
228: (8)                    src_list = src_item.split(",")
229: (8)                    password = src_list[1] if len(src_list) > 1 else None
230: (8)                    src = open_file(src_list[0], password, pdf=True)
231: (8)                    pages = ",".join(src_list[2:])  # get 'pages' specifications
232: (8)                    if pages:  # if anything there, retrieve a list of desired pages
233: (12)                       page_list = get_list(",".join(src_list[2:]), src.page_count + 1)
234: (8)                    else:  # take all pages
235: (12)                       page_list = range(1, src.page_count + 1)
236: (8)                    for i in page_list:
```

```
237: (12)                        doc.insert_pdf(src, from_page=i - 1, to_page=i - 1)  # copy each
source page
238: (8)                    src.close()
239: (4)                doc.save(args.output, garbage=4, deflate=True)
240: (4)                doc.close()
241: (0)            def embedded_copy(args):
242: (4)                """Copy embedded files between PDFs."""
243: (4)                doc = open_file(args.input, args.password, pdf=True)
244: (4)                if not doc.can_save_incrementally() and (
245: (8)                    not args.output or args.output == args.input
246: (4)                ):
247: (8)                    sys.exit("cannot save PDF incrementally")
248: (4)                src = open_file(args.source, args.pwdsource)
249: (4)                names = set(args.name) if args.name else set()
250: (4)                src_names = set(src.embfile_names())
251: (4)                if names:
252: (8)                    if not names <= src_names:
253: (12)                       sys.exit("not all names are contained in source")
254: (4)                else:
255: (8)                    names = src_names
256: (4)                if not names:
257: (8)                    sys.exit("nothing to copy")
258: (4)                intersect = names & set(doc.embfile_names())  # any equal name already in
target?
259: (4)                if intersect:
260: (8)                    sys.exit("following names already exist in receiving PDF: %s" %
str(intersect))
261: (4)                for item in names:
262: (8)                    info = src.embfile_info(item)
263: (8)                    buff = src.embfile_get(item)
264: (8)                    doc.embfile_add(
265: (12)                       item,
266: (12)                       buff,
267: (12)                       filename=info["filename"],
268: (12)                       ufilename=info["ufilename"],
269: (12)                       desc=info["desc"],
270: (8)                    )
271: (8)                    pymupdf.message("copied entry '%s' from '%s'" % (item, src.name))
272: (4)                src.close()
273: (4)                if args.output and args.output != args.input:
274: (8)                    doc.save(args.output, garbage=3)
275: (4)                else:
276: (8)                    doc.saveIncr()
277: (4)                doc.close()
278: (0)            def embedded_del(args):
279: (4)                """Delete an embedded file entry."""
280: (4)                doc = open_file(args.input, args.password, pdf=True)
281: (4)                if not doc.can_save_incrementally() and (
282: (8)                    not args.output or args.output == args.input
283: (4)                ):
284: (8)                    sys.exit("cannot save PDF incrementally")
285: (4)                exception_types = (ValueError, pymupdf.mupdf.FzErrorBase)
286: (4)                if pymupdf.mupdf_version_tuple < (1, 24):
287: (8)                    exception_types = ValueError
288: (4)                try:
289: (8)                    doc.embfile_del(args.name)
290: (4)                except exception_types as e:    # pylint: disable=catching-non-exception
291: (8)                    sys.exit(f'no such embedded file {args.name!r}: {e}')
292: (4)                if not args.output or args.output == args.input:
293: (8)                    doc.saveIncr()
294: (4)                else:
295: (8)                    doc.save(args.output, garbage=1)
296: (4)                doc.close()
297: (0)            def embedded_get(args):
298: (4)                """Retrieve contents of an embedded file."""
299: (4)                doc = open_file(args.input, args.password, pdf=True)
300: (4)                exception_types = (ValueError, pymupdf.mupdf.FzErrorBase)
301: (4)                if pymupdf.mupdf_version_tuple < (1, 24):
302: (8)                    exception_types = ValueError
```

```
(4)                 try:
(8)                     stream = doc.embfile_get(args.name)
(8)                     d = doc.embfile_info(args.name)
(4)                 except exception_types as e:    # pylint: disable=catching-non-exception
(8)                     sys.exit(f'no such embedded file {args.name!r}: {e}')
(4)                 filename = args.output if args.output else d["filename"]
(4)                 with open(filename, "wb") as output:
(8)                     output.write(stream)
(4)                 pymupdf.message("saved entry '%s' as '%s'" % (args.name, filename))
(4)                 doc.close()
(0)             def embedded_add(args):
(4)                 """Insert a new embedded file."""
(4)                 doc = open_file(args.input, args.password, pdf=True)
(4)                 if not doc.can_save_incrementally() and (
(8)                     args.output is None or args.output == args.input
(4)                 ):
(8)                     sys.exit("cannot save PDF incrementally")
(4)                 try:
(8)                     doc.embfile_del(args.name)
(8)                     sys.exit("entry '%s' already exists" % args.name)
(4)                 except Exception:
(8)                     pass
(4)                 if not os.path.exists(args.path) or not os.path.isfile(args.path):
(8)                     sys.exit("no such file '%s'" % args.path)
(4)                 with open(args.path, "rb") as f:
(8)                     stream = f.read()
(4)                 filename = args.path
(4)                 ufilename = filename
(4)                 if not args.desc:
(8)                     desc = filename
(4)                 else:
(8)                     desc = args.desc
(4)                 doc.embfile_add(
(8)                     args.name, stream, filename=filename, ufilename=ufilename, desc=desc
(4)                 )
(4)                 if not args.output or args.output == args.input:
(8)                     doc.saveIncr()
(4)                 else:
(8)                     doc.save(args.output, garbage=3)
(4)                 doc.close()
(0)             def embedded_upd(args):
(4)                 """Update contents or metadata of an embedded file."""
(4)                 doc = open_file(args.input, args.password, pdf=True)
(4)                 if not doc.can_save_incrementally() and (
(8)                     args.output is None or args.output == args.input
(4)                 ):
(8)                     sys.exit("cannot save PDF incrementally")
(4)                 try:
(8)                     doc.embfile_info(args.name)
(4)                 except Exception:
(8)                     sys.exit("no such embedded file '%s'" % args.name)
(4)                 if (
(8)                     args.path is not None
(8)                     and os.path.exists(args.path)
(8)                     and os.path.isfile(args.path)
(4)                 ):
(8)                     with open(args.path, "rb") as f:
(12)                        stream = f.read()
(4)                 else:
(8)                     stream = None
(4)                 if args.filename:
(8)                     filename = args.filename
(4)                 else:
(8)                     filename = None
(4)                 if args.ufilename:
(8)                     ufilename = args.ufilename
(4)                 elif args.filename:
(8)                     ufilename = args.filename
(4)                 else:
```

```
372: (8)                    ufilename = None
373: (4)                if args.desc:
374: (8)                    desc = args.desc
375: (4)                else:
376: (8)                    desc = None
377: (4)                doc.embfile_upd(
378: (8)                    args.name, stream, filename=filename, ufilename=ufilename, desc=desc
379: (4)                )
380: (4)                if args.output is None or args.output == args.input:
381: (8)                    doc.saveIncr()
382: (4)                else:
383: (8)                    doc.save(args.output, garbage=3)
384: (4)                doc.close()
385: (0)            def embedded_list(args):
386: (4)                """List embedded files."""
387: (4)                doc = open_file(args.input, args.password, pdf=True)
388: (4)                names = doc.embfile_names()
389: (4)                if args.name is not None:
390: (8)                    if args.name not in names:
391: (12)                       sys.exit("no such embedded file '%s'" % args.name)
392: (8)                    else:
393: (12)                       pymupdf.message()
394: (12)                       pymupdf.message(
395: (16)                           "printing 1 of %i embedded file%s:"
396: (16)                           % (len(names), "s" if len(names) > 1 else "")
397: (12)                       )
398: (12)                       pymupdf.message()
399: (12)                       print_dict(doc.embfile_info(args.name))
400: (12)                       pymupdf.message()
401: (12)                       return
402: (4)                if not names:
403: (8)                    pymupdf.message("'%s' contains no embedded files" % doc.name)
404: (8)                    return
405: (4)                if len(names) > 1:
406: (8)                    msg = "'%s' contains the following %i embedded files" % (doc.name,
len(names))
407: (4)                else:
408: (8)                    msg = "'%s' contains the following embedded file" % doc.name
409: (4)                pymupdf.message(msg)
410: (4)                pymupdf.message()
411: (4)                for name in names:
412: (8)                    if not args.detail:
413: (12)                       pymupdf.message(name)
414: (12)                       continue
415: (8)                    _ = doc.embfile_info(name)
416: (8)                    print_dict(doc.embfile_info(name))
417: (8)                    pymupdf.message()
418: (4)                doc.close()
419: (0)            def extract_objects(args):
420: (4)                """Extract images and / or fonts from a PDF."""
421: (4)                if not args.fonts and not args.images:
422: (8)                    sys.exit("neither fonts nor images requested")
423: (4)                doc = open_file(args.input, args.password, pdf=True)
424: (4)                if args.pages:
425: (8)                    pages = get_list(args.pages, doc.page_count + 1)
426: (4)                else:
427: (8)                    pages = range(1, doc.page_count + 1)
428: (4)                if not args.output:
429: (8)                    out_dir = os.path.abspath(os.curdir)
430: (4)                else:
431: (8)                    out_dir = args.output
432: (8)                    if not (os.path.exists(out_dir) and os.path.isdir(out_dir)):
433: (12)                       sys.exit("output directory %s does not exist" % out_dir)
434: (4)                font_xrefs = set()  # already saved fonts
435: (4)                image_xrefs = set()  # already saved images
436: (4)                for pno in pages:
437: (8)                    if args.fonts:
438: (12)                       itemlist = doc.get_page_fonts(pno - 1)
439: (12)                       for item in itemlist:
```

```
440: (16)                            xref = item[0]
441: (16)                            if xref not in font_xrefs:
442: (20)                                font_xrefs.add(xref)
443: (20)                                fontname, ext, _, buffer = doc.extract_font(xref)
444: (20)                                if ext == "n/a" or not buffer:
445: (24)                                    continue
446: (20)                                outname = os.path.join(
447: (24)                                    out_dir, f"{fontname.replace(' ', '-')}-{xref}.{ext}"
448: (20)                                )
449: (20)                                with open(outname, "wb") as outfile:
450: (24)                                    outfile.write(buffer)
451: (20)                                buffer = None
452: (8)                    if args.images:
453: (12)                       itemlist = doc.get_page_images(pno - 1)
454: (12)                       for item in itemlist:
455: (16)                           xref = item[0]
456: (16)                           if xref not in image_xrefs:
457: (20)                               image_xrefs.add(xref)
458: (20)                               pix = recoverpix(doc, item)
459: (20)                               if type(pix) is dict:
460: (24)                                   ext = pix["ext"]
461: (24)                                   imgdata = pix["image"]
462: (24)                                   outname = os.path.join(out_dir, "img-%i.%s" % (xref,
ext))
463: (24)                                   with open(outname, "wb") as outfile:
464: (28)                                       outfile.write(imgdata)
465: (20)                               else:
466: (24)                                   outname = os.path.join(out_dir, "img-%i.png" % xref)
467: (24)                                   pix2 = (
468: (28)                                       pix
469: (28)                                       if pix.colorspace.n < 4
470: (28)                                       else pymupdf.Pixmap(pymupdf.csRGB, pix)
471: (24)                                   )
472: (24)                                   pix2.save(outname)
473: (4)                if args.fonts:
474: (8)                    pymupdf.message("saved %i fonts to '%s'" % (len(font_xrefs), out_dir))
475: (4)                if args.images:
476: (8)                    pymupdf.message("saved %i images to '%s'" % (len(image_xrefs),
out_dir))
477: (4)                doc.close()
478: (0)            def page_simple(page, textout, GRID, fontsize, noformfeed, skip_empty, flags):
479: (4)                eop = b"\n" if noformfeed else bytes([12])
480: (4)                text = page.get_text("text", flags=flags)
481: (4)                if not text:
482: (8)                    if not skip_empty:
483: (12)                       textout.write(eop)  # write formfeed
484: (8)                    return
485: (4)                textout.write(text.encode("utf8", errors="surrogatepass"))
486: (4)                textout.write(eop)
487: (4)                return
488: (0)            def page_blocksort(page, textout, GRID, fontsize, noformfeed, skip_empty,
flags):
489: (4)                eop = b"\n" if noformfeed else bytes([12])
490: (4)                blocks = page.get_text("blocks", flags=flags)
491: (4)                if blocks == []:
492: (8)                    if not skip_empty:
493: (12)                       textout.write(eop)  # write formfeed
494: (8)                    return
495: (4)                blocks.sort(key=lambda b: (b[3], b[0]))
496: (4)                for b in blocks:
497: (8)                    textout.write(b[4].encode("utf8", errors="surrogatepass"))
498: (4)                textout.write(eop)
499: (4)                return
500: (0)            def page_layout(page, textout, GRID, fontsize, noformfeed, skip_empty, flags):
501: (4)                eop = b"\n" if noformfeed else bytes([12])
502: (4)                def find_line_index(values: List[int], value: int) -> int:
503: (8)                    """Find the right row coordinate.
504: (8)                    Args:
505: (12)                       values: (list) y-coordinates of rows.
```

```
506: (12)                        value: (int) lookup for this value (y-origin of char).
507: (8)                    Returns:
508: (12)                        y-ccordinate of appropriate line for value.
509: (8)                    """
510: (8)                    i = bisect.bisect_right(values, value)
511: (8)                    if i:
512: (12)                        return values[i - 1]
513: (8)                    raise RuntimeError("Line for %g not found in %s" % (value, values))
514: (4)                def curate_rows(rows: Set[int], GRID) -> List:
515: (8)                    rows = list(rows)
516: (8)                    rows.sort()  # sort ascending
517: (8)                    nrows = [rows[0]]
518: (8)                    for h in rows[1:]:
519: (12)                       if h >= nrows[-1] + GRID:  # only keep significant differences
520: (16)                           nrows.append(h)
521: (8)                    return nrows  # curated list of line bottom coordinates
522: (4)                def process_blocks(blocks: List[Dict], page: pymupdf.Page):
523: (8)                    rows = set()
524: (8)                    page_width = page.rect.width
525: (8)                    page_height = page.rect.height
526: (8)                    rowheight = page_height
527: (8)                    left = page_width
528: (8)                    right = 0
529: (8)                    chars = []
530: (8)                    for block in blocks:
531: (12)                       for line in block["lines"]:
532: (16)                           if line["dir"] != (1, 0):  # ignore non-horizontal text
533: (20)                               continue
534: (16)                           x0, y0, x1, y1 = line["bbox"]
535: (16)                           if y1 < 0 or y0 > page.rect.height:  # ignore if outside
CropBox
536: (20)                               continue
537: (16)                           height = y1 - y0
538: (16)                           if rowheight > height:
539: (20)                               rowheight = height
540: (16)                           for span in line["spans"]:
541: (20)                               if span["size"] <= fontsize:
542: (24)                                   continue
543: (20)                               for c in span["chars"]:
544: (24)                                   x0, _, x1, _ = c["bbox"]
545: (24)                                   cwidth = x1 - x0
546: (24)                                   ox, oy = c["origin"]
547: (24)                                   oy = int(round(oy))
548: (24)                                   rows.add(oy)
549: (24)                                   ch = c["c"]
550: (24)                                   if left > ox and ch != " ":
551: (28)                                       left = ox  # update left coordinate
552: (24)                                   if right < x1:
553: (28)                                       right = x1  # update right coordinate
554: (24)                                   if cwidth == 0 and chars != []:  # potential ligature
555: (28)                                       old_ch, old_ox, old_oy, old_cwidth = chars[-1]
556: (28)                                       if old_oy == oy:  # ligature
557: (32)                                           if old_ch != chr(0xFB00):  # previous "ff"
char lig?
558: (36)                                               lig = joinligature(old_ch + ch)  # no
559: (32)                                           elif ch == "i":
560: (36)                                               lig = chr(0xFB03)  # "ffi"
561: (32)                                           elif ch == "l":
562: (36)                                               lig = chr(0xFB04)  # "ffl"
563: (32)                                           else:  # something wrong, leave old char in
place
564: (36)                                               lig = old_ch
565: (32)                                           chars[-1] = (lig, old_ox, old_oy, old_cwidth)
566: (32)                                           continue
567: (24)                                   chars.append((ch, ox, oy, cwidth))  # all chars on
page
568: (8)                    return chars, rows, left, right, rowheight
569: (4)                def joinligature(lig: str) -> str:
570: (8)                    """Return ligature character for a given pair / triple of characters.
```

```
571: (8)                    Args:
572: (12)                       lig: (str) 2/3 characters, e.g. "ff"
573: (8)                    Returns:
574: (12)                       Ligature, e.g. "ff" -> chr(0xFB00)
575: (8)                    """
576: (8)                    if lig == "ff":
577: (12)                       return chr(0xFB00)
578: (8)                    elif lig == "fi":
579: (12)                       return chr(0xFB01)
580: (8)                    elif lig == "fl":
581: (12)                       return chr(0xFB02)
582: (8)                    elif lig == "ffi":
583: (12)                       return chr(0xFB03)
584: (8)                    elif lig == "ffl":
585: (12)                       return chr(0xFB04)
586: (8)                    elif lig == "ft":
587: (12)                       return chr(0xFB05)
588: (8)                    elif lig == "st":
589: (12)                       return chr(0xFB06)
590: (8)                    return lig
591: (4)                def make_textline(left, slot, minslot, lchars):
592: (8)                    """Produce the text of one output line.
593: (8)                    Args:
594: (12)                       left: (float) left most coordinate used on page
595: (12)                       slot: (float) avg width of one character in any font in use.
596: (12)                       minslot: (float) min width for the characters in this line.
597: (12)                       chars: (list[tuple]) characters of this line.
598: (8)                    Returns:
599: (12)                       text: (str) text string for this line
600: (8)                    """
601: (8)                    text = ""  # we output this
602: (8)                    old_char = ""
603: (8)                    old_x1 = 0  # end coordinate of last char
604: (8)                    old_ox = 0  # x-origin of last char
605: (8)                    if minslot <= pymupdf.EPSILON:
606: (12)                       raise RuntimeError("program error: minslot too small = %g" %
minslot)
607: (8)                    for c in lchars:  # loop over characters
608: (12)                       char, ox, _, cwidth = c
609: (12)                       ox = ox - left  # its (relative) start coordinate
610: (12)                       x1 = ox + cwidth  # ending coordinate
611: (12)                       if old_char == char and ox - old_ox <= cwidth * 0.2:
612: (16)                           continue
613: (12)                       if char == " " and (old_x1 - ox) / cwidth > 0.8:
614: (16)                           continue
615: (12)                       old_char = char
616: (12)                       if ox < old_x1 + minslot:  # assume char adjacent to previous
617: (16)                           text += char  # append to output
618: (16)                           old_x1 = x1  # new end coord
619: (16)                           old_ox = ox  # new origin.x
620: (16)                           continue
621: (12)                       if char == " ":  # rest relevant for non-space only
622: (16)                           continue
623: (12)                       delta = int(ox / slot) - len(text)
624: (12)                       if ox > old_x1 and delta > 1:
625: (16)                           text += " " * delta
626: (12)                       text += char
627: (12)                       old_x1 = x1  # new end coordinate
628: (12)                       old_ox = ox  # new origin
629: (8)                    return text.rstrip()
630: (4)                blocks = page.get_text("rawdict", flags=flags)["blocks"]
631: (4)                chars, rows, left, right, rowheight = process_blocks(blocks, page)
632: (4)                if chars == []:
633: (8)                    if not skip_empty:
634: (12)                       textout.write(eop)  # write formfeed
635: (8)                    return
636: (4)                rows = curate_rows(rows, GRID)
637: (4)                chars.sort(key=lambda c: c[1])
638: (4)                lines = {}  # key: y1-ccordinate, value: char list
```

```
639: (4)                for c in chars:
640: (8)                    _, _, oy, _ = c
641: (8)                    y = find_line_index(rows, oy)  # y-coord of the right line
642: (8)                    lchars = lines.get(y, [])  # read line chars so far
643: (8)                    lchars.append(c)  # append this char
644: (8)                    lines[y] = lchars  # write back to line
645: (4)                keys = list(lines.keys())
646: (4)                keys.sort()
647: (4)                slot = right - left
648: (4)                minslots = {}
649: (4)                for k in keys:
650: (8)                    lchars = lines[k]
651: (8)                    ccount = len(lchars)
652: (8)                    if ccount < 2:
653: (12)                       minslots[k] = 1
654: (12)                       continue
655: (8)                    widths = [c[3] for c in lchars]
656: (8)                    widths.sort()
657: (8)                    this_slot = statistics.median(widths)  # take median value
658: (8)                    if this_slot < slot:
659: (12)                       slot = this_slot
660: (8)                    minslots[k] = widths[0]
661: (4)                rowheight = rowheight * (rows[-1] - rows[0]) / (rowheight * len(rows)) *
1.2
662: (4)                rowpos = rows[0]  # first line positioned here
663: (4)                textout.write(b"\n")
664: (4)                for k in keys:  # walk through the lines
665: (8)                    while rowpos < k:  # honor distance between lines
666: (12)                       textout.write(b"\n")
667: (12)                       rowpos += rowheight
668: (8)                    text = make_textline(left, slot, minslots[k], lines[k])
669: (8)                    textout.write((text + "\n").encode("utf8", errors="surrogatepass"))
670: (8)                    rowpos = k + rowheight
671: (4)                textout.write(eop)  # write formfeed
672: (0)            def gettext(args):
673: (4)                doc = open_file(args.input, args.password, pdf=False)
674: (4)                pagel = get_list(args.pages, doc.page_count + 1)
675: (4)                output = args.output
676: (4)                if output is None:
677: (8)                    filename, _ = os.path.splitext(doc.name)
678: (8)                    output = filename + ".txt"
679: (4)                with open(output, "wb") as textout:
680: (8)                    flags = pymupdf.TEXT_PRESERVE_LIGATURES |
pymupdf.TEXT_PRESERVE_WHITESPACE
681: (8)                    if args.convert_white:
682: (12)                       flags ^= pymupdf.TEXT_PRESERVE_WHITESPACE
683: (8)                    if args.noligatures:
684: (12)                       flags ^= pymupdf.TEXT_PRESERVE_LIGATURES
685: (8)                    if args.extra_spaces:
686: (12)                       flags ^= pymupdf.TEXT_INHIBIT_SPACES
687: (8)                    func = {
688: (12)                       "simple": page_simple,
689: (12)                       "blocks": page_blocksort,
690: (12)                       "layout": page_layout,
691: (8)                    }
692: (8)                    for pno in pagel:
693: (12)                       page = doc[pno - 1]
694: (12)                       func[args.mode](
695: (16)                           page,
696: (16)                           textout,
697: (16)                           args.grid,
698: (16)                           args.fontsize,
699: (16)                           args.noformfeed,
700: (16)                           args.skip_empty,
701: (16)                           flags=flags,
702: (12)                       )
703: (0)            def _internal(args):
704: (4)                pymupdf.message('This is from PyMuPDF message().')
705: (4)                pymupdf.log('This is from PyMuPDF log().')
```

```
(0)             def main():
(4)                 """Define command configurations."""
(4)                 parser = argparse.ArgumentParser(
(8)                     prog="pymupdf",
(8)                     description=mycenter("Basic PyMuPDF Functions"),
(4)                 )
(4)                 subps = parser.add_subparsers(
(8)                     title="Subcommands", help="Enter 'command -h' for subcommand specific help"
(4)                 )
(4)                 ps_show = subps.add_parser("show", description=mycenter("display PDF information"))
(4)                 ps_show.add_argument("input", type=str, help="PDF filename")
(4)                 ps_show.add_argument("-password", help="password")
(4)                 ps_show.add_argument("-catalog", action="store_true", help="show PDF catalog")
(4)                 ps_show.add_argument("-trailer", action="store_true", help="show PDF trailer")
(4)                 ps_show.add_argument("-metadata", action="store_true", help="show PDF metadata")
(4)                 ps_show.add_argument(
(8)                     "-xrefs", type=str, help="show selected objects, format: 1,5-7,N"
(4)                 )
(4)                 ps_show.add_argument(
(8)                     "-pages", type=str, help="show selected pages, format: 1,5-7,50-N"
(4)                 )
(4)                 ps_show.set_defaults(func=show)
(4)                 ps_clean = subps.add_parser(
(8)                     "clean", description=mycenter("optimize PDF, or create sub-PDF if pages given")
(4)                 )
(4)                 ps_clean.add_argument("input", type=str, help="PDF filename")
(4)                 ps_clean.add_argument("output", type=str, help="output PDF filename")
(4)                 ps_clean.add_argument("-password", help="password")
(4)                 ps_clean.add_argument(
(8)                     "-encryption",
(8)                     help="encryption method",
(8)                     choices=("keep", "none", "rc4-40", "rc4-128", "aes-128", "aes-256"),
(8)                     default="none",
(4)                 )
(4)                 ps_clean.add_argument("-owner", type=str, help="owner password")
(4)                 ps_clean.add_argument("-user", type=str, help="user password")
(4)                 ps_clean.add_argument(
(8)                     "-garbage",
(8)                     type=int,
(8)                     help="garbage collection level",
(8)                     choices=range(5),
(8)                     default=0,
(4)                 )
(4)                 ps_clean.add_argument(
(8)                     "-compress",
(8)                     action="store_true",
(8)                     default=False,
(8)                     help="compress (deflate) output",
(4)                 )
(4)                 ps_clean.add_argument(
(8)                     "-ascii", action="store_true", default=False, help="ASCII encode binary data"
(4)                 )
(4)                 ps_clean.add_argument(
(8)                     "-linear",
(8)                     action="store_true",
(8)                     default=False,
(8)                     help="format for fast web display",
(4)                 )
(4)                 ps_clean.add_argument(
(8)                     "-permission", type=int, default=-1, help="integer with permission levels"
(4)                 )
```

```
767: (4)                ps_clean.add_argument(
768: (8)                    "-sanitize",
769: (8)                    action="store_true",
770: (8)                    default=False,
771: (8)                    help="sanitize / clean contents",
772: (4)                )
773: (4)                ps_clean.add_argument(
774: (8)                    "-pretty", action="store_true", default=False, help="prettify PDF
structure"
775: (4)                )
776: (4)                ps_clean.add_argument(
777: (8)                    "-pages", help="output selected pages pages, format: 1,5-7,50-N"
778: (4)                )
779: (4)                ps_clean.set_defaults(func=clean)
780: (4)                ps_join = subps.add_parser(
781: (8)                    "join",
782: (8)                    description=mycenter("join PDF documents"),
783: (8)                    epilog="specify each input as 'filename[,password[,pages]]'",
784: (4)                )
785: (4)                ps_join.add_argument("input", nargs="*", help="input filenames")
786: (4)                ps_join.add_argument("-output", required=True, help="output filename")
787: (4)                ps_join.set_defaults(func=doc_join)
788: (4)                ps_extract = subps.add_parser(
789: (8)                    "extract", description=mycenter("extract images and fonts to disk")
790: (4)                )
791: (4)                ps_extract.add_argument("input", type=str, help="PDF filename")
792: (4)                ps_extract.add_argument("-images", action="store_true", help="extract
images")
793: (4)                ps_extract.add_argument("-fonts", action="store_true", help="extract
fonts")
794: (4)                ps_extract.add_argument(
795: (8)                    "-output", help="folder to receive output, defaults to current"
796: (4)                )
797: (4)                ps_extract.add_argument("-password", help="password")
798: (4)                ps_extract.add_argument(
799: (8)                    "-pages", type=str, help="consider these pages only, format: 1,5-7,50-
N"
800: (4)                )
801: (4)                ps_extract.set_defaults(func=extract_objects)
802: (4)                ps_show = subps.add_parser(
803: (8)                    "embed-info", description=mycenter("list embedded files")
804: (4)                )
805: (4)                ps_show.add_argument("input", help="PDF filename")
806: (4)                ps_show.add_argument("-name", help="if given, report only this one")
807: (4)                ps_show.add_argument("-detail", action="store_true", help="detail
information")
808: (4)                ps_show.add_argument("-password", help="password")
809: (4)                ps_show.set_defaults(func=embedded_list)
810: (4)                ps_embed_add = subps.add_parser(
811: (8)                    "embed-add", description=mycenter("add embedded file")
812: (4)                )
813: (4)                ps_embed_add.add_argument("input", help="PDF filename")
814: (4)                ps_embed_add.add_argument("-password", help="password")
815: (4)                ps_embed_add.add_argument(
816: (8)                    "-output", help="output PDF filename, incremental save if none"
817: (4)                )
818: (4)                ps_embed_add.add_argument("-name", required=True, help="name of new
entry")
819: (4)                ps_embed_add.add_argument("-path", required=True, help="path to data for
new entry")
820: (4)                ps_embed_add.add_argument("-desc", help="description of new entry")
821: (4)                ps_embed_add.set_defaults(func=embedded_add)
822: (4)                ps_embed_del = subps.add_parser(
823: (8)                    "embed-del", description=mycenter("delete embedded file")
824: (4)                )
825: (4)                ps_embed_del.add_argument("input", help="PDF filename")
826: (4)                ps_embed_del.add_argument("-password", help="password")
827: (4)                ps_embed_del.add_argument(
828: (8)                    "-output", help="output PDF filename, incremental save if none"
```

```
829: (4)                )
830: (4)                ps_embed_del.add_argument("-name", required=True, help="name of entry to
delete")
831: (4)                ps_embed_del.set_defaults(func=embedded_del)
832: (4)                ps_embed_upd = subps.add_parser(
833: (8)                    "embed-upd",
834: (8)                    description=mycenter("update embedded file"),
835: (8)                    epilog="except '-name' all parameters are optional",
836: (4)                )
837: (4)                ps_embed_upd.add_argument("input", help="PDF filename")
838: (4)                ps_embed_upd.add_argument("-name", required=True, help="name of entry")
839: (4)                ps_embed_upd.add_argument("-password", help="password")
840: (4)                ps_embed_upd.add_argument(
841: (8)                    "-output", help="Output PDF filename, incremental save if none"
842: (4)                )
843: (4)                ps_embed_upd.add_argument("-path", help="path to new data for entry")
844: (4)                ps_embed_upd.add_argument("-filename", help="new filename to store in
entry")
845: (4)                ps_embed_upd.add_argument(
846: (8)                    "-ufilename", help="new unicode filename to store in entry"
847: (4)                )
848: (4)                ps_embed_upd.add_argument("-desc", help="new description to store in
entry")
849: (4)                ps_embed_upd.set_defaults(func=embedded_upd)
850: (4)                ps_embed_extract = subps.add_parser(
851: (8)                    "embed-extract", description=mycenter("extract embedded file to disk")
852: (4)                )
853: (4)                ps_embed_extract.add_argument("input", type=str, help="PDF filename")
854: (4)                ps_embed_extract.add_argument("-name", required=True, help="name of
entry")
855: (4)                ps_embed_extract.add_argument("-password", help="password")
856: (4)                ps_embed_extract.add_argument(
857: (8)                    "-output", help="output filename, default is stored name"
858: (4)                )
859: (4)                ps_embed_extract.set_defaults(func=embedded_get)
860: (4)                ps_embed_copy = subps.add_parser(
861: (8)                    "embed-copy", description=mycenter("copy embedded files between PDFs")
862: (4)                )
863: (4)                ps_embed_copy.add_argument("input", type=str, help="PDF to receive
embedded files")
864: (4)                ps_embed_copy.add_argument("-password", help="password of input")
865: (4)                ps_embed_copy.add_argument(
866: (8)                    "-output", help="output PDF, incremental save to 'input' if omitted"
867: (4)                )
868: (4)                ps_embed_copy.add_argument(
869: (8)                    "-source", required=True, help="copy embedded files from here"
870: (4)                )
871: (4)                ps_embed_copy.add_argument("-pwdsource", help="password of 'source' PDF")
872: (4)                ps_embed_copy.add_argument(
873: (8)                    "-name", nargs="*", help="restrict copy to these entries"
874: (4)                )
875: (4)                ps_embed_copy.set_defaults(func=embedded_copy)
876: (4)                ps_gettext = subps.add_parser(
877: (8)                    "gettext", description=mycenter("extract text in various formatting
modes")
878: (4)                )
879: (4)                ps_gettext.add_argument("input", type=str, help="input document filename")
880: (4)                ps_gettext.add_argument("-password", help="password for input document")
881: (4)                ps_gettext.add_argument(
882: (8)                    "-mode",
883: (8)                    type=str,
884: (8)                    help="mode: simple, block sort, or layout (default)",
885: (8)                    choices=("simple", "blocks", "layout"),
886: (8)                    default="layout",
887: (4)                )
888: (4)                ps_gettext.add_argument(
889: (8)                    "-pages",
890: (8)                    type=str,
891: (8)                    help="select pages, format: 1,5-7,50-N",
```

```
892: (8)                    default="1-N",
893: (4)                )
894: (4)                ps_gettext.add_argument(
895: (8)                    "-noligatures",
896: (8)                    action="store_true",
897: (8)                    help="expand ligature characters (default False)",
898: (8)                    default=False,
899: (4)                )
900: (4)                ps_gettext.add_argument(
901: (8)                    "-convert-white",
902: (8)                    action="store_true",
903: (8)                    help="convert whitespace characters to white (default False)",
904: (8)                    default=False,
905: (4)                )
906: (4)                ps_gettext.add_argument(
907: (8)                    "-extra-spaces",
908: (8)                    action="store_true",
909: (8)                    help="fill gaps with spaces (default False)",
910: (8)                    default=False,
911: (4)                )
912: (4)                ps_gettext.add_argument(
913: (8)                    "-noformfeed",
914: (8)                    action="store_true",
915: (8)                    help="write linefeeds, no formfeeds (default False)",
916: (8)                    default=False,
917: (4)                )
918: (4)                ps_gettext.add_argument(
919: (8)                    "-skip-empty",
920: (8)                    action="store_true",
921: (8)                    help="suppress pages with no text (default False)",
922: (8)                    default=False,
923: (4)                )
924: (4)                ps_gettext.add_argument(
925: (8)                    "-output",
926: (8)                    help="store text in this file (default inputfilename.txt)",
927: (4)                )
928: (4)                ps_gettext.add_argument(
929: (8)                    "-grid",
930: (8)                    type=float,
931: (8)                    help="merge lines if closer than this (default 2)",
932: (8)                    default=2,
933: (4)                )
934: (4)                ps_gettext.add_argument(
935: (8)                    "-fontsize",
936: (8)                    type=float,
937: (8)                    help="only include text with a larger fontsize (default 3)",
938: (8)                    default=3,
939: (4)                )
940: (4)                ps_gettext.set_defaults(func=gettext)
941: (4)                ps_internal = subps.add_parser(
942: (8)                    "internal", description=mycenter("internal testing")
943: (4)                )
944: (4)                ps_internal.set_defaults(func=_internal)
945: (4)                args = parser.parse_args()  # create parameter arguments class
946: (4)                if not hasattr(args, "func"):  # no function selected
947: (8)                    parser.print_help()  # so print top level help
948: (4)                else:
949: (8)                    args.func(args)  # execute requested command
950: (0)            if __name__ == "__main__":
951: (4)                main()
```

File 10 - withheadingssummslinesnumbers_clubbing_files_fileloggerpythons.py:

```
1: (0)              import os
2: (0)              from datetime import datetime
3: (0)              from collections import Counter
4: (0)              def get_file_info(root_folder):
```

```
5: (4)                  folder_counter = 0
6: (4)                  file_counter = 0
7: (4)                  extension_counter = {}
8: (4)                  folder_sizes = {}
9: (4)                  file_info_list = []
10: (4)                 readable_extensions = ['.txt', '.py', '.cs', '.cpp', '.c', '.h', '.pyi',
'.java', '.ini', '.cfg', '.xml']
11: (4)                 for root, dirs, files in os.walk(root_folder):
12: (8)                     folder_counter += 1
13: (8)                     folder_size = 0
14: (8)                     for file in files:
15: (12)                        try:
16: (16)                            file_counter += 1
17: (16)                            file_path = os.path.join(root, file)
18: (16)                            file_size = os.path.getsize(file_path)
19: (16)                            folder_size += file_size
20: (16)                            file_extension = os.path.splitext(file)[1].lower()
21: (16)                            if file_extension not in extension_counter:
22: (20)                                extension_counter[file_extension] = 0
23: (16)                            extension_counter[file_extension] += 1
24: (16)                            creation_time =
datetime.fromtimestamp(os.path.getctime(file_path))
25: (16)                            modified_time =
datetime.fromtimestamp(os.path.getmtime(file_path))
26: (16)                            accessed_time =
datetime.fromtimestamp(os.path.getatime(file_path))
27: (16)                            hours_unaccessed = (datetime.now() -
accessed_time).total_seconds() / 3600
28: (16)                            file_info_list.append([
29: (20)                                folder_counter,
30: (20)                                file_counter,
31: (20)                                file_extension,
32: (20)                                folder_size,
33: (20)                                file_size,
34: (20)                                creation_time,
35: (20)                                modified_time,
36: (20)                                accessed_time,
37: (20)                                hours_unaccessed,
38: (20)                                root,
39: (20)                                file
40: (16)                            ])
41: (16)                            if file_extension in readable_extensions:
42: (20)                                try:
43: (24)                                    with open(file_path, 'r', encoding='utf-8',
errors='ignore') as f:
44: (28)                                        content = f.read()
45: (28)                                        line_count = content.count('\n') + 1
46: (28)                                        content_with_line_numbers = '\n'.join(f"{i+1}:
{line}" for i, line in enumerate(content.split('\n')))
47: (28)                                        file_info_list.append([
48: (32)                                            'File Content:',
49: (32)                                            content_with_line_numbers,
50: (32)                                            'Line Count:',
51: (32)                                            line_count
52: (28)                                        ])
53: (20)                                except Exception as e:
54: (24)                                    print(f"Error reading file {file_path}: {e}")
55: (12)                        except Exception as e:
56: (16)                            print(f"Error processing file {file}: {e}")
57: (8)                     folder_sizes[root] = folder_size
58: (4)                 return folder_counter, file_counter, extension_counter, folder_sizes,
file_info_list
59: (0)             def write_file_info_to_log(file_info_list, output_file):
60: (4)                 try:
61: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
62: (12)                        logfile.write('Folder Counter###File Counter###File
Extension###Folder Size (bytes)###File Size (bytes)###Creation Time###Modified Time###Accessed
Time###Hours Unaccessed###Folder Path###File Name\n')
63: (12)                        for info in file_info_list:
```

```
64: (16)                            logfile.write('###'.join(map(str, info)) + '\n')
65: (4)                 except Exception as e:
66: (8)                     print(f"Error writing to log file {output_file}: {e}")
67: (0)             def write_file_summary_to_log(file_info_list, output_file):
68: (4)                 try:
69: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
70: (12)                        logfile.write('Folder Counter###File Counter###File
Extension###Folder Size (bytes)###File Size (bytes)###Creation Time###Modified Time###Accessed
Time###Hours Unaccessed###Folder Path###File Name\n')
71: (12)                        for info in file_info_list:
72: (16)                            if isinstance(info[0], int):  # Only write the summary lines
(not the content lines)
73: (20)                                logfile.write('###'.join(map(str, info)) + '\n')
74: (4)                 except Exception as e:
75: (8)                     print(f"Error writing to summary log file {output_file}: {e}")
76: (0)             def write_extension_size_distribution(file_info_list, output_file):
77: (4)                 try:
78: (8)                     extension_size = {}
79: (8)                     for info in file_info_list:
80: (12)                        if isinstance(info[0], int):  # Only process the summary lines
81: (16)                            extension = info[2]
82: (16)                            size = info[4]
83: (16)                            if extension not in extension_size:
84: (20)                                extension_size[extension] = 0
85: (16)                            extension_size[extension] += size
86: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
87: (12)                        logfile.write('Extension###Size (bytes)\n')
88: (12)                        for ext, size in extension_size.items():
89: (16)                            logfile.write(f"{ext}###{size}\n")
90: (4)                 except Exception as e:
91: (8)                     print(f"Error writing to extension size distribution log file
{output_file}: {e}")
92: (0)             def write_keyword_frequency(file_info_list, output_file):
93: (4)                 try:
94: (8)                     keyword_counter = Counter()
95: (8)                     for info in file_info_list:
96: (12)                        if isinstance(info[0], list) and info[0] == 'File Content:':
97: (16)                            content = info[1]
98: (16)                            words = content.split()
99: (16)                            keyword_counter.update(words)
100: (8)                    with open(output_file, 'w', encoding='utf-8') as logfile:
101: (12)                       logfile.write('Keyword###Frequency\n')
102: (12)                       for word, freq in keyword_counter.items():
103: (16)                           logfile.write(f"{word}###{freq}\n")
104: (4)                except Exception as e:
105: (8)                    print(f"Error writing to keyword frequency log file {output_file}:
{e}")
106: (0)            root_folder_path = '.'  # Change this to the desired root folder path
107: (0)            timestamp = datetime.now().strftime('%Y%m%d%H%M%S')
108: (0)            base_folder_name = os.path.basename(os.path.abspath(root_folder_path))
109: (0)            output_log_file =
f'{base_folder_name}_file_info_log_{timestamp}.saan_file_log'
110: (0)            output_summary_file =
f'{base_folder_name}_file_summary_{timestamp}.saan_file_log'
111: (0)            output_extension_size_file =
f'{base_folder_name}_extension_size_{timestamp}.saan_file_log'
112: (0)            output_keyword_file =
f'{base_folder_name}_keyword_frequency_{timestamp}.saan_file_log'
113: (0)            try:
114: (4)                folder_counter, file_counter, extension_counter, folder_sizes,
file_info_list = get_file_info(root_folder_path)
115: (4)                write_file_info_to_log(file_info_list, output_log_file)
116: (4)                write_file_summary_to_log(file_info_list, output_summary_file)
117: (4)                write_extension_size_distribution(file_info_list,
output_extension_size_file)
118: (4)                write_keyword_frequency(file_info_list, output_keyword_file)
119: (4)                print(f"File info logged to {output_log_file}")
120: (4)                print(f"File summary logged to {output_summary_file}")
121: (4)                print(f"Extension size distribution logged to
```

```
{output_extension_size_file}")
122: (4)                print(f"Keyword frequency logged to {output_keyword_file}")
123: (0)            except Exception as e:
124: (4)                print(f"Error during processing: {e}")
```

File 11 - withheadingssummslinesnumbers_clubbing_files_fileloggerpythons.py:

```
1: (0)              import os
2: (0)              from datetime import datetime
3: (0)              from collections import Counter
4: (0)              def get_file_info(root_folder):
5: (4)                  folder_counter = 0
6: (4)                  file_counter = 0
7: (4)                  extension_counter = {}
8: (4)                  folder_sizes = {}
9: (4)                  file_info_list = []
10: (4)                 readable_extensions = ['.txt', '.py', '.cs', '.cpp', '.c', '.h', '.pyi',
'.java', '.ini', '.cfg', '.xml']
11: (4)                 for root, dirs, files in os.walk(root_folder):
12: (8)                     folder_counter += 1
13: (8)                     folder_size = 0
14: (8)                     for file in files:
15: (12)                        try:
16: (16)                            file_counter += 1
17: (16)                            file_path = os.path.join(root, file)
18: (16)                            file_size = os.path.getsize(file_path)
19: (16)                            folder_size += file_size
20: (16)                            file_extension = os.path.splitext(file)[1].lower()
21: (16)                            if file_extension not in extension_counter:
22: (20)                                extension_counter[file_extension] = 0
23: (16)                            extension_counter[file_extension] += 1
24: (16)                            creation_time =
datetime.fromtimestamp(os.path.getctime(file_path))
25: (16)                            modified_time =
datetime.fromtimestamp(os.path.getmtime(file_path))
26: (16)                            accessed_time =
datetime.fromtimestamp(os.path.getatime(file_path))
27: (16)                            hours_unaccessed = (datetime.now() -
accessed_time).total_seconds() / 3600
28: (16)                            file_info_list.append([
29: (20)                                folder_counter,
30: (20)                                file_counter,
31: (20)                                file_extension,
32: (20)                                folder_size,
33: (20)                                file_size,
34: (20)                                creation_time,
35: (20)                                modified_time,
36: (20)                                accessed_time,
37: (20)                                hours_unaccessed,
38: (20)                                root,
39: (20)                                file
40: (16)                            ])
41: (16)                            if file_extension in readable_extensions:
42: (20)                                try:
43: (24)                                    with open(file_path, 'r', encoding='utf-8',
errors='ignore') as f:
44: (28)                                        content = f.read()
45: (28)                                        line_count = content.count('\n') + 1
46: (28)                                        content_with_line_numbers = '\n'.join(f"{i+1}:
{line}" for i, line in enumerate(content.split('\n')))
47: (28)                                        file_info_list.append([
48: (32)                                            'File Content:',
49: (32)                                            content_with_line_numbers,
50: (32)                                            'Line Count:',
51: (32)                                            line_count
52: (28)                                        ])
53: (20)                                except Exception as e:
```

```
54: (24)                                    print(f"Error reading file {file_path}: {e}")
55: (12)                        except Exception as e:
56: (16)                            print(f"Error processing file {file}: {e}")
57: (8)                     folder_sizes[root] = folder_size
58: (4)                 return folder_counter, file_counter, extension_counter, folder_sizes,
file_info_list
59: (0)             def write_file_info_to_log(file_info_list, output_file):
60: (4)                 try:
61: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
62: (12)                        logfile.write('Folder Counter###File Counter###File
Extension###Folder Size (bytes)###File Size (bytes)###Creation Time###Modified Time###Accessed
Time###Hours Unaccessed###Folder Path###File Name\n')
63: (12)                        for info in file_info_list:
64: (16)                            logfile.write('###'.join(map(str, info)) + '\n')
65: (4)                 except Exception as e:
66: (8)                     print(f"Error writing to log file {output_file}: {e}")
67: (0)             def write_file_summary_to_log(file_info_list, output_file):
68: (4)                 try:
69: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
70: (12)                        logfile.write('Folder Counter###File Counter###File
Extension###Folder Size (bytes)###File Size (bytes)###Creation Time###Modified Time###Accessed
Time###Hours Unaccessed###Folder Path###File Name\n')
71: (12)                        for info in file_info_list:
72: (16)                            if isinstance(info[0], int):  # Only write the summary lines
(not the content lines)
73: (20)                                logfile.write('###'.join(map(str, info)) + '\n')
74: (4)                 except Exception as e:
75: (8)                     print(f"Error writing to summary log file {output_file}: {e}")
76: (0)             def write_extension_size_distribution(file_info_list, output_file):
77: (4)                 try:
78: (8)                     extension_size = {}
79: (8)                     for info in file_info_list:
80: (12)                        if isinstance(info[0], int):  # Only process the summary lines
81: (16)                            extension = info[2]
82: (16)                            size = info[4]
83: (16)                            if extension not in extension_size:
84: (20)                                extension_size[extension] = 0
85: (16)                            extension_size[extension] += size
86: (8)                     with open(output_file, 'w', encoding='utf-8') as logfile:
87: (12)                        logfile.write('Extension###Size (bytes)\n')
88: (12)                        for ext, size in extension_size.items():
89: (16)                            logfile.write(f"{ext}###{size}\n")
90: (4)                 except Exception as e:
91: (8)                     print(f"Error writing to extension size distribution log file
{output_file}: {e}")
92: (0)             def write_keyword_frequency(file_info_list, output_file):
93: (4)                 try:
94: (8)                     keyword_counter = Counter()
95: (8)                     for info in file_info_list:
96: (12)                        if isinstance(info[0], list) and info[0] == 'File Content:':
97: (16)                            content = info[1]
98: (16)                            words = content.split()
99: (16)                            keyword_counter.update(words)
100: (8)                    with open(output_file, 'w', encoding='utf-8') as logfile:
101: (12)                       logfile.write('Keyword###Frequency\n')
102: (12)                       for word, freq in keyword_counter.items():
103: (16)                           logfile.write(f"{word}###{freq}\n")
104: (4)                 except Exception as e:
105: (8)                    print(f"Error writing to keyword frequency log file {output_file}:
{e}")
106: (0)            root_folder_path = '.'  # Change this to the desired root folder path
107: (0)            timestamp = datetime.now().strftime('%Y%m%d%H%M%S')
108: (0)            base_folder_name = os.path.basename(os.path.abspath(root_folder_path))
109: (0)            output_log_file =
f'{base_folder_name}_file_info_log_{timestamp}.saan_file_log'
110: (0)            output_summary_file =
f'{base_folder_name}_file_summary_{timestamp}.saan_file_log'
111: (0)            output_extension_size_file =
f'{base_folder_name}_extension_size_{timestamp}.saan_file_log'
```

```
112: (0)            output_keyword_file =
f'{base_folder_name}_keyword_frequency_{timestamp}.saan_file_log'
113: (0)            try:
114: (4)                folder_counter, file_counter, extension_counter, folder_sizes,
file_info_list = get_file_info(root_folder_path)
115: (4)                write_file_info_to_log(file_info_list, output_log_file)
116: (4)                write_file_summary_to_log(file_info_list, output_summary_file)
117: (4)                write_extension_size_distribution(file_info_list,
output_extension_size_file)
118: (4)                write_keyword_frequency(file_info_list, output_keyword_file)
119: (4)                print(f"File info logged to {output_log_file}")
120: (4)                print(f"File summary logged to {output_summary_file}")
121: (4)                print(f"Extension size distribution logged to
{output_extension_size_file}")
122: (4)                print(f"Keyword frequency logged to {output_keyword_file}")
123: (0)            except Exception as e:
124: (4)                print(f"Error during processing: {e}")
```

File 12 - SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRYCOMBINER_aligner_20_characters_for_pythons_codes.py:

```
1: (0)              import os
2: (0)              from datetime import datetime
3: (0)              def get_file_info(root_folder):
4: (4)                  file_info_list = []
5: (4)                  for root, dirs, files in os.walk(root_folder):
6: (8)                      for file in files:
7: (12)                         try:
8: (16)                             if file.endswith('.py'):
9: (20)                                 file_path = os.path.join(root, file)
10: (20)                                creation_time =
datetime.fromtimestamp(os.path.getctime(file_path))
11: (20)                                modified_time =
datetime.fromtimestamp(os.path.getmtime(file_path))
12: (20)                                file_extension = os.path.splitext(file)[1].lower()
13: (20)                                file_info_list.append([file, file_path, creation_time,
modified_time, file_extension, root])
14: (12)                        except Exception as e:
15: (16)                            print(f"Error processing file {file}: {e}")
16: (4)                 file_info_list.sort(key=lambda x: (x[2], x[3], len(x[0]), x[4]))  # Sort
by creation, modification time, name length, extension
17: (4)                 return file_info_list
18: (0)             def process_file(file_info_list):
19: (4)                 combined_output = []
20: (4)                 for idx, (file_name, file_path, creation_time, modified_time,
file_extension, root) in enumerate(file_info_list):
21: (8)                     with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
22: (12)                        content = f.read()
23: (12)                        content = "\n".join([line for line in content.split('\n') if
line.strip() and not line.strip().startswith("#")])
24: (12)                        content = content.replace('\t', '    ')
25: (12)                        processed_lines = []
26: (12)                        for i, line in enumerate(content.split('\n')):
27: (16)                            leading_spaces = len(line) - len(line.lstrip(' '))
28: (16)                            line_number_str = f"{i+1}: ({leading_spaces})"
29: (16)                            padding = ' ' * (20 - len(line_number_str))
30: (16)                            processed_line = f"{line_number_str}{padding}{line}"
31: (16)                            processed_lines.append(processed_line)
32: (12)                        content_with_line_numbers = "\n".join(processed_lines)
33: (12)                        combined_output.append(f"File {idx + 1} - {file_name}:\n")
34: (12)                        combined_output.append(content_with_line_numbers)
35: (12)                        combined_output.append("\n" + "-"*40 + "\n")
36: (4)                 return combined_output
37: (0)             root_folder_path = '.'  # Set this to the desired folder
38: (0)             file_info_list = get_file_info(root_folder_path)
39: (0)             combined_output = process_file(file_info_list)
```

```
40: (0)             output_file =
'SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRY_combined_python_files_20_chars.txt'
41: (0)             with open(output_file, 'w', encoding='utf-8') as logfile:
42: (4)                 logfile.write("\n".join(combined_output))
43: (0)             print(f"Processed file info logged to {output_file}")

----------------------------------------
```