File 1 - _3d.py:

```
(0)             from openpyxl.descriptors import Typed, Alias
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors.nested import (
(4)                 NestedBool,
(4)                 NestedInteger,
(4)                 NestedMinMax,
(0)             )
(0)             from openpyxl.descriptors.excel import ExtensionList
(0)             from .marker import PictureOptions
(0)             from .shapes import GraphicalProperties
(0)             class View3D(Serialisable):
(4)                 tagname = "view3D"
(4)                 rotX = NestedMinMax(min=-90, max=90, allow_none=True)
(4)                 x_rotation = Alias('rotX')
(4)                 hPercent = NestedMinMax(min=5, max=500, allow_none=True)
(4)                 height_percent = Alias('hPercent')
(4)                 rotY = NestedInteger(min=-90, max=90, allow_none=True)
(4)                 y_rotation = Alias('rotY')
(4)                 depthPercent = NestedInteger(allow_none=True)
(4)                 rAngAx = NestedBool(allow_none=True)
(4)                 right_angle_axes = Alias('rAngAx')
(4)                 perspective = NestedInteger(allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ('rotX', 'hPercent', 'rotY', 'depthPercent', 'rAngAx',
(20)                                'perspective',)
(4)                 def __init__(self,
(17)                             rotX=15,
(17)                             hPercent=None,
(17)                             rotY=20,
(17)                             depthPercent=None,
(17)                             rAngAx=True,
(17)                             perspective=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.rotX = rotX
(8)                     self.hPercent = hPercent
(8)                     self.rotY = rotY
(8)                     self.depthPercent = depthPercent
(8)                     self.rAngAx = rAngAx
(8)                     self.perspective = perspective
(0)             class Surface(Serialisable):
(4)                 tagname = "surface"
(4)                 thickness = NestedInteger(allow_none=True)
(4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)                 graphicalProperties = Alias('spPr')
(4)                 pictureOptions = Typed(expected_type=PictureOptions, allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ('thickness', 'spPr', 'pictureOptions',)
(4)                 def __init__(self,
(17)                             thickness=None,
(17)                             spPr=None,
(17)                             pictureOptions=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.thickness = thickness
(8)                     self.spPr = spPr
(8)                     self.pictureOptions = pictureOptions
(0)             class _3DBase(Serialisable):
(4)                 """
(4)                 Base class for 3D charts
(4)                 """
(4)                 tagname = "ChartBase"
(4)                 view3D = Typed(expected_type=View3D, allow_none=True)
(4)                 floor = Typed(expected_type=Surface, allow_none=True)
(4)                 sideWall = Typed(expected_type=Surface, allow_none=True)
(4)                 backWall = Typed(expected_type=Surface, allow_none=True)
(4)                 def __init__(self,
68: (17)                             view3D=None,
69: (17)                             floor=None,
70: (17)                             sideWall=None,
71: (17)                             backWall=None,
72: (17)                             ):
73: (8)                      if view3D is None:
74: (12)                         view3D = View3D()
75: (8)                      self.view3D = view3D
76: (8)                      if floor is None:
77: (12)                         floor = Surface()
78: (8)                      self.floor = floor
79: (8)                      if sideWall is None:
80: (12)                         sideWall = Surface()
81: (8)                      self.sideWall = sideWall
82: (8)                      if backWall is None:
83: (12)                         backWall = Surface()
84: (8)                      self.backWall = backWall
85: (8)                      super(_3DBase, self).__init__()

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

File 2 - cell.py:

1: (0)              """Manage individual cells in a spreadsheet.
2: (0)              The Cell class is required to know its value and type, display options,
3: (0)              and any other features of an Excel cell.  Utilities for referencing
4: (0)              cells using Excel's 'A1' column/row nomenclature are also provided.
5: (0)              """
6: (0)              __docformat__ = "restructuredtext en"
7: (0)              from copy import copy
8: (0)              import datetime
9: (0)              import re
10: (0)             from openpyxl.compat import (
11: (4)                 NUMERIC_TYPES,
12: (0)             )
13: (0)             from openpyxl.utils.exceptions import IllegalCharacterError
14: (0)             from openpyxl.utils import get_column_letter
15: (0)             from openpyxl.styles import numbers, is_date_format
16: (0)             from openpyxl.styles.styleable import StyleableObject
17: (0)             from openpyxl.worksheet.hyperlink import Hyperlink
18: (0)             from openpyxl.worksheet.formula import DataTableFormula, ArrayFormula
19: (0)             from openpyxl.cell.rich_text import CellRichText
20: (0)             TIME_TYPES = (datetime.datetime, datetime.date, datetime.time,
datetime.timedelta)
21: (0)             TIME_FORMATS = {
22: (4)                 datetime.datetime:numbers.FORMAT_DATE_DATETIME,
23: (4)                 datetime.date:numbers.FORMAT_DATE_YYYYMMDD2,
24: (4)                 datetime.time:numbers.FORMAT_DATE_TIME6,
25: (4)                 datetime.timedelta:numbers.FORMAT_DATE_TIMEDELTA,
26: (16)                            }
27: (0)             STRING_TYPES = (str, bytes, CellRichText)
28: (0)             KNOWN_TYPES = NUMERIC_TYPES + TIME_TYPES + STRING_TYPES + (bool, type(None))
29: (0)             ILLEGAL_CHARACTERS_RE = re.compile(r'[\000-\010]|[\013-\014]|[\016-\037]')
30: (0)             ERROR_CODES = ('#NULL!', '#DIV/0!', '#VALUE!', '#REF!', '#NAME?', '#NUM!',
31: (15)                           '#N/A')
32: (0)             TYPE_STRING = 's'
33: (0)             TYPE_FORMULA = 'f'
34: (0)             TYPE_NUMERIC = 'n'
35: (0)             TYPE_BOOL = 'b'
36: (0)             TYPE_NULL = 'n'
37: (0)             TYPE_INLINE = 'inlineStr'
38: (0)             TYPE_ERROR = 'e'
39: (0)             TYPE_FORMULA_CACHE_STRING = 'str'
40: (0)             VALID_TYPES = (TYPE_STRING, TYPE_FORMULA, TYPE_NUMERIC, TYPE_BOOL,
41: (15)                           TYPE_NULL, TYPE_INLINE, TYPE_ERROR, TYPE_FORMULA_CACHE_STRING)
42: (0)             _TYPES = {int:'n', float:'n', str:'s', bool:'b'}
43: (0)             def get_type(t, value):
44: (4)                 if isinstance(value, NUMERIC_TYPES):
45: (8)                     dt = 'n'
```

```
46: (4)             elif isinstance(value, STRING_TYPES):
47: (8)                 dt = 's'
48: (4)             elif isinstance(value, TIME_TYPES):
49: (8)                 dt = 'd'
50: (4)             elif isinstance(value, (DataTableFormula, ArrayFormula)):
51: (8)                 dt = 'f'
52: (4)             else:
53: (8)                 return
54: (4)             _TYPES[t] = dt
55: (4)             return dt
56: (0)         def get_time_format(t):
57: (4)             value = TIME_FORMATS.get(t)
58: (4)             if value:
59: (8)                 return value
60: (4)             for base in t.mro()[1:]:
61: (8)                 value = TIME_FORMATS.get(base)
62: (8)                 if value:
63: (12)                    TIME_FORMATS[t] = value
64: (12)                    return value
65: (4)             raise ValueError("Could not get time format for {0!r}".format(value))
66: (0)         class Cell(StyleableObject):
67: (4)             """Describes cell associated properties.
68: (4)             Properties of interest include style, type, value, and address.
69: (4)             """
70: (4)             __slots__ = (
71: (8)                 'row',
72: (8)                 'column',
73: (8)                 '_value',
74: (8)                 'data_type',
75: (8)                 'parent',
76: (8)                 '_hyperlink',
77: (8)                 '_comment',
78: (17)                         )
79: (4)             def __init__(self, worksheet, row=None, column=None, value=None,
style_array=None):
80: (8)                 super().__init__(worksheet, style_array)
81: (8)                 self.row = row
82: (8)                 """Row number of this cell (1-based)"""
83: (8)                 self.column = column
84: (8)                 """Column number of this cell (1-based)"""
85: (8)                 self._value = None
86: (8)                 self._hyperlink = None
87: (8)                 self.data_type = 'n'
88: (8)                 if value is not None:
89: (12)                    self.value = value
90: (8)                 self._comment = None
91: (4)             @property
92: (4)             def coordinate(self):
93: (8)                 """This cell's coordinate (ex. 'A5')"""
94: (8)                 col = get_column_letter(self.column)
95: (8)                 return f"{col}{self.row}"
96: (4)             @property
97: (4)             def col_idx(self):
98: (8)                 """The numerical index of the column"""
99: (8)                 return self.column
100: (4)            @property
101: (4)            def column_letter(self):
102: (8)                return get_column_letter(self.column)
103: (4)            @property
104: (4)            def encoding(self):
105: (8)                return self.parent.encoding
106: (4)            @property
107: (4)            def base_date(self):
108: (8)                return self.parent.parent.epoch
109: (4)            def __repr__(self):
110: (8)                return "<Cell {0!r}.{1}>".format(self.parent.title, self.coordinate)
111: (4)            def check_string(self, value):
112: (8)                """Check string coding, length, and line break character"""
113: (8)                if value is None:
```

```
(12)                        return
(8)                     if not isinstance(value, str):
(12)                        value = str(value, self.encoding)
(8)                     value = str(value)
(8)                     value = value[:32767]
(8)                     if next(ILLEGAL_CHARACTERS_RE.finditer(value), None):
(12)                        raise IllegalCharacterError(f"{value} cannot be used in worksheets.")
(8)                     return value
(4)                 def check_error(self, value):
(8)                     """Tries to convert Error" else N/A"""
(8)                     try:
(12)                        return str(value)
(8)                     except UnicodeDecodeError:
(12)                        return u'#N/A'
(4)                 def _bind_value(self, value):
(8)                     """Given a value, infer the correct data type"""
(8)                     self.data_type = "n"
(8)                     t = type(value)
(8)                     try:
(12)                        dt = _TYPES[t]
(8)                     except KeyError:
(12)                        dt = get_type(t, value)
(8)                     if dt is None and value is not None:
(12)                        raise ValueError("Cannot convert {0!r} to Excel".format(value))
(8)                     if dt:
(12)                        self.data_type = dt
(8)                     if dt == 'd':
(12)                        if not is_date_format(self.number_format):
(16)                            self.number_format = get_time_format(t)
(8)                     elif dt == "s" and not isinstance(value, CellRichText):
(12)                        value = self.check_string(value)
(12)                        if len(value) > 1 and value.startswith("="):
(16)                            self.data_type = 'f'
(12)                        elif value in ERROR_CODES:
(16)                            self.data_type = 'e'
(8)                     self._value = value
(4)                 @property
(4)                 def value(self):
(8)                     """Get or set the value held in the cell.
(8)                     :type: depends on the value (string, float, int or
(12)                        :class:`datetime.datetime`)
(8)                     """
(8)                     return self._value
(4)                 @value.setter
(4)                 def value(self, value):
(8)                     """Set the value and infer type and display options."""
(8)                     self._bind_value(value)
(4)                 @property
(4)                 def internal_value(self):
(8)                     """Always returns the value for excel."""
(8)                     return self._value
(4)                 @property
(4)                 def hyperlink(self):
(8)                     """Return the hyperlink target or an empty string"""
(8)                     return self._hyperlink
(4)                 @hyperlink.setter
(4)                 def hyperlink(self, val):
(8)                     """Set value and display for hyperlinks in a cell.
(8)                     Automatically sets the `value` of the cell with link text,
(8)                     but you can modify it afterwards by setting the `value`
(8)                     property, and the hyperlink will remain.
(8)                     Hyperlink is removed if set to ``None``."""
(8)                     if val is None:
(12)                        self._hyperlink = None
(8)                     else:
(12)                        if not isinstance(val, Hyperlink):
(16)                            val = Hyperlink(ref="", target=val)
(12)                        val.ref = self.coordinate
```

```
182: (12)                       self._hyperlink = val
183: (12)                       if self._value is None:
184: (16)                           self.value = val.target or val.location
185: (4)                @property
186: (4)                def is_date(self):
187: (8)                    """True if the value is formatted as a date
188: (8)                    :type: bool
189: (8)                    """
190: (8)                    return self.data_type == 'd' or (
191: (12)                       self.data_type == 'n' and is_date_format(self.number_format)
192: (12)                       )
193: (4)                def offset(self, row=0, column=0):
194: (8)                    """Returns a cell location relative to this cell.
195: (8)                    :param row: number of rows to offset
196: (8)                    :type row: int
197: (8)                    :param column: number of columns to offset
198: (8)                    :type column: int
199: (8)                    :rtype: :class:`openpyxl.cell.Cell`
200: (8)                    """
201: (8)                    offset_column = self.col_idx + column
202: (8)                    offset_row = self.row + row
203: (8)                    return self.parent.cell(column=offset_column, row=offset_row)
204: (4)                @property
205: (4)                def comment(self):
206: (8)                    """ Returns the comment associated with this cell
207: (12)                       :type: :class:`openpyxl.comments.Comment`
208: (8)                    """
209: (8)                    return self._comment
210: (4)                @comment.setter
211: (4)                def comment(self, value):
212: (8)                    """
213: (8)                    Assign a comment to a cell
214: (8)                    """
215: (8)                    if value is not None:
216: (12)                       if value.parent:
217: (16)                           value = copy(value)
218: (12)                       value.bind(self)
219: (8)                    elif value is None and self._comment:
220: (12)                       self._comment.unbind()
221: (8)                    self._comment = value
222: (0)            class MergedCell(StyleableObject):
223: (4)                """
224: (4)                Describes the properties of a cell in a merged cell and helps to
225: (4)                display the borders of the merged cell.
226: (4)                The value of a MergedCell is always None.
227: (4)                """
228: (4)                __slots__ = ('row', 'column')
229: (4)                _value = None
230: (4)                data_type = "n"
231: (4)                comment = None
232: (4)                hyperlink = None
233: (4)                def __init__(self, worksheet, row=None, column=None):
234: (8)                    super().__init__(worksheet)
235: (8)                    self.row = row
236: (8)                    self.column = column
237: (4)                def __repr__(self):
238: (8)                    return "<MergedCell {0!r}.{1}>".format(self.parent.title,
self.coordinate)
239: (4)                coordinate = Cell.coordinate
240: (4)                _comment = comment
241: (4)                value = _value
242: (0)            def WriteOnlyCell(ws=None, value=None):
243: (4)                return Cell(worksheet=ws, column=1, row=1, value=value)
```

File 3 - text.py:

```
1: (0)            """
```

```
2: (0)              Richtext definition
3: (0)              """
4: (0)              from openpyxl.descriptors.serialisable import Serialisable
5: (0)              from openpyxl.descriptors import (
6: (4)                  Alias,
7: (4)                  Typed,
8: (4)                  Integer,
9: (4)                  Set,
10: (4)                 NoneSet,
11: (4)                 Bool,
12: (4)                 String,
13: (4)                 Sequence,
14: (0)             )
15: (0)             from openpyxl.descriptors.nested import (
16: (4)                 NestedBool,
17: (4)                 NestedInteger,
18: (4)                 NestedString,
19: (4)                 NestedText,
20: (0)             )
21: (0)             from openpyxl.styles.fonts import Font
22: (0)             class PhoneticProperties(Serialisable):
23: (4)                 tagname = "phoneticPr"
24: (4)                 fontId = Integer()
25: (4)                 type = NoneSet(values=(['halfwidthKatakana', 'fullwidthKatakana',
26: (28)                                        'Hiragana', 'noConversion']))
27: (4)                 alignment = NoneSet(values=(['noControl', 'left', 'center',
'distributed']))
28: (4)                 def __init__(self,
29: (17)                             fontId=None,
30: (17)                             type=None,
31: (17)                             alignment=None,
32: (16)                            ):
33: (8)                     self.fontId = fontId
34: (8)                     self.type = type
35: (8)                     self.alignment = alignment
36: (0)             class PhoneticText(Serialisable):
37: (4)                 tagname = "rPh"
38: (4)                 sb = Integer()
39: (4)                 eb = Integer()
40: (4)                 t = NestedText(expected_type=str)
41: (4)                 text = Alias('t')
42: (4)                 def __init__(self,
43: (17)                             sb=None,
44: (17)                             eb=None,
45: (17)                             t=None,
46: (16)                            ):
47: (8)                     self.sb = sb
48: (8)                     self.eb = eb
49: (8)                     self.t = t
50: (0)             class InlineFont(Font):
51: (4)                 """
52: (4)                 Font for inline text because, yes what you need are different objects with
the same elements but different constraints.
53: (4)                 """
54: (4)                 tagname = "RPrElt"
55: (4)                 rFont = NestedString(allow_none=True)
56: (4)                 charset = Font.charset
57: (4)                 family = Font.family
58: (4)                 b =Font.b
59: (4)                 i = Font.i
60: (4)                 strike = Font.strike
61: (4)                 outline = Font.outline
62: (4)                 shadow = Font.shadow
63: (4)                 condense = Font.condense
64: (4)                 extend = Font.extend
65: (4)                 color = Font.color
66: (4)                 sz = Font.sz
67: (4)                 u = Font.u
68: (4)                 vertAlign = Font.vertAlign
```

```
(4)              scheme = Font.scheme
(4)              __elements__ = ('rFont', 'charset', 'family', 'b', 'i', 'strike',
(20)                             'outline', 'shadow', 'condense', 'extend', 'color', 'sz',
'u',
(20)                             'vertAlign', 'scheme')
(4)              def __init__(self,
(17)                          rFont=None,
(17)                          charset=None,
(17)                          family=None,
(17)                          b=None,
(17)                          i=None,
(17)                          strike=None,
(17)                          outline=None,
(17)                          shadow=None,
(17)                          condense=None,
(17)                          extend=None,
(17)                          color=None,
(17)                          sz=None,
(17)                          u=None,
(17)                          vertAlign=None,
(17)                          scheme=None,
(16)                         ):
(8)                  self.rFont = rFont
(8)                  self.charset = charset
(8)                  self.family = family
(8)                  self.b = b
(8)                  self.i = i
(8)                  self.strike = strike
(8)                  self.outline = outline
(8)                  self.shadow = shadow
(8)                  self.condense = condense
(8)                  self.extend = extend
(8)                  self.color = color
(8)                  self.sz = sz
(8)                  self.u = u
(8)                  self.vertAlign = vertAlign
(8)                  self.scheme = scheme
(0)          class RichText(Serialisable):
(4)              tagname = "RElt"
(4)              rPr = Typed(expected_type=InlineFont, allow_none=True)
(4)              font = Alias("rPr")
(4)              t = NestedText(expected_type=str, allow_none=True)
(4)              text = Alias("t")
(4)              __elements__ = ('rPr', 't')
(4)              def __init__(self,
(17)                          rPr=None,
(17)                          t=None,
(16)                         ):
(8)                  self.rPr = rPr
(8)                  self.t = t
(0)          class Text(Serialisable):
(4)              tagname = "text"
(4)              t = NestedText(allow_none=True, expected_type=str)
(4)              plain = Alias("t")
(4)              r = Sequence(expected_type=RichText, allow_none=True)
(4)              formatted = Alias("r")
(4)              rPh = Sequence(expected_type=PhoneticText, allow_none=True)
(4)              phonetic = Alias("rPh")
(4)              phoneticPr = Typed(expected_type=PhoneticProperties, allow_none=True)
(4)              PhoneticProperties = Alias("phoneticPr")
(4)              __elements__ = ('t', 'r', 'rPh', 'phoneticPr')
(4)              def __init__(self,
(17)                          t=None,
(17)                          r=(),
(17)                          rPh=(),
(17)                          phoneticPr=None,
(16)                         ):
(8)                  self.t = t
(8)                  self.r = r
```

```
        self.rPh = rPh
        self.phoneticPr = phoneticPr
    @property
    def content(self):
        """
        Text stripped of all formatting
        """
        snippets = []
        if self.plain is not None:
            snippets.append(self.plain)
        for block in self.formatted:
            if block.t is not None:
                snippets.append(block.t)
        return u"".join(snippets)
```

File 4 - axis.py:

```
from openpyxl.descriptors.serialisable import Serialisable
from openpyxl.descriptors import (
    Typed,
    Float,
    NoneSet,
    Bool,
    Integer,
    MinMax,
    NoneSet,
    Set,
    String,
    Alias,
)
from openpyxl.descriptors.excel import (
    ExtensionList,
    Percentage,
    _explicit_none,
)
from openpyxl.descriptors.nested import (
    NestedValue,
    NestedSet,
    NestedBool,
    NestedNoneSet,
    NestedFloat,
    NestedInteger,
    NestedMinMax,
)
from openpyxl.xml.constants import CHART_NS
from .descriptors import NumberFormatDescriptor
from .layout import Layout
from .text import Text, RichText
from .shapes import GraphicalProperties
from .title import Title, TitleDescriptor
class ChartLines(Serialisable):
    tagname = "chartLines"
    spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
    graphicalProperties = Alias('spPr')
    def __init__(self, spPr=None):
        self.spPr = spPr
class Scaling(Serialisable):
    tagname = "scaling"
    logBase = NestedFloat(allow_none=True)
    orientation = NestedSet(values=(['maxMin', 'minMax']))
    max = NestedFloat(allow_none=True)
    min = NestedFloat(allow_none=True)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('logBase', 'orientation', 'max', 'min',)
    def __init__(self,
                 logBase=None,
                 orientation="minMax",
51: (17)                            max=None,
52: (17)                            min=None,
53: (17)                            extLst=None,
54: (16)                           ):
55: (8)                    self.logBase = logBase
56: (8)                    self.orientation = orientation
57: (8)                    self.max = max
58: (8)                    self.min = min
59: (0)            class _BaseAxis(Serialisable):
60: (4)                axId = NestedInteger(expected_type=int)
61: (4)                scaling = Typed(expected_type=Scaling)
62: (4)                delete = NestedBool(allow_none=True)
63: (4)                axPos = NestedSet(values=(['b', 'l', 'r', 't']))
64: (4)                majorGridlines = Typed(expected_type=ChartLines, allow_none=True)
65: (4)                minorGridlines = Typed(expected_type=ChartLines, allow_none=True)
66: (4)                title = TitleDescriptor()
67: (4)                numFmt = NumberFormatDescriptor()
68: (4)                number_format = Alias("numFmt")
69: (4)                majorTickMark = NestedNoneSet(values=(['cross', 'in', 'out']),
to_tree=_explicit_none)
70: (4)                minorTickMark = NestedNoneSet(values=(['cross', 'in', 'out']),
to_tree=_explicit_none)
71: (4)                tickLblPos = NestedNoneSet(values=(['high', 'low', 'nextTo']))
72: (4)                spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
73: (4)                graphicalProperties = Alias('spPr')
74: (4)                txPr = Typed(expected_type=RichText, allow_none=True)
75: (4)                textProperties = Alias('txPr')
76: (4)                crossAx = NestedInteger(expected_type=int) # references other axis
77: (4)                crosses = NestedNoneSet(values=(['autoZero', 'max', 'min']))
78: (4)                crossesAt = NestedFloat(allow_none=True)
79: (4)                __elements__ = ('axId', 'scaling', 'delete', 'axPos', 'majorGridlines',
80: (20)                                'minorGridlines', 'title', 'numFmt', 'majorTickMark',
'minorTickMark',
81: (20)                                'tickLblPos', 'spPr', 'txPr', 'crossAx', 'crosses',
'crossesAt')
82: (4)                def __init__(self,
83: (17)                            axId=None,
84: (17)                            scaling=None,
85: (17)                            delete=None,
86: (17)                            axPos='l',
87: (17)                            majorGridlines=None,
88: (17)                            minorGridlines=None,
89: (17)                            title=None,
90: (17)                            numFmt=None,
91: (17)                            majorTickMark=None,
92: (17)                            minorTickMark=None,
93: (17)                            tickLblPos=None,
94: (17)                            spPr=None,
95: (17)                            txPr= None,
96: (17)                            crossAx=None,
97: (17)                            crosses=None,
98: (17)                            crossesAt=None,
99: (16)                           ):
100: (8)                   self.axId = axId
101: (8)                   if scaling is None:
102: (12)                      scaling = Scaling()
103: (8)                   self.scaling = scaling
104: (8)                   self.delete = delete
105: (8)                   self.axPos = axPos
106: (8)                   self.majorGridlines = majorGridlines
107: (8)                   self.minorGridlines = minorGridlines
108: (8)                   self.title = title
109: (8)                   self.numFmt = numFmt
110: (8)                   self.majorTickMark = majorTickMark
111: (8)                   self.minorTickMark = minorTickMark
112: (8)                   self.tickLblPos = tickLblPos
113: (8)                   self.spPr = spPr
114: (8)                   self.txPr = txPr
115: (8)                   self.crossAx = crossAx
```

```
116: (8)                    self.crosses = crosses
117: (8)                    self.crossesAt = crossesAt
118: (0)            class DisplayUnitsLabel(Serialisable):
119: (4)                tagname = "dispUnitsLbl"
120: (4)                layout = Typed(expected_type=Layout, allow_none=True)
121: (4)                tx = Typed(expected_type=Text, allow_none=True)
122: (4)                text = Alias("tx")
123: (4)                spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
124: (4)                graphicalProperties = Alias("spPr")
125: (4)                txPr = Typed(expected_type=RichText, allow_none=True)
126: (4)                textPropertes = Alias("txPr")
127: (4)                __elements__ = ('layout', 'tx', 'spPr', 'txPr')
128: (4)                def __init__(self,
129: (17)                             layout=None,
130: (17)                             tx=None,
131: (17)                             spPr=None,
132: (17)                             txPr=None,
133: (16)                            ):
134: (8)                    self.layout = layout
135: (8)                    self.tx = tx
136: (8)                    self.spPr = spPr
137: (8)                    self.txPr = txPr
138: (0)            class DisplayUnitsLabelList(Serialisable):
139: (4)                tagname = "dispUnits"
140: (4)                custUnit = NestedFloat(allow_none=True)
141: (4)                builtInUnit = NestedNoneSet(values=(['hundreds', 'thousands',
142: (41)                                                     'tenThousands', 'hundredThousands',
'millions', 'tenMillions',
143: (41)                                                     'hundredMillions', 'billions',
'trillions']))
144: (4)                dispUnitsLbl = Typed(expected_type=DisplayUnitsLabel, allow_none=True)
145: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
146: (4)                __elements__ = ('custUnit', 'builtInUnit', 'dispUnitsLbl',)
147: (4)                def __init__(self,
148: (17)                             custUnit=None,
149: (17)                             builtInUnit=None,
150: (17)                             dispUnitsLbl=None,
151: (17)                             extLst=None,
152: (16)                            ):
153: (8)                    self.custUnit = custUnit
154: (8)                    self.builtInUnit = builtInUnit
155: (8)                    self.dispUnitsLbl = dispUnitsLbl
156: (0)            class NumericAxis(_BaseAxis):
157: (4)                tagname = "valAx"
158: (4)                axId = _BaseAxis.axId
159: (4)                scaling = _BaseAxis.scaling
160: (4)                delete = _BaseAxis.delete
161: (4)                axPos = _BaseAxis.axPos
162: (4)                majorGridlines = _BaseAxis.majorGridlines
163: (4)                minorGridlines = _BaseAxis.minorGridlines
164: (4)                title = _BaseAxis.title
165: (4)                numFmt = _BaseAxis.numFmt
166: (4)                majorTickMark = _BaseAxis.majorTickMark
167: (4)                minorTickMark = _BaseAxis.minorTickMark
168: (4)                tickLblPos = _BaseAxis.tickLblPos
169: (4)                spPr = _BaseAxis.spPr
170: (4)                txPr = _BaseAxis.txPr
171: (4)                crossAx = _BaseAxis.crossAx
172: (4)                crosses = _BaseAxis.crosses
173: (4)                crossesAt = _BaseAxis.crossesAt
174: (4)                crossBetween = NestedNoneSet(values=(['between', 'midCat']))
175: (4)                majorUnit = NestedFloat(allow_none=True)
176: (4)                minorUnit = NestedFloat(allow_none=True)
177: (4)                dispUnits = Typed(expected_type=DisplayUnitsLabelList, allow_none=True)
178: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
179: (4)                __elements__ = _BaseAxis.__elements__ + ('crossBetween', 'majorUnit',
180: (45)                                                         'minorUnit', 'dispUnits',)
181: (4)                def __init__(self,
182: (17)                             crossBetween=None,
(17)                     majorUnit=None,
(17)                     minorUnit=None,
(17)                     dispUnits=None,
(17)                     extLst=None,
(17)                     **kw
(16)                    ):
(8)             self.crossBetween = crossBetween
(8)             self.majorUnit = majorUnit
(8)             self.minorUnit = minorUnit
(8)             self.dispUnits = dispUnits
(8)             kw.setdefault('majorGridlines', ChartLines())
(8)             kw.setdefault('axId', 100)
(8)             kw.setdefault('crossAx', 10)
(8)             super().__init__(**kw)
(4)         @classmethod
(4)         def from_tree(cls, node):
(8)             """
(8)             Special case value axes with no gridlines
(8)             """
(8)             self = super().from_tree(node)
(8)             gridlines = node.find("{%s}majorGridlines" % CHART_NS)
(8)             if gridlines is None:
(12)                self.majorGridlines = None
(8)             return self
(0)     class TextAxis(_BaseAxis):
(4)         tagname = "catAx"
(4)         axId = _BaseAxis.axId
(4)         scaling = _BaseAxis.scaling
(4)         delete = _BaseAxis.delete
(4)         axPos = _BaseAxis.axPos
(4)         majorGridlines = _BaseAxis.majorGridlines
(4)         minorGridlines = _BaseAxis.minorGridlines
(4)         title = _BaseAxis.title
(4)         numFmt = _BaseAxis.numFmt
(4)         majorTickMark = _BaseAxis.majorTickMark
(4)         minorTickMark = _BaseAxis.minorTickMark
(4)         tickLblPos = _BaseAxis.tickLblPos
(4)         spPr = _BaseAxis.spPr
(4)         txPr = _BaseAxis.txPr
(4)         crossAx = _BaseAxis.crossAx
(4)         crosses = _BaseAxis.crosses
(4)         crossesAt = _BaseAxis.crossesAt
(4)         auto = NestedBool(allow_none=True)
(4)         lblAlgn = NestedNoneSet(values=(['ctr', 'l', 'r']))
(4)         lblOffset = NestedMinMax(min=0, max=1000)
(4)         tickLblSkip = NestedInteger(allow_none=True)
(4)         tickMarkSkip = NestedInteger(allow_none=True)
(4)         noMultiLvlLbl = NestedBool(allow_none=True)
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         __elements__ = _BaseAxis.__elements__ + ('auto', 'lblAlgn', 'lblOffset',
(45)                                                 'tickLblSkip', 'tickMarkSkip',
'noMultiLvlLbl')
(4)         def __init__(self,
(17)                     auto=None,
(17)                     lblAlgn=None,
(17)                     lblOffset=100,
(17)                     tickLblSkip=None,
(17)                     tickMarkSkip=None,
(17)                     noMultiLvlLbl=None,
(17)                     extLst=None,
(17)                     **kw
(16)                    ):
(8)             self.auto = auto
(8)             self.lblAlgn = lblAlgn
(8)             self.lblOffset = lblOffset
(8)             self.tickLblSkip = tickLblSkip
(8)             self.tickMarkSkip = tickMarkSkip
(8)             self.noMultiLvlLbl = noMultiLvlLbl
(8)             kw.setdefault('axId', 10)
```

```
251: (8)                    kw.setdefault('crossAx', 100)
252: (8)                    super().__init__(**kw)
253: (0)            class DateAxis(TextAxis):
254: (4)                tagname = "dateAx"
255: (4)                axId = _BaseAxis.axId
256: (4)                scaling = _BaseAxis.scaling
257: (4)                delete = _BaseAxis.delete
258: (4)                axPos = _BaseAxis.axPos
259: (4)                majorGridlines = _BaseAxis.majorGridlines
260: (4)                minorGridlines = _BaseAxis.minorGridlines
261: (4)                title = _BaseAxis.title
262: (4)                numFmt = _BaseAxis.numFmt
263: (4)                majorTickMark = _BaseAxis.majorTickMark
264: (4)                minorTickMark = _BaseAxis.minorTickMark
265: (4)                tickLblPos = _BaseAxis.tickLblPos
266: (4)                spPr = _BaseAxis.spPr
267: (4)                txPr = _BaseAxis.txPr
268: (4)                crossAx = _BaseAxis.crossAx
269: (4)                crosses = _BaseAxis.crosses
270: (4)                crossesAt = _BaseAxis.crossesAt
271: (4)                auto = NestedBool(allow_none=True)
272: (4)                lblOffset = NestedInteger(allow_none=True)
273: (4)                baseTimeUnit = NestedNoneSet(values=(['days', 'months', 'years']))
274: (4)                majorUnit = NestedFloat(allow_none=True)
275: (4)                majorTimeUnit = NestedNoneSet(values=(['days', 'months', 'years']))
276: (4)                minorUnit = NestedFloat(allow_none=True)
277: (4)                minorTimeUnit = NestedNoneSet(values=(['days', 'months', 'years']))
278: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
279: (4)                __elements__ = _BaseAxis.__elements__ + ('auto', 'lblOffset',
280: (45)                                                 'baseTimeUnit', 'majorUnit',
'majorTimeUnit', 'minorUnit',
281: (45)                                                 'minorTimeUnit')
282: (4)                def __init__(self,
283: (17)                             auto=None,
284: (17)                             lblOffset=None,
285: (17)                             baseTimeUnit=None,
286: (17)                             majorUnit=None,
287: (17)                             majorTimeUnit=None,
288: (17)                             minorUnit=None,
289: (17)                             minorTimeUnit=None,
290: (17)                             extLst=None,
291: (17)                             **kw
292: (16)                            ):
293: (8)                    self.auto = auto
294: (8)                    self.lblOffset = lblOffset
295: (8)                    self.baseTimeUnit = baseTimeUnit
296: (8)                    self.majorUnit = majorUnit
297: (8)                    self.majorTimeUnit = majorTimeUnit
298: (8)                    self.minorUnit = minorUnit
299: (8)                    self.minorTimeUnit = minorTimeUnit
300: (8)                    kw.setdefault('axId', 500)
301: (8)                    kw.setdefault('lblOffset', lblOffset)
302: (8)                    super().__init__(**kw)
303: (0)            class SeriesAxis(_BaseAxis):
304: (4)                tagname = "serAx"
305: (4)                axId = _BaseAxis.axId
306: (4)                scaling = _BaseAxis.scaling
307: (4)                delete = _BaseAxis.delete
308: (4)                axPos = _BaseAxis.axPos
309: (4)                majorGridlines = _BaseAxis.majorGridlines
310: (4)                minorGridlines = _BaseAxis.minorGridlines
311: (4)                title = _BaseAxis.title
312: (4)                numFmt = _BaseAxis.numFmt
313: (4)                majorTickMark = _BaseAxis.majorTickMark
314: (4)                minorTickMark = _BaseAxis.minorTickMark
315: (4)                tickLblPos = _BaseAxis.tickLblPos
316: (4)                spPr = _BaseAxis.spPr
317: (4)                txPr = _BaseAxis.txPr
318: (4)                crossAx = _BaseAxis.crossAx
```

```
319: (4)                crosses = _BaseAxis.crosses
320: (4)                crossesAt = _BaseAxis.crossesAt
321: (4)                tickLblSkip = NestedInteger(allow_none=True)
322: (4)                tickMarkSkip = NestedInteger(allow_none=True)
323: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
324: (4)                __elements__ = _BaseAxis.__elements__ + ('tickLblSkip', 'tickMarkSkip')
325: (4)                def __init__(self,
326: (17)                            tickLblSkip=None,
327: (17)                            tickMarkSkip=None,
328: (17)                            extLst=None,
329: (17)                            **kw
330: (16)                           ):
331: (8)                    self.tickLblSkip = tickLblSkip
332: (8)                    self.tickMarkSkip = tickMarkSkip
333: (8)                    kw.setdefault('axId', 1000)
334: (8)                    kw.setdefault('crossAx', 10)
335: (8)                    super().__init__(**kw)

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

File 5 - _chart.py:

1: (0)              from collections import OrderedDict
2: (0)              from operator import attrgetter
3: (0)              from openpyxl.descriptors import (
4: (4)                  Typed,
5: (4)                  Integer,
6: (4)                  Alias,
7: (4)                  MinMax,
8: (4)                  Bool,
9: (4)                  Set,
10: (0)             )
11: (0)             from openpyxl.descriptors.sequence import ValueSequence
12: (0)             from openpyxl.descriptors.serialisable import Serialisable
13: (0)             from ._3d import _3DBase
14: (0)             from .data_source import AxDataSource, NumRef
15: (0)             from .layout import Layout
16: (0)             from .legend import Legend
17: (0)             from .reference import Reference
18: (0)             from .series_factory import SeriesFactory
19: (0)             from .series import attribute_mapping
20: (0)             from .shapes import GraphicalProperties
21: (0)             from .title import TitleDescriptor
22: (0)             class AxId(Serialisable):
23: (4)                 val = Integer()
24: (4)                 def __init__(self, val):
25: (8)                     self.val = val
26: (0)             def PlotArea():
27: (4)                 from .chartspace import PlotArea
28: (4)                 return PlotArea()
29: (0)             class ChartBase(Serialisable):
30: (4)                 """
31: (4)                 Base class for all charts
32: (4)                 """
33: (4)                 legend = Typed(expected_type=Legend, allow_none=True)
34: (4)                 layout = Typed(expected_type=Layout, allow_none=True)
35: (4)                 roundedCorners = Bool(allow_none=True)
36: (4)                 axId = ValueSequence(expected_type=int)
37: (4)                 visible_cells_only = Bool(allow_none=True)
38: (4)                 display_blanks = Set(values=['span', 'gap', 'zero'])
39: (4)                 graphical_properties = Typed(expected_type=GraphicalProperties,
allow_none=True)
40: (4)                 _series_type = ""
41: (4)                 ser = ()
42: (4)                 series = Alias('ser')
43: (4)                 title = TitleDescriptor()
44: (4)                 anchor = "E15" # default anchor position
45: (4)                 width = 15 # in cm, approx 5 rows
46: (4)                 height = 7.5 # in cm, approx 14 rows
```

```
(4)         _id = 1
(4)         _path = "/xl/charts/chart{0}.xml"
(4)         style = MinMax(allow_none=True, min=1, max=48)
(4)         mime_type = "application/vnd.openxmlformats-officedocument.drawingml.chart+xml"
(4)         graphical_properties = Typed(expected_type=GraphicalProperties, allow_none=True) # mapped to chartspace
(4)         __elements__ = ()
(4)         def __init__(self, axId=(), **kw):
(8)             self._charts = [self]
(8)             self.title = None
(8)             self.layout = None
(8)             self.roundedCorners = None
(8)             self.legend = Legend()
(8)             self.graphical_properties = None
(8)             self.style = None
(8)             self.plot_area = PlotArea()
(8)             self.axId = axId
(8)             self.display_blanks = 'gap'
(8)             self.pivotSource = None
(8)             self.pivotFormats = ()
(8)             self.visible_cells_only = True
(8)             self.idx_base = 0
(8)             self.graphical_properties = None
(8)             super().__init__()
(4)         def __hash__(self):
(8)             """
(8)             Just need to check for identity
(8)             """
(8)             return id(self)
(4)         def __iadd__(self, other):
(8)             """
(8)             Combine the chart with another one
(8)             """
(8)             if not isinstance(other, ChartBase):
(12)                raise TypeError("Only other charts can be added")
(8)             self._charts.append(other)
(8)             return self
(4)         def to_tree(self, namespace=None, tagname=None, idx=None):
(8)             self.axId = [id for id in self._axes]
(8)             if self.ser is not None:
(12)                for s in self.ser:
(16)                    s.__elements__ = attribute_mapping[self._series_type]
(8)             return super().to_tree(tagname, idx)
(4)         def _reindex(self):
(8)             """
(8)             Normalise and rebase series: sort by order and then rebase order
(8)             """
(8)             ds = sorted(self.series, key=attrgetter("order"))
(8)             for idx, s in enumerate(ds):
(12)                s.order = idx
(8)             self.series = ds
(4)         def _write(self):
(8)             from .chartspace import ChartSpace, ChartContainer
(8)             self.plot_area.layout = self.layout
(8)             idx_base = self.idx_base
(8)             for chart in self._charts:
(12)                if chart not in self.plot_area._charts:
(16)                    chart.idx_base = idx_base
(16)                    idx_base += len(chart.series)
(8)             self.plot_area._charts = self._charts
(8)             container = ChartContainer(plotArea=self.plot_area, legend=self.legend, title=self.title)
(8)             if isinstance(chart, _3DBase):
(12)                container.view3D = chart.view3D
(12)                container.floor = chart.floor
(12)                container.sideWall = chart.sideWall
(12)                container.backWall = chart.backWall
(8)             container.plotVisOnly = self.visible_cells_only
```

```
        container.dispBlanksAs = self.display_blanks
        container.pivotFmts = self.pivotFormats
        cs = ChartSpace(chart=container)
        cs.style = self.style
        cs.roundedCorners = self.roundedCorners
        cs.pivotSource = self.pivotSource
        cs.spPr = self.graphical_properties
        return cs.to_tree()
    @property
    def _axes(self):
        x = getattr(self, "x_axis", None)
        y = getattr(self, "y_axis", None)
        z = getattr(self, "z_axis", None)
        return OrderedDict([(axis.axId, axis) for axis in (x, y, z) if axis])
    def set_categories(self, labels):
        """
        Set the categories / x-axis values
        """
        if not isinstance(labels, Reference):
            labels = Reference(range_string=labels)
        for s in self.ser:
            s.cat = AxDataSource(numRef=NumRef(f=labels))
    def add_data(self, data, from_rows=False, titles_from_data=False):
        """
        Add a range of data in a single pass.
        The default is to treat each column as a data series.
        """
        if not isinstance(data, Reference):
            data = Reference(range_string=data)
        if from_rows:
            values = data.rows
        else:
            values = data.cols
        for ref in values:
            series = SeriesFactory(ref, title_from_data=titles_from_data)
            self.series.append(series)
    def append(self, value):
        """Append a data series to the chart"""
        l = self.series[:]
        l.append(value)
        self.series = l
    @property
    def path(self):
        return self._path.format(self._id)
```

File 6 - _writer.py:

```
from openpyxl.compat import safe_string
from openpyxl.xml.functions import Element, SubElement, whitespace, XML_NS
from openpyxl import LXML
from openpyxl.utils.datetime import to_excel, to_ISO8601
from datetime import timedelta
from openpyxl.worksheet.formula import DataTableFormula, ArrayFormula
from openpyxl.cell.rich_text import CellRichText
def _set_attributes(cell, styled=None):
    """
    Set coordinate and datatype
    """
    coordinate = cell.coordinate
    attrs = {'r': coordinate}
    if styled:
        attrs['s'] = f"{cell.style_id}"
    if cell.data_type == "s":
        attrs['t'] = "inlineStr"
    elif cell.data_type != 'f':
        attrs['t'] = cell.data_type
    value = cell._value
```

```
21: (4)                 if cell.data_type == "d":
22: (8)                     if hasattr(value, "tzinfo") and value.tzinfo is not None:
23: (12)                        raise TypeError("Excel does not support timezones in datetimes. "
24: (20)                                "The tzinfo in the datetime/time object must be set to
None.")
25: (8)                     if cell.parent.parent.iso_dates and not isinstance(value, timedelta):
26: (12)                        value = to_ISO8601(value)
27: (8)                     else:
28: (12)                        attrs['t'] = "n"
29: (12)                        value = to_excel(value, cell.parent.parent.epoch)
30: (4)                 if cell.hyperlink:
31: (8)                     cell.parent._hyperlinks.append(cell.hyperlink)
32: (4)                 return value, attrs
33: (0)             def etree_write_cell(xf, worksheet, cell, styled=None):
34: (4)                 value, attributes = _set_attributes(cell, styled)
35: (4)                 el = Element("c", attributes)
36: (4)                 if value is None or value == "":
37: (8)                     xf.write(el)
38: (8)                     return
39: (4)                 if cell.data_type == 'f':
40: (8)                     attrib = {}
41: (8)                     if isinstance(value, ArrayFormula):
42: (12)                        attrib = dict(value)
43: (12)                        value = value.text
44: (8)                     elif isinstance(value, DataTableFormula):
45: (12)                        attrib = dict(value)
46: (12)                        value = None
47: (8)                     formula = SubElement(el, 'f', attrib)
48: (8)                     if value is not None and not attrib.get('t') == "dataTable":
49: (12)                        formula.text = value[1:]
50: (12)                        value = None
51: (4)                 if cell.data_type == 's':
52: (8)                     if isinstance(value, CellRichText):
53: (12)                        el.append(value.to_tree())
54: (8)                     else:
55: (12)                        inline_string = Element("is")
56: (12)                        text = Element('t')
57: (12)                        text.text = value
58: (12)                        whitespace(text)
59: (12)                        inline_string.append(text)
60: (12)                        el.append(inline_string)
61: (4)                 else:
62: (8)                     cell_content = SubElement(el, 'v')
63: (8)                     if value is not None:
64: (12)                        cell_content.text = safe_string(value)
65: (4)                 xf.write(el)
66: (0)             def lxml_write_cell(xf, worksheet, cell, styled=False):
67: (4)                 value, attributes = _set_attributes(cell, styled)
68: (4)                 if value == '' or value is None:
69: (8)                     with xf.element("c", attributes):
70: (12)                        return
71: (4)                 with xf.element('c', attributes):
72: (8)                     if cell.data_type == 'f':
73: (12)                        attrib = {}
74: (12)                        if isinstance(value, ArrayFormula):
75: (16)                            attrib = dict(value)
76: (16)                            value = value.text
77: (12)                        elif isinstance(value, DataTableFormula):
78: (16)                            attrib = dict(value)
79: (16)                            value = None
80: (12)                        with xf.element('f', attrib):
81: (16)                            if value is not None and not attrib.get('t') == "dataTable":
82: (20)                                xf.write(value[1:])
83: (20)                                value = None
84: (8)                     if cell.data_type == 's':
85: (12)                        if isinstance(value, CellRichText):
86: (16)                            el = value.to_tree()
87: (16)                            xf.write(el)
88: (12)                        else:
```

```
89: (16)                        with xf.element("is"):
90: (20)                            if isinstance(value, str):
91: (24)                                attrs = {}
92: (24)                                if value != value.strip():
93: (28)                                    attrs["{%s}space" % XML_NS] = "preserve"
94: (24)                                el = Element("t", attrs) # lxml can't handle xml-ns
95: (24)                                el.text = value
96: (24)                                xf.write(el)
97: (8)                 else:
98: (12)                    with xf.element("v"):
99: (16)                        if value is not None:
100: (20)                           xf.write(safe_string(value))
101: (0)        if LXML:
102: (4)            write_cell = lxml_write_cell
103: (0)        else:
104: (4)            write_cell = etree_write_cell

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

File 7 - __init__.py:

1: (0)              DEBUG = False
2: (0)              from openpyxl.compat.numbers import NUMPY
3: (0)              from openpyxl.xml import DEFUSEDXML, LXML
4: (0)              from openpyxl.workbook import Workbook
5: (0)              from openpyxl.reader.excel import load_workbook as open
6: (0)              from openpyxl.reader.excel import load_workbook
7: (0)              import openpyxl._constants as constants
8: (0)              __author__ = constants.__author__
9: (0)              __author_email__ = constants.__author_email__
10: (0)             __license__ = constants.__license__
11: (0)             __maintainer_email__ = constants.__maintainer_email__
12: (0)             __url__ = constants.__url__
13: (0)             __version__ = constants.__version__

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

File 8 - __init__.py:

1: (0)              from .cell import Cell, WriteOnlyCell, MergedCell
2: (0)              from .read_only import ReadOnlyCell

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

File 9 - __init__.py:

1: (0)              from .area_chart import AreaChart, AreaChart3D
2: (0)              from .bar_chart import BarChart, BarChart3D
3: (0)              from .bubble_chart import BubbleChart
4: (0)              from .line_chart import LineChart, LineChart3D
5: (0)              from .pie_chart import (
6: (4)                  PieChart,
7: (4)                  PieChart3D,
8: (4)                  DoughnutChart,
9: (4)                  ProjectedPieChart
10: (0)             )
11: (0)             from .radar_chart import RadarChart
12: (0)             from .scatter_chart import ScatterChart
13: (0)             from .stock_chart import StockChart
14: (0)             from .surface_chart import SurfaceChart, SurfaceChart3D
15: (0)             from .series_factory import SeriesFactory as Series
16: (0)             from .reference import Reference

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

File 10 - read_only.py:

1: (0)              from openpyxl.cell import Cell
2: (0)              from openpyxl.utils import get_column_letter
```

```
(0)             from openpyxl.utils.datetime import from_excel
(0)             from openpyxl.styles import is_date_format
(0)             from openpyxl.styles.numbers import BUILTIN_FORMATS, BUILTIN_FORMATS_MAX_SIZE
(0)             class ReadOnlyCell:
(4)                 __slots__ =  ('parent', 'row', 'column', '_value', 'data_type', '_style_id')
(4)                 def __init__(self, sheet, row, column, value, data_type='n', style_id=0):
(8)                     self.parent = sheet
(8)                     self._value = None
(8)                     self.row = row
(8)                     self.column = column
(8)                     self.data_type = data_type
(8)                     self.value = value
(8)                     self._style_id = style_id
(4)                 def __eq__(self, other):
(8)                     for a in self.__slots__:
(12)                        if getattr(self, a) != getattr(other, a):
(16)                            return
(8)                     return True
(4)                 def __ne__(self, other):
(8)                     return not self.__eq__(other)
(4)                 def __repr__(self):
(8)                     return "<ReadOnlyCell {0!r}.{1}>".format(self.parent.title, self.coordinate)
(4)                 @property
(4)                 def coordinate(self):
(8)                     column = get_column_letter(self.column)
(8)                     return "{1}{0}".format(self.row, column)
(4)                 @property
(4)                 def coordinate(self):
(8)                     return Cell.coordinate.__get__(self)
(4)                 @property
(4)                 def column_letter(self):
(8)                     return Cell.column_letter.__get__(self)
(4)                 @property
(4)                 def style_array(self):
(8)                     return self.parent.parent._cell_styles[self._style_id]
(4)                 @property
(4)                 def has_style(self):
(8)                     return self._style_id != 0
(4)                 @property
(4)                 def number_format(self):
(8)                     _id = self.style_array.numFmtId
(8)                     if _id < BUILTIN_FORMATS_MAX_SIZE:
(12)                        return BUILTIN_FORMATS.get(_id, "General")
(8)                     else:
(12)                        return self.parent.parent._number_formats[
(16)                            _id - BUILTIN_FORMATS_MAX_SIZE]
(4)                 @property
(4)                 def font(self):
(8)                     _id = self.style_array.fontId
(8)                     return self.parent.parent._fonts[_id]
(4)                 @property
(4)                 def fill(self):
(8)                     _id = self.style_array.fillId
(8)                     return self.parent.parent._fills[_id]
(4)                 @property
(4)                 def border(self):
(8)                     _id = self.style_array.borderId
(8)                     return self.parent.parent._borders[_id]
(4)                 @property
(4)                 def alignment(self):
(8)                     _id = self.style_array.alignmentId
(8)                     return self.parent.parent._alignments[_id]
(4)                 @property
(4)                 def protection(self):
(8)                     _id = self.style_array.protectionId
(8)                     return self.parent.parent._protections[_id]
(4)                 @property
```

```
(4)                 def is_date(self):
(8)                     return Cell.is_date.__get__(self)
(4)                 @property
(4)                 def internal_value(self):
(8)                     return self._value
(4)                 @property
(4)                 def value(self):
(8)                     return self._value
(4)                 @value.setter
(4)                 def value(self, value):
(8)                     if self._value is not None:
(12)                        raise AttributeError("Cell is read only")
(8)                     self._value = value
(0)             class EmptyCell:
(4)                 __slots__ = ()
(4)                 value = None
(4)                 is_date = False
(4)                 font = None
(4)                 border = None
(4)                 fill = None
(4)                 number_format = None
(4)                 alignment = None
(4)                 data_type = 'n'
(4)                 def __repr__(self):
(8)                     return "<EmptyCell>"
(0)             EMPTY_CELL = EmptyCell()

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

File 11 - rich_text.py:

(0)             """
(0)             RichText definition
(0)             """
(0)             from copy import copy
(0)             from openpyxl.compat import NUMERIC_TYPES
(0)             from openpyxl.cell.text import InlineFont, Text
(0)             from openpyxl.descriptors import (
(4)                 Strict,
(4)                 String,
(4)                 Typed
(0)             )
(0)             from openpyxl.xml.functions import Element, whitespace
(0)             class TextBlock(Strict):
(4)                 """ Represents text string in a specific format
(4)                 This class is used as part of constructing a rich text strings.
(4)                 """
(4)                 font = Typed(expected_type=InlineFont)
(4)                 text = String()
(4)                 def __init__(self, font, text):
(8)                     self.font = font
(8)                     self.text = text
(4)                 def __eq__(self, other):
(8)                     return self.text == other.text and self.font == other.font
(4)                 def __str__(self):
(8)                     """Just retun the text"""
(8)                     return self.text
(4)                 def __repr__(self):
(8)                     font = self.font != InlineFont() and self.font or "default"
(8)                     return f"{self.__class__.__name__} text={self.text}, font={font}"
(4)                 def to_tree(self):
(8)                     el = Element("r")
(8)                     el.append(self.font.to_tree(tagname="rPr"))
(8)                     t = Element("t")
(8)                     t.text = self.text
(8)                     whitespace(t)
(8)                     el.append(t)
(8)                     return el
(0)             class CellRichText(list):
```

```
39: (4)                 """Represents a rich text string.
40: (4)                 Initialize with a list made of pure strings or :class:`TextBlock` elements
41: (4)                 Can index object to access or modify individual rich text elements
42: (4)                 it also supports the + and += operators between rich text strings
43: (4)                 There are no user methods for this class
44: (4)                 operations which modify the string will generally call an optimization
pass afterwards,
45: (4)                 that merges text blocks with identical formats, consecutive pure text
strings,
46: (4)                 and remove empty strings and empty text blocks
47: (4)                 """
48: (4)                 def __init__(self, *args):
49: (8)                     if len(args) == 1:
50: (12)                        args = args[0]
51: (12)                        if isinstance(args, (list, tuple)):
52: (16)                            CellRichText._check_rich_text(args)
53: (12)                        else:
54: (16)                            CellRichText._check_element(args)
55: (16)                            args = [args]
56: (8)                     else:
57: (12)                        CellRichText._check_rich_text(args)
58: (8)                     super().__init__(args)
59: (4)                 @classmethod
60: (4)                 def _check_element(cls, value):
61: (8)                     if not isinstance(value, (str, TextBlock, NUMERIC_TYPES)):
62: (12)                        raise TypeError(f"Illegal CellRichText element {value}")
63: (4)                 @classmethod
64: (4)                 def _check_rich_text(cls, rich_text):
65: (8)                     for t in rich_text:
66: (12)                        CellRichText._check_element(t)
67: (4)                 @classmethod
68: (4)                 def from_tree(cls, node):
69: (8)                     text = Text.from_tree(node)
70: (8)                     if text.t:
71: (12)                        return (text.t.replace('x005F_', ''),)
72: (8)                     s = []
73: (8)                     for r in text.r:
74: (12)                        t = ""
75: (12)                        if r.t:
76: (16)                            t = r.t.replace('x005F_', '')
77: (12)                        if r.rPr:
78: (16)                            s.append(TextBlock(r.rPr, t))
79: (12)                        else:
80: (16)                            s.append(t)
81: (8)                     return cls(s)
82: (4)                 def _opt(self):
83: (8)                     last_t = None
84: (8)                     l = CellRichText(tuple())
85: (8)                     for t in self:
86: (12)                        if isinstance(t, str):
87: (16)                            if not t:
88: (20)                                continue
89: (12)                        elif not t.text:
90: (16)                            continue
91: (12)                        if type(last_t) == type(t):
92: (16)                            if isinstance(t, str):
93: (20)                                last_t += t
94: (20)                                continue
95: (16)                            elif last_t.font == t.font:
96: (20)                                last_t.text += t.text
97: (20)                                continue
98: (12)                        if last_t:
99: (16)                            l.append(last_t)
100: (12)                       last_t = t
101: (8)                    if last_t:
102: (12)                       l.append(last_t)
103: (8)                    super().__setitem__(slice(None), l)
104: (8)                    return self
105: (4)                def __iadd__(self, arg):
```

```
106: (8)                        CellRichText._check_rich_text(arg)
107: (8)                        super().__iadd__([copy(e) for e in list(arg)])
108: (8)                        return self._opt()
109: (4)                    def __add__(self, arg):
110: (8)                        return CellRichText([copy(e) for e in list(self) + list(arg)])._opt()
111: (4)                    def __setitem__(self, indx, val):
112: (8)                        CellRichText._check_element(val)
113: (8)                        super().__setitem__(indx, val)
114: (8)                        self._opt()
115: (4)                    def append(self, arg):
116: (8)                        CellRichText._check_element(arg)
117: (8)                        super().append(arg)
118: (4)                    def extend(self, arg):
119: (8)                        CellRichText._check_rich_text(arg)
120: (8)                        super().extend(arg)
121: (4)                    def __repr__(self):
122: (8)                        return "CellRichText([{}])".format(', '.join((repr(s) for s in self)))
123: (4)                    def __str__(self):
124: (8)                        return ''.join([str(s) for s in self])
125: (4)                    def as_list(self):
126: (8)                        """
127: (8)                        Returns a list of the strings contained.
128: (8)                        The main reason for this is to make editing easier.
129: (8)                        """
130: (8)                        return [str(s) for s in self]
131: (4)                    def to_tree(self):
132: (8)                        """
133: (8)                        Return the full XML representation
134: (8)                        """
135: (8)                        container = Element("is")
136: (8)                        for obj in self:
137: (12)                           if isinstance(obj, TextBlock):
138: (16)                               container.append(obj.to_tree())
139: (12)                           else:
140: (16)                               el = Element("r")
141: (16)                               t = Element("t")
142: (16)                               t.text = obj
143: (16)                               whitespace(t)
144: (16)                               el.append(t)
145: (16)                               container.append(el)
146: (8)                        return container

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

File 12 - bar_chart.py:

1: (0)                 from openpyxl.descriptors.serialisable import Serialisable
2: (0)                 from openpyxl.descriptors import (
3: (4)                     Typed,
4: (4)                     Bool,
5: (4)                     Integer,
6: (4)                     Sequence,
7: (4)                     Alias,
8: (0)                 )
9: (0)                 from openpyxl.descriptors.excel import ExtensionList
10: (0)                from openpyxl.descriptors.nested import (
11: (4)                    NestedNoneSet,
12: (4)                    NestedSet,
13: (4)                    NestedBool,
14: (4)                    NestedInteger,
15: (4)                    NestedMinMax,
16: (0)                )
17: (0)                from .descriptors import (
18: (4)                    NestedGapAmount,
19: (4)                    NestedOverlap,
20: (0)                )
21: (0)                from ._chart import ChartBase
22: (0)                from ._3d import _3DBase
23: (0)                from .axis import TextAxis, NumericAxis, SeriesAxis, ChartLines
```

```
  24: (0)             from .shapes import GraphicalProperties
  25: (0)             from .series import Series
  26: (0)             from .legend import Legend
  27: (0)             from .label import DataLabelList
  28: (0)             class _BarChartBase(ChartBase):
  29: (4)                 barDir = NestedSet(values=(['bar', 'col']))
  30: (4)                 type = Alias("barDir")
  31: (4)                 grouping = NestedSet(values=(['percentStacked', 'clustered', 'standard',
  32: (34)                                                'stacked']))
  33: (4)                 varyColors = NestedBool(nested=True, allow_none=True)
  34: (4)                 ser = Sequence(expected_type=Series, allow_none=True)
  35: (4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
  36: (4)                 dataLabels = Alias("dLbls")
  37: (4)                 __elements__ = ('barDir', 'grouping', 'varyColors', 'ser', 'dLbls')
  38: (4)                 _series_type = "bar"
  39: (4)                 def __init__(self,
  40: (17)                             barDir="col",
  41: (17)                             grouping="clustered",
  42: (17)                             varyColors=None,
  43: (17)                             ser=(),
  44: (17)                             dLbls=None,
  45: (17)                             **kw
  46: (16)                            ):
  47: (8)                     self.barDir = barDir
  48: (8)                     self.grouping = grouping
  49: (8)                     self.varyColors = varyColors
  50: (8)                     self.ser = ser
  51: (8)                     self.dLbls = dLbls
  52: (8)                     super().__init__(**kw)
  53: (0)             class BarChart(_BarChartBase):
  54: (4)                 tagname = "barChart"
  55: (4)                 barDir = _BarChartBase.barDir
  56: (4)                 grouping = _BarChartBase.grouping
  57: (4)                 varyColors = _BarChartBase.varyColors
  58: (4)                 ser = _BarChartBase.ser
  59: (4)                 dLbls = _BarChartBase.dLbls
  60: (4)                 gapWidth = NestedGapAmount()
  61: (4)                 overlap = NestedOverlap()
  62: (4)                 serLines = Typed(expected_type=ChartLines, allow_none=True)
  63: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
  64: (4)                 x_axis = Typed(expected_type=TextAxis)
  65: (4)                 y_axis = Typed(expected_type=NumericAxis)
  66: (4)                 __elements__ = _BarChartBase.__elements__ + ('gapWidth', 'overlap',
'serLines', 'axId')
  67: (4)                 def __init__(self,
  68: (17)                             gapWidth=150,
  69: (17)                             overlap=None,
  70: (17)                             serLines=None,
  71: (17)                             extLst=None,
  72: (17)                             **kw
  73: (16)                            ):
  74: (8)                     self.gapWidth = gapWidth
  75: (8)                     self.overlap = overlap
  76: (8)                     self.serLines = serLines
  77: (8)                     self.x_axis = TextAxis()
  78: (8)                     self.y_axis = NumericAxis()
  79: (8)                     self.legend = Legend()
  80: (8)                     super().__init__(**kw)
  81: (0)             class BarChart3D(_BarChartBase, _3DBase):
  82: (4)                 tagname = "bar3DChart"
  83: (4)                 barDir = _BarChartBase.barDir
  84: (4)                 grouping = _BarChartBase.grouping
  85: (4)                 varyColors = _BarChartBase.varyColors
  86: (4)                 ser = _BarChartBase.ser
  87: (4)                 dLbls = _BarChartBase.dLbls
  88: (4)                 view3D = _3DBase.view3D
  89: (4)                 floor = _3DBase.floor
  90: (4)                 sideWall = _3DBase.sideWall
  91: (4)                 backWall = _3DBase.backWall
```

```
92: (4)                 gapWidth = NestedGapAmount()
93: (4)                 gapDepth = NestedGapAmount()
94: (4)                 shape = NestedNoneSet(values=(['cone', 'coneToMax', 'box', 'cylinder',
'pyramid', 'pyramidToMax']))
95: (4)                 serLines = Typed(expected_type=ChartLines, allow_none=True)
96: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
97: (4)                 x_axis = Typed(expected_type=TextAxis)
98: (4)                 y_axis = Typed(expected_type=NumericAxis)
99: (4)                 z_axis = Typed(expected_type=SeriesAxis, allow_none=True)
100: (4)                __elements__ = _BarChartBase.__elements__ + ('gapWidth', 'gapDepth',
'shape', 'serLines', 'axId')
101: (4)                def __init__(self,
102: (17)                            gapWidth=150,
103: (17)                            gapDepth=150,
104: (17)                            shape=None,
105: (17)                            serLines=None,
106: (17)                            extLst=None,
107: (17)                            **kw
108: (16)                           ):
109: (8)                    self.gapWidth = gapWidth
110: (8)                    self.gapDepth = gapDepth
111: (8)                    self.shape = shape
112: (8)                    self.serLines = serLines
113: (8)                    self.x_axis = TextAxis()
114: (8)                    self.y_axis = NumericAxis()
115: (8)                    self.z_axis = SeriesAxis()
116: (8)                    super(BarChart3D, self).__init__(**kw)

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

File 13 - _constants.py:

1: (0)              """
2: (0)              Package metadata
3: (0)              """
4: (0)              __author__ = "See AUTHORS"
5: (0)              __author_email__ = "charlie.clark@clark-consulting.eu"
6: (0)              __license__ = "MIT"
7: (0)              __maintainer_email__ = "openpyxl-users@googlegroups.com"
8: (0)              __url__ = "https://openpyxl.readthedocs.io"
9: (0)              __version__ = "3.1.5"
10: (0)             __python__ = "3.8"

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

File 14 - area_chart.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Set,
5: (4)                  Bool,
6: (4)                  Integer,
7: (4)                  Sequence,
8: (4)                  Alias,
9: (0)              )
10: (0)             from openpyxl.descriptors.excel import ExtensionList
11: (0)             from openpyxl.descriptors.nested import (
12: (4)                 NestedMinMax,
13: (4)                 NestedSet,
14: (4)                 NestedBool,
15: (0)             )
16: (0)             from ._chart import ChartBase
17: (0)             from .descriptors import NestedGapAmount
18: (0)             from .axis import TextAxis, NumericAxis, SeriesAxis, ChartLines
19: (0)             from .label import DataLabelList
20: (0)             from .series import Series
21: (0)             class _AreaChartBase(ChartBase):
22: (4)                 grouping = NestedSet(values=(['percentStacked', 'standard', 'stacked']))
```

```
23: (4)                 varyColors = NestedBool(nested=True, allow_none=True)
24: (4)                 ser = Sequence(expected_type=Series, allow_none=True)
25: (4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
26: (4)                 dataLabels = Alias("dLbls")
27: (4)                 dropLines = Typed(expected_type=ChartLines, allow_none=True)
28: (4)                 _series_type = "area"
29: (4)                 __elements__ = ('grouping', 'varyColors', 'ser', 'dLbls', 'dropLines')
30: (4)                 def __init__(self,
31: (17)                             grouping="standard",
32: (17)                             varyColors=None,
33: (17)                             ser=(),
34: (17)                             dLbls=None,
35: (17)                             dropLines=None,
36: (16)                            ):
37: (8)                     self.grouping = grouping
38: (8)                     self.varyColors = varyColors
39: (8)                     self.ser = ser
40: (8)                     self.dLbls = dLbls
41: (8)                     self.dropLines = dropLines
42: (8)                     super().__init__()
43: (0)             class AreaChart(_AreaChartBase):
44: (4)                 tagname = "areaChart"
45: (4)                 grouping = _AreaChartBase.grouping
46: (4)                 varyColors = _AreaChartBase.varyColors
47: (4)                 ser = _AreaChartBase.ser
48: (4)                 dLbls = _AreaChartBase.dLbls
49: (4)                 dropLines = _AreaChartBase.dropLines
50: (4)                 x_axis = Typed(expected_type=TextAxis)
51: (4)                 y_axis = Typed(expected_type=NumericAxis)
52: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
53: (4)                 __elements__ = _AreaChartBase.__elements__ + ('axId',)
54: (4)                 def __init__(self,
55: (17)                             axId=None,
56: (17)                             extLst=None,
57: (17)                             **kw
58: (16)                            ):
59: (8)                     self.x_axis = TextAxis()
60: (8)                     self.y_axis = NumericAxis()
61: (8)                     super().__init__(**kw)
62: (0)             class AreaChart3D(AreaChart):
63: (4)                 tagname = "area3DChart"
64: (4)                 grouping = _AreaChartBase.grouping
65: (4)                 varyColors = _AreaChartBase.varyColors
66: (4)                 ser = _AreaChartBase.ser
67: (4)                 dLbls = _AreaChartBase.dLbls
68: (4)                 dropLines = _AreaChartBase.dropLines
69: (4)                 gapDepth = NestedGapAmount()
70: (4)                 x_axis = Typed(expected_type=TextAxis)
71: (4)                 y_axis = Typed(expected_type=NumericAxis)
72: (4)                 z_axis = Typed(expected_type=SeriesAxis, allow_none=True)
73: (4)                 __elements__ = AreaChart.__elements__ + ('gapDepth', )
74: (4)                 def __init__(self, gapDepth=None, **kw):
75: (8)                     self.gapDepth = gapDepth
76: (8)                     super(AreaChart3D, self).__init__(**kw)
77: (8)                     self.x_axis = TextAxis()
78: (8)                     self.y_axis = NumericAxis()
79: (8)                     self.z_axis = SeriesAxis()

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

File 15 - chartspace.py:

1: (0)              """
2: (0)              Enclosing chart object. The various chart types are actually child objects.
3: (0)              Will probably need to call this indirectly
4: (0)              """
5: (0)              from openpyxl.descriptors.serialisable import Serialisable
6: (0)              from openpyxl.descriptors import (
7: (4)                  Typed,
8: (4)                  String,
9: (4)                  Alias,
10: (0)             )
11: (0)             from openpyxl.descriptors.excel import (
12: (4)                 ExtensionList,
13: (4)                 Relation
14: (0)             )
15: (0)             from openpyxl.descriptors.nested import (
16: (4)                 NestedBool,
17: (4)                 NestedNoneSet,
18: (4)                 NestedString,
19: (4)                 NestedMinMax,
20: (0)             )
21: (0)             from openpyxl.descriptors.sequence import NestedSequence
22: (0)             from openpyxl.xml.constants import CHART_NS
23: (0)             from openpyxl.drawing.colors import ColorMapping
24: (0)             from .text import RichText
25: (0)             from .shapes import GraphicalProperties
26: (0)             from .legend import Legend
27: (0)             from ._3d import _3DBase
28: (0)             from .plotarea import PlotArea
29: (0)             from .title import Title
30: (0)             from .pivot import (
31: (4)                 PivotFormat,
32: (4)                 PivotSource,
33: (0)             )
34: (0)             from .print_settings import PrintSettings
35: (0)             class ChartContainer(Serialisable):
36: (4)                 tagname = "chart"
37: (4)                 title = Typed(expected_type=Title, allow_none=True)
38: (4)                 autoTitleDeleted = NestedBool(allow_none=True)
39: (4)                 pivotFmts = NestedSequence(expected_type=PivotFormat)
40: (4)                 view3D = _3DBase.view3D
41: (4)                 floor = _3DBase.floor
42: (4)                 sideWall = _3DBase.sideWall
43: (4)                 backWall = _3DBase.backWall
44: (4)                 plotArea = Typed(expected_type=PlotArea, )
45: (4)                 legend = Typed(expected_type=Legend, allow_none=True)
46: (4)                 plotVisOnly = NestedBool()
47: (4)                 dispBlanksAs = NestedNoneSet(values=(['span', 'gap', 'zero']))
48: (4)                 showDLblsOverMax = NestedBool(allow_none=True)
49: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
50: (4)                 __elements__ = ('title', 'autoTitleDeleted', 'pivotFmts', 'view3D',
51: (20)                                'floor', 'sideWall', 'backWall', 'plotArea', 'legend',
'plotVisOnly',
52: (20)                                'dispBlanksAs', 'showDLblsOverMax')
53: (4)                 def __init__(self,
54: (17)                             title=None,
55: (17)                             autoTitleDeleted=None,
56: (17)                             pivotFmts=(),
57: (17)                             view3D=None,
58: (17)                             floor=None,
59: (17)                             sideWall=None,
60: (17)                             backWall=None,
61: (17)                             plotArea=None,
62: (17)                             legend=None,
63: (17)                             plotVisOnly=True,
64: (17)                             dispBlanksAs="gap",
65: (17)                             showDLblsOverMax=None,
66: (17)                             extLst=None,
67: (16)                            ):
68: (8)                     self.title = title
69: (8)                     self.autoTitleDeleted = autoTitleDeleted
70: (8)                     self.pivotFmts = pivotFmts
71: (8)                     self.view3D = view3D
72: (8)                     self.floor = floor
73: (8)                     self.sideWall = sideWall
74: (8)                     self.backWall = backWall
75: (8)                     if plotArea is None:
```

```
(12)                    plotArea = PlotArea()
(8)                 self.plotArea = plotArea
(8)                 self.legend = legend
(8)                 self.plotVisOnly = plotVisOnly
(8)                 self.dispBlanksAs = dispBlanksAs
(8)                 self.showDLblsOverMax = showDLblsOverMax
(0)         class Protection(Serialisable):
(4)             tagname = "protection"
(4)             chartObject = NestedBool(allow_none=True)
(4)             data = NestedBool(allow_none=True)
(4)             formatting = NestedBool(allow_none=True)
(4)             selection = NestedBool(allow_none=True)
(4)             userInterface = NestedBool(allow_none=True)
(4)             __elements__ = ("chartObject", "data", "formatting", "selection", "userInterface")
(4)             def __init__(self,
(17)                         chartObject=None,
(17)                         data=None,
(17)                         formatting=None,
(17)                         selection=None,
(17)                         userInterface=None,
(16)                        ):
(8)                 self.chartObject = chartObject
(8)                 self.data = data
(8)                 self.formatting = formatting
(8)                 self.selection = selection
(8)                 self.userInterface = userInterface
(0)         class ExternalData(Serialisable):
(4)             tagname = "externalData"
(4)             autoUpdate = NestedBool(allow_none=True)
(4)             id = String() # Needs namespace
(4)             def __init__(self,
(17)                         autoUpdate=None,
(17)                         id=None
(16)                        ):
(8)                 self.autoUpdate = autoUpdate
(8)                 self.id = id
(0)         class ChartSpace(Serialisable):
(4)             tagname = "chartSpace"
(4)             date1904 = NestedBool(allow_none=True)
(4)             lang = NestedString(allow_none=True)
(4)             roundedCorners = NestedBool(allow_none=True)
(4)             style = NestedMinMax(allow_none=True, min=1, max=48)
(4)             clrMapOvr = Typed(expected_type=ColorMapping, allow_none=True)
(4)             pivotSource = Typed(expected_type=PivotSource, allow_none=True)
(4)             protection = Typed(expected_type=Protection, allow_none=True)
(4)             chart = Typed(expected_type=ChartContainer)
(4)             spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)             graphical_properties = Alias("spPr")
(4)             txPr = Typed(expected_type=RichText, allow_none=True)
(4)             textProperties = Alias("txPr")
(4)             externalData = Typed(expected_type=ExternalData, allow_none=True)
(4)             printSettings = Typed(expected_type=PrintSettings, allow_none=True)
(4)             userShapes = Relation()
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)             __elements__ = ('date1904', 'lang', 'roundedCorners', 'style',
(20)                            'clrMapOvr', 'pivotSource', 'protection', 'chart', 'spPr', 'txPr',
(20)                            'externalData', 'printSettings', 'userShapes')
(4)             def __init__(self,
(17)                         date1904=None,
(17)                         lang=None,
(17)                         roundedCorners=None,
(17)                         style=None,
(17)                         clrMapOvr=None,
(17)                         pivotSource=None,
(17)                         protection=None,
(17)                         chart=None,
(17)                         spPr=None,
143: (17)                    txPr=None,
144: (17)                    externalData=None,
145: (17)                    printSettings=None,
146: (17)                    userShapes=None,
147: (17)                    extLst=None,
148: (16)                   ):
149: (8)            self.date1904 = date1904
150: (8)            self.lang = lang
151: (8)            self.roundedCorners = roundedCorners
152: (8)            self.style = style
153: (8)            self.clrMapOvr = clrMapOvr
154: (8)            self.pivotSource = pivotSource
155: (8)            self.protection = protection
156: (8)            self.chart = chart
157: (8)            self.spPr = spPr
158: (8)            self.txPr = txPr
159: (8)            self.externalData = externalData
160: (8)            self.printSettings = printSettings
161: (8)            self.userShapes = userShapes
162: (4)        def to_tree(self, tagname=None, idx=None, namespace=None):
163: (8)            tree = super().to_tree()
164: (8)            tree.set("xmlns", CHART_NS)
165: (8)            return tree

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

File 16 - data_source.py:

1: (0)      """
2: (0)      Collection of utility primitives for charts.
3: (0)      """
4: (0)      from openpyxl.descriptors.serialisable import Serialisable
5: (0)      from openpyxl.descriptors import (
6: (4)          Bool,
7: (4)          Typed,
8: (4)          Alias,
9: (4)          String,
10: (4)         Integer,
11: (4)         Sequence,
12: (0)     )
13: (0)     from openpyxl.descriptors.excel import ExtensionList
14: (0)     from openpyxl.descriptors.nested import (
15: (4)         NestedString,
16: (4)         NestedText,
17: (4)         NestedInteger,
18: (0)     )
19: (0)     class NumFmt(Serialisable):
20: (4)         formatCode = String()
21: (4)         sourceLinked = Bool()
22: (4)         def __init__(self,
23: (17)                     formatCode=None,
24: (17)                     sourceLinked=False
25: (16)                    ):
26: (8)             self.formatCode = formatCode
27: (8)             self.sourceLinked = sourceLinked
28: (0)     class NumberValueDescriptor(NestedText):
29: (4)         """
30: (4)         Data should be numerical but isn't always :-/
31: (4)         """
32: (4)         allow_none = True
33: (4)         def __set__(self, instance, value):
34: (8)             if value == "#N/A":
35: (12)                self.expected_type = str
36: (8)             else:
37: (12)                self.expected_type = float
38: (8)             super().__set__(instance, value)
39: (0)     class NumVal(Serialisable):
40: (4)         idx = Integer()
41: (4)         formatCode = NestedText(allow_none=True, expected_type=str)
```

```
    v = NumberValueDescriptor()
    def __init__(self,
                 idx=None,
                 formatCode=None,
                 v=None,
                ):
        self.idx = idx
        self.formatCode = formatCode
        self.v = v
class NumData(Serialisable):
    formatCode = NestedText(expected_type=str, allow_none=True)
    ptCount = NestedInteger(allow_none=True)
    pt = Sequence(expected_type=NumVal)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('formatCode', 'ptCount', 'pt')
    def __init__(self,
                 formatCode=None,
                 ptCount=None,
                 pt=(),
                 extLst=None,
                ):
        self.formatCode = formatCode
        self.ptCount = ptCount
        self.pt = pt
class NumRef(Serialisable):
    f = NestedText(expected_type=str)
    ref = Alias('f')
    numCache = Typed(expected_type=NumData, allow_none=True)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('f', 'numCache')
    def __init__(self,
                 f=None,
                 numCache=None,
                 extLst=None,
                ):
        self.f = f
        self.numCache = numCache
class StrVal(Serialisable):
    tagname = "strVal"
    idx = Integer()
    v = NestedText(expected_type=str)
    def __init__(self,
                 idx=0,
                 v=None,
                ):
        self.idx = idx
        self.v = v
class StrData(Serialisable):
    tagname = "strData"
    ptCount = NestedInteger(allow_none=True)
    pt = Sequence(expected_type=StrVal)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('ptCount', 'pt')
    def __init__(self,
                 ptCount=None,
                 pt=(),
                 extLst=None,
                ):
        self.ptCount = ptCount
        self.pt = pt
class StrRef(Serialisable):
    tagname = "strRef"
    f = NestedText(expected_type=str, allow_none=True)
    strCache = Typed(expected_type=StrData, allow_none=True)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('f', 'strCache')
    def __init__(self,
                 f=None,
                 strCache=None,
111: (17)                    extLst=None,
112: (16)                   ):
113: (8)            self.f = f
114: (8)            self.strCache = strCache
115: (0)    class NumDataSource(Serialisable):
116: (4)        numRef = Typed(expected_type=NumRef, allow_none=True)
117: (4)        numLit = Typed(expected_type=NumData, allow_none=True)
118: (4)        def __init__(self,
119: (17)                    numRef=None,
120: (17)                    numLit=None,
121: (17)                    ):
122: (8)            self.numRef = numRef
123: (8)            self.numLit = numLit
124: (0)    class Level(Serialisable):
125: (4)        tagname = "lvl"
126: (4)        pt = Sequence(expected_type=StrVal)
127: (4)        __elements__ = ('pt',)
128: (4)        def __init__(self,
129: (17)                    pt=(),
130: (16)                   ):
131: (8)            self.pt = pt
132: (0)    class MultiLevelStrData(Serialisable):
133: (4)        tagname = "multiLvlStrData"
134: (4)        ptCount = Integer(allow_none=True)
135: (4)        lvl = Sequence(expected_type=Level)
136: (4)        extLst = Typed(expected_type=ExtensionList, allow_none=True)
137: (4)        __elements__ = ('ptCount', 'lvl',)
138: (4)        def __init__(self,
139: (17)                    ptCount=None,
140: (17)                    lvl=(),
141: (17)                    extLst=None,
142: (16)                   ):
143: (8)            self.ptCount = ptCount
144: (8)            self.lvl = lvl
145: (0)    class MultiLevelStrRef(Serialisable):
146: (4)        tagname = "multiLvlStrRef"
147: (4)        f = NestedText(expected_type=str)
148: (4)        multiLvlStrCache = Typed(expected_type=MultiLevelStrData, allow_none=True)
149: (4)        extLst = Typed(expected_type=ExtensionList, allow_none=True)
150: (4)        __elements__ = ('multiLvlStrCache', 'f')
151: (4)        def __init__(self,
152: (17)                    f=None,
153: (17)                    multiLvlStrCache=None,
154: (17)                    extLst=None,
155: (16)                   ):
156: (8)            self.f = f
157: (8)            self.multiLvlStrCache = multiLvlStrCache
158: (0)    class AxDataSource(Serialisable):
159: (4)        tagname = "cat"
160: (4)        numRef = Typed(expected_type=NumRef, allow_none=True)
161: (4)        numLit = Typed(expected_type=NumData, allow_none=True)
162: (4)        strRef = Typed(expected_type=StrRef, allow_none=True)
163: (4)        strLit = Typed(expected_type=StrData, allow_none=True)
164: (4)        multiLvlStrRef = Typed(expected_type=MultiLevelStrRef, allow_none=True)
165: (4)        def __init__(self,
166: (17)                    numRef=None,
167: (17)                    numLit=None,
168: (17)                    strRef=None,
169: (17)                    strLit=None,
170: (17)                    multiLvlStrRef=None,
171: (17)                    ):
172: (8)            if not any([numLit, numRef, strRef, strLit, multiLvlStrRef]):
173: (12)               raise TypeError("A data source must be provided")
174: (8)            self.numRef = numRef
175: (8)            self.numLit = numLit
176: (8)            self.strRef = strRef
177: (8)            self.strLit = strLit
178: (8)            self.multiLvlStrRef = multiLvlStrRef
```

```
----------------------------------------

File 17 - descriptors.py:

1: (0)              from openpyxl.descriptors.nested import (
2: (4)                  NestedMinMax
3: (4)                  )
4: (0)              from openpyxl.descriptors import Typed
5: (0)              from .data_source import NumFmt
6: (0)              """
7: (0)              Utility descriptors for the chart module.
8: (0)              For convenience but also clarity.
9: (0)              """
10: (0)             class NestedGapAmount(NestedMinMax):
11: (4)                 allow_none = True
12: (4)                 min = 0
13: (4)                 max = 500
14: (0)             class NestedOverlap(NestedMinMax):
15: (4)                 allow_none = True
16: (4)                 min = -100
17: (4)                 max = 100
18: (0)             class NumberFormatDescriptor(Typed):
19: (4)                 """
20: (4)                 Allow direct assignment of format code
21: (4)                 """
22: (4)                 expected_type = NumFmt
23: (4)                 allow_none = True
24: (4)                 def __set__(self, instance, value):
25: (8)                     if isinstance(value, str):
26: (12)                        value = NumFmt(value)
27: (8)                     super().__set__(instance, value)

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

File 18 - bubble_chart.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Set,
5: (4)                  MinMax,
6: (4)                  Bool,
7: (4)                  Integer,
8: (4)                  Alias,
9: (4)                  Sequence,
10: (0)             )
11: (0)             from openpyxl.descriptors.excel import ExtensionList
12: (0)             from openpyxl.descriptors.nested import (
13: (4)                 NestedNoneSet,
14: (4)                 NestedMinMax,
15: (4)                 NestedBool,
16: (0)             )
17: (0)             from ._chart import ChartBase
18: (0)             from .axis import TextAxis, NumericAxis
19: (0)             from .series import XYSeries
20: (0)             from .label import DataLabelList
21: (0)             class BubbleChart(ChartBase):
22: (4)                 tagname = "bubbleChart"
23: (4)                 varyColors = NestedBool(allow_none=True)
24: (4)                 ser = Sequence(expected_type=XYSeries, allow_none=True)
25: (4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
26: (4)                 dataLabels = Alias("dLbls")
27: (4)                 bubble3D = NestedBool(allow_none=True)
28: (4)                 bubbleScale = NestedMinMax(min=0, max=300, allow_none=True)
29: (4)                 showNegBubbles = NestedBool(allow_none=True)
30: (4)                 sizeRepresents = NestedNoneSet(values=(['area', 'w']))
31: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
32: (4)                 x_axis = Typed(expected_type=NumericAxis)
33: (4)                 y_axis = Typed(expected_type=NumericAxis)
```

```
34: (4)             _series_type = "bubble"
35: (4)             __elements__ = ('varyColors', 'ser', 'dLbls', 'bubble3D', 'bubbleScale',
36: (20)                            'showNegBubbles', 'sizeRepresents', 'axId')
37: (4)             def __init__(self,
38: (17)                         varyColors=None,
39: (17)                         ser=(),
40: (17)                         dLbls=None,
41: (17)                         bubble3D=None,
42: (17)                         bubbleScale=None,
43: (17)                         showNegBubbles=None,
44: (17)                         sizeRepresents=None,
45: (17)                         extLst=None,
46: (17)                         **kw
47: (16)                        ):
48: (8)                 self.varyColors = varyColors
49: (8)                 self.ser = ser
50: (8)                 self.dLbls = dLbls
51: (8)                 self.bubble3D = bubble3D
52: (8)                 self.bubbleScale = bubbleScale
53: (8)                 self.showNegBubbles = showNegBubbles
54: (8)                 self.sizeRepresents = sizeRepresents
55: (8)                 self.x_axis = NumericAxis(axId=10, crossAx=20)
56: (8)                 self.y_axis = NumericAxis(axId=20, crossAx=10)
57: (8)                 super().__init__(**kw)

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

File 19 - text.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Alias,
5: (4)                  Sequence,
6: (0)              )
7: (0)              from openpyxl.drawing.text import (
8: (4)                  RichTextProperties,
9: (4)                  ListStyle,
10: (4)                 Paragraph,
11: (0)             )
12: (0)             from .data_source import StrRef
13: (0)             class RichText(Serialisable):
14: (4)                 """
15: (4)                 From the specification: 21.2.2.216
16: (4)                 This element specifies text formatting. The lstStyle element is not
supported.
17: (4)                 """
18: (4)                 tagname = "rich"
19: (4)                 bodyPr = Typed(expected_type=RichTextProperties)
20: (4)                 properties = Alias("bodyPr")
21: (4)                 lstStyle = Typed(expected_type=ListStyle, allow_none=True)
22: (4)                 p = Sequence(expected_type=Paragraph)
23: (4)                 paragraphs = Alias('p')
24: (4)                 __elements__ = ("bodyPr", "lstStyle", "p")
25: (4)                 def __init__(self,
26: (17)                             bodyPr=None,
27: (17)                             lstStyle=None,
28: (17)                             p=None,
29: (16)                            ):
30: (8)                     if bodyPr is None:
31: (12)                        bodyPr = RichTextProperties()
32: (8)                     self.bodyPr = bodyPr
33: (8)                     self.lstStyle = lstStyle
34: (8)                     if p is None:
35: (12)                        p = [Paragraph()]
36: (8)                     self.p = p
37: (0)             class Text(Serialisable):
38: (4)                 """
39: (4)                 The value can be either a cell reference or a text element
```

```
40: (4)                 If both are present then the reference will be used.
41: (4)                 """
42: (4)                 tagname = "tx"
43: (4)                 strRef = Typed(expected_type=StrRef, allow_none=True)
44: (4)                 rich = Typed(expected_type=RichText, allow_none=True)
45: (4)                 __elements__ = ("strRef", "rich")
46: (4)                 def __init__(self,
47: (17)                             strRef=None,
48: (17)                             rich=None
49: (17)                             ):
50: (8)                     self.strRef = strRef
51: (8)                     if rich is None:
52: (12)                        rich = RichText()
53: (8)                     self.rich = rich
54: (4)                 def to_tree(self, tagname=None, idx=None, namespace=None):
55: (8)                     if self.strRef and self.rich:
56: (12)                        self.rich = None # can only have one
57: (8)                     return super().to_tree(tagname, idx, namespace)

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

File 20 - label.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence,
4: (4)                  Alias,
5: (4)                  Typed
6: (0)              )
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from openpyxl.descriptors.nested import (
9: (4)                  NestedNoneSet,
10: (4)                 NestedBool,
11: (4)                 NestedString,
12: (4)                 NestedInteger,
13: (4)                 )
14: (0)             from .shapes import GraphicalProperties
15: (0)             from .text import RichText
16: (0)             class _DataLabelBase(Serialisable):
17: (4)                 numFmt = NestedString(allow_none=True, attribute="formatCode")
18: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
19: (4)                 graphicalProperties = Alias('spPr')
20: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
21: (4)                 textProperties = Alias('txPr')
22: (4)                 dLblPos = NestedNoneSet(values=['bestFit', 'b', 'ctr', 'inBase', 'inEnd',
23: (36)                                                'l', 'outEnd', 'r', 't'])
24: (4)                 position = Alias('dLblPos')
25: (4)                 showLegendKey = NestedBool(allow_none=True)
26: (4)                 showVal = NestedBool(allow_none=True)
27: (4)                 showCatName = NestedBool(allow_none=True)
28: (4)                 showSerName = NestedBool(allow_none=True)
29: (4)                 showPercent = NestedBool(allow_none=True)
30: (4)                 showBubbleSize = NestedBool(allow_none=True)
31: (4)                 showLeaderLines = NestedBool(allow_none=True)
32: (4)                 separator = NestedString(allow_none=True)
33: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
34: (4)                 __elements__ = ("numFmt", "spPr", "txPr", "dLblPos", "showLegendKey",
35: (20)                                "showVal", "showCatName", "showSerName", "showPercent",
"showBubbleSize",
36: (20)                                "showLeaderLines", "separator")
37: (4)                 def __init__(self,
38: (17)                             numFmt=None,
39: (17)                             spPr=None,
40: (17)                             txPr=None,
41: (17)                             dLblPos=None,
42: (17)                             showLegendKey=None,
43: (17)                             showVal=None,
44: (17)                             showCatName=None,
45: (17)                             showSerName=None,
46: (17)                         showPercent=None,
47: (17)                         showBubbleSize=None,
48: (17)                         showLeaderLines=None,
49: (17)                         separator=None,
50: (17)                         extLst=None,
51: (17)                         ):
52: (8)                 self.numFmt = numFmt
53: (8)                 self.spPr = spPr
54: (8)                 self.txPr = txPr
55: (8)                 self.dLblPos = dLblPos
56: (8)                 self.showLegendKey = showLegendKey
57: (8)                 self.showVal = showVal
58: (8)                 self.showCatName = showCatName
59: (8)                 self.showSerName = showSerName
60: (8)                 self.showPercent = showPercent
61: (8)                 self.showBubbleSize = showBubbleSize
62: (8)                 self.showLeaderLines = showLeaderLines
63: (8)                 self.separator = separator
64: (0)         class DataLabel(_DataLabelBase):
65: (4)             tagname = "dLbl"
66: (4)             idx = NestedInteger()
67: (4)             numFmt = _DataLabelBase.numFmt
68: (4)             spPr = _DataLabelBase.spPr
69: (4)             txPr = _DataLabelBase.txPr
70: (4)             dLblPos = _DataLabelBase.dLblPos
71: (4)             showLegendKey = _DataLabelBase.showLegendKey
72: (4)             showVal = _DataLabelBase.showVal
73: (4)             showCatName = _DataLabelBase.showCatName
74: (4)             showSerName = _DataLabelBase.showSerName
75: (4)             showPercent = _DataLabelBase.showPercent
76: (4)             showBubbleSize = _DataLabelBase.showBubbleSize
77: (4)             showLeaderLines = _DataLabelBase.showLeaderLines
78: (4)             separator = _DataLabelBase.separator
79: (4)             extLst = _DataLabelBase.extLst
80: (4)             __elements__ = ("idx",)  + _DataLabelBase.__elements__
81: (4)             def __init__(self, idx=0, **kw ):
82: (8)                 self.idx = idx
83: (8)                 super().__init__(**kw)
84: (0)         class DataLabelList(_DataLabelBase):
85: (4)             tagname = "dLbls"
86: (4)             dLbl = Sequence(expected_type=DataLabel, allow_none=True)
87: (4)             delete = NestedBool(allow_none=True)
88: (4)             numFmt = _DataLabelBase.numFmt
89: (4)             spPr = _DataLabelBase.spPr
90: (4)             txPr = _DataLabelBase.txPr
91: (4)             dLblPos = _DataLabelBase.dLblPos
92: (4)             showLegendKey = _DataLabelBase.showLegendKey
93: (4)             showVal = _DataLabelBase.showVal
94: (4)             showCatName = _DataLabelBase.showCatName
95: (4)             showSerName = _DataLabelBase.showSerName
96: (4)             showPercent = _DataLabelBase.showPercent
97: (4)             showBubbleSize = _DataLabelBase.showBubbleSize
98: (4)             showLeaderLines = _DataLabelBase.showLeaderLines
99: (4)             separator = _DataLabelBase.separator
100: (4)            extLst = _DataLabelBase.extLst
101: (4)            __elements__ = ("delete", "dLbl",) + _DataLabelBase.__elements__
102: (4)            def __init__(self, dLbl=(), delete=None,  **kw):
103: (8)                self.dLbl = dLbl
104: (8)                self.delete = delete
105: (8)                super().__init__(**kw)

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

File 21 - pivot.py:

1: (0)          from openpyxl.descriptors.serialisable import Serialisable
2: (0)          from openpyxl.descriptors import (
3: (4)              Alias,
4: (4)              Typed,
5: (0)              )
6: (0)              from openpyxl.descriptors.nested import NestedInteger, NestedText
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from .label import DataLabel
9: (0)              from .marker import Marker
10: (0)             from .shapes import GraphicalProperties
11: (0)             from .text import RichText
12: (0)             class PivotSource(Serialisable):
13: (4)                 tagname = "pivotSource"
14: (4)                 name = NestedText(expected_type=str)
15: (4)                 fmtId = NestedInteger(expected_type=int)
16: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
17: (4)                 __elements__ = ('name', 'fmtId')
18: (4)                 def __init__(self,
19: (17)                             name=None,
20: (17)                             fmtId=None,
21: (17)                             extLst=None,
22: (16)                            ):
23: (8)                     self.name = name
24: (8)                     self.fmtId = fmtId
25: (0)             class PivotFormat(Serialisable):
26: (4)                 tagname = "pivotFmt"
27: (4)                 idx = NestedInteger(nested=True)
28: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
29: (4)                 graphicalProperties = Alias("spPr")
30: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
31: (4)                 TextBody = Alias("txPr")
32: (4)                 marker = Typed(expected_type=Marker, allow_none=True)
33: (4)                 dLbl = Typed(expected_type=DataLabel, allow_none=True)
34: (4)                 DataLabel = Alias("dLbl")
35: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
36: (4)                 __elements__ = ('idx', 'spPr', 'txPr', 'marker', 'dLbl')
37: (4)                 def __init__(self,
38: (17)                             idx=0,
39: (17)                             spPr=None,
40: (17)                             txPr=None,
41: (17)                             marker=None,
42: (17)                             dLbl=None,
43: (17)                             extLst=None,
44: (16)                            ):
45: (8)                     self.idx = idx
46: (8)                     self.spPr = spPr
47: (8)                     self.txPr = txPr
48: (8)                     self.marker = marker
49: (8)                     self.dLbl = dLbl

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

File 22 - title.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Alias,
5: (0)              )
6: (0)              from openpyxl.descriptors.excel import ExtensionList
7: (0)              from openpyxl.descriptors.nested import NestedBool
8: (0)              from .text import Text, RichText
9: (0)              from .layout import Layout
10: (0)             from .shapes import GraphicalProperties
11: (0)             from openpyxl.drawing.text import (
12: (4)                 Paragraph,
13: (4)                 RegularTextRun,
14: (4)                 LineBreak,
15: (4)                 ParagraphProperties,
16: (4)                 CharacterProperties,
17: (0)             )
18: (0)             class Title(Serialisable):
19: (4)                 tagname = "title"
```

```
20: (4)                 tx = Typed(expected_type=Text, allow_none=True)
21: (4)                 text = Alias('tx')
22: (4)                 layout = Typed(expected_type=Layout, allow_none=True)
23: (4)                 overlay = NestedBool(allow_none=True)
24: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
25: (4)                 graphicalProperties = Alias('spPr')
26: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
27: (4)                 body = Alias('txPr')
28: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
29: (4)                 __elements__ = ('tx', 'layout', 'overlay', 'spPr', 'txPr')
30: (4)                 def __init__(self,
31: (17)                             tx=None,
32: (17)                             layout=None,
33: (17)                             overlay=None,
34: (17)                             spPr=None,
35: (17)                             txPr=None,
36: (17)                             extLst=None,
37: (16)                            ):
38: (8)                     if tx is None:
39: (12)                        tx = Text()
40: (8)                     self.tx = tx
41: (8)                     self.layout = layout
42: (8)                     self.overlay = overlay
43: (8)                     self.spPr = spPr
44: (8)                     self.txPr = txPr
45: (0)             def title_maker(text):
46: (4)                 title = Title()
47: (4)                 paraprops = ParagraphProperties()
48: (4)                 paraprops.defRPr = CharacterProperties()
49: (4)                 paras = [Paragraph(r=[RegularTextRun(t=s)], pPr=paraprops) for s in
text.split("\n")]
50: (4)                 title.tx.rich.paragraphs = paras
51: (4)                 return title
52: (0)             class TitleDescriptor(Typed):
53: (4)                 expected_type = Title
54: (4)                 allow_none = True
55: (4)                 def __set__(self, instance, value):
56: (8)                     if isinstance(value, str):
57: (12)                        value = title_maker(value)
58: (8)                     super().__set__(instance, value)

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

File 23 - layout.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  NoneSet,
4: (4)                  Float,
5: (4)                  Typed,
6: (4)                  Alias,
7: (0)              )
8: (0)              from openpyxl.descriptors.excel import ExtensionList
9: (0)              from openpyxl.descriptors.nested import (
10: (4)                 NestedNoneSet,
11: (4)                 NestedSet,
12: (4)                 NestedMinMax,
13: (0)             )
14: (0)             class ManualLayout(Serialisable):
15: (4)                 tagname = "manualLayout"
16: (4)                 layoutTarget = NestedNoneSet(values=(['inner', 'outer']))
17: (4)                 xMode = NestedNoneSet(values=(['edge', 'factor']))
18: (4)                 yMode = NestedNoneSet(values=(['edge', 'factor']))
19: (4)                 wMode = NestedSet(values=(['edge', 'factor']))
20: (4)                 hMode = NestedSet(values=(['edge', 'factor']))
21: (4)                 x = NestedMinMax(min=-1, max=1, allow_none=True)
22: (4)                 y = NestedMinMax(min=-1, max=1, allow_none=True)
23: (4)                 w = NestedMinMax(min=0, max=1, allow_none=True)
24: (4)                 width = Alias('w')
```

```
25: (4)                 h = NestedMinMax(min=0, max=1,  allow_none=True)
26: (4)                 height = Alias('h')
27: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
28: (4)                 __elements__ = ('layoutTarget', 'xMode', 'yMode', 'wMode', 'hMode', 'x',
29: (20)                                'y', 'w', 'h')
30: (4)                 def __init__(self,
31: (17)                             layoutTarget=None,
32: (17)                             xMode=None,
33: (17)                             yMode=None,
34: (17)                             wMode="factor",
35: (17)                             hMode="factor",
36: (17)                             x=None,
37: (17)                             y=None,
38: (17)                             w=None,
39: (17)                             h=None,
40: (17)                             extLst=None,
41: (16)                            ):
42: (8)                     self.layoutTarget = layoutTarget
43: (8)                     self.xMode = xMode
44: (8)                     self.yMode = yMode
45: (8)                     self.wMode = wMode
46: (8)                     self.hMode = hMode
47: (8)                     self.x = x
48: (8)                     self.y = y
49: (8)                     self.w = w
50: (8)                     self.h = h
51: (0)             class Layout(Serialisable):
52: (4)                 tagname = "layout"
53: (4)                 manualLayout = Typed(expected_type=ManualLayout, allow_none=True)
54: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
55: (4)                 __elements__ = ('manualLayout',)
56: (4)                 def __init__(self,
57: (17)                             manualLayout=None,
58: (17)                             extLst=None,
59: (16)                            ):
60: (8)                     self.manualLayout = manualLayout

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

File 24 - legend.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Integer,
5: (4)                  Alias,
6: (4)                  Sequence,
7: (0)              )
8: (0)              from openpyxl.descriptors.excel import ExtensionList
9: (0)              from openpyxl.descriptors.nested import (
10: (4)                 NestedBool,
11: (4)                 NestedSet,
12: (4)                 NestedInteger
13: (0)             )
14: (0)             from .layout import Layout
15: (0)             from .shapes import GraphicalProperties
16: (0)             from .text import RichText
17: (0)             class LegendEntry(Serialisable):
18: (4)                 tagname = "legendEntry"
19: (4)                 idx = NestedInteger()
20: (4)                 delete = NestedBool()
21: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
22: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
23: (4)                 __elements__ = ('idx', 'delete', 'txPr')
24: (4)                 def __init__(self,
25: (17)                             idx=0,
26: (17)                             delete=False,
27: (17)                             txPr=None,
28: (17)                             extLst=None,
29: (16)                        ):
30: (8)                     self.idx = idx
31: (8)                     self.delete = delete
32: (8)                     self.txPr = txPr
33: (0)             class Legend(Serialisable):
34: (4)                 tagname = "legend"
35: (4)                 legendPos = NestedSet(values=(['b', 'tr', 'l', 'r', 't']))
36: (4)                 position = Alias('legendPos')
37: (4)                 legendEntry = Sequence(expected_type=LegendEntry)
38: (4)                 layout = Typed(expected_type=Layout, allow_none=True)
39: (4)                 overlay = NestedBool(allow_none=True)
40: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
41: (4)                 graphicalProperties = Alias('spPr')
42: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
43: (4)                 textProperties = Alias('txPr')
44: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
45: (4)                 __elements__ = ('legendPos', 'legendEntry', 'layout', 'overlay', 'spPr',
'txPr',)
46: (4)                 def __init__(self,
47: (17)                             legendPos="r",
48: (17)                             legendEntry=(),
49: (17)                             layout=None,
50: (17)                             overlay=None,
51: (17)                             spPr=None,
52: (17)                             txPr=None,
53: (17)                             extLst=None,
54: (16)                            ):
55: (8)                     self.legendPos = legendPos
56: (8)                     self.legendEntry = legendEntry
57: (8)                     self.layout = layout
58: (8)                     self.overlay = overlay
59: (8)                     self.spPr = spPr
60: (8)                     self.txPr = txPr

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

File 25 - marker.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Alias,
5: (0)              )
6: (0)              from openpyxl.descriptors.excel import(
7: (4)                  ExtensionList,
8: (4)                  _explicit_none,
9: (0)              )
10: (0)             from openpyxl.descriptors.nested import (
11: (4)                 NestedBool,
12: (4)                 NestedInteger,
13: (4)                 NestedMinMax,
14: (4)                 NestedNoneSet,
15: (0)             )
16: (0)             from .layout import Layout
17: (0)             from .picture import PictureOptions
18: (0)             from .shapes import *
19: (0)             from .text import *
20: (0)             from .error_bar import *
21: (0)             class Marker(Serialisable):
22: (4)                 tagname = "marker"
23: (4)                 symbol = NestedNoneSet(values=(['circle', 'dash', 'diamond', 'dot',
'picture',
24: (30)                                              'plus', 'square', 'star', 'triangle', 'x',
'auto']),
25: (27)                                          to_tree=_explicit_none)
26: (4)                 size = NestedMinMax(min=2, max=72, allow_none=True)
27: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
28: (4)                 graphicalProperties = Alias('spPr')
29: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
```

```
(4)             __elements__ = ('symbol', 'size', 'spPr')
(4)             def __init__(self,
(17)                         symbol=None,
(17)                         size=None,
(17)                         spPr=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.symbol = symbol
(8)                 self.size = size
(8)                 if spPr is None:
(12)                    spPr = GraphicalProperties()
(8)                 self.spPr = spPr
(0)         class DataPoint(Serialisable):
(4)             tagname = "dPt"
(4)             idx = NestedInteger()
(4)             invertIfNegative = NestedBool(allow_none=True)
(4)             marker = Typed(expected_type=Marker, allow_none=True)
(4)             bubble3D = NestedBool(allow_none=True)
(4)             explosion = NestedInteger(allow_none=True)
(4)             spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)             graphicalProperties = Alias('spPr')
(4)             pictureOptions = Typed(expected_type=PictureOptions, allow_none=True)
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)             __elements__ = ('idx', 'invertIfNegative', 'marker', 'bubble3D',
(20)                            'explosion', 'spPr', 'pictureOptions')
(4)             def __init__(self,
(17)                         idx=None,
(17)                         invertIfNegative=None,
(17)                         marker=None,
(17)                         bubble3D=None,
(17)                         explosion=None,
(17)                         spPr=None,
(17)                         pictureOptions=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.idx = idx
(8)                 self.invertIfNegative = invertIfNegative
(8)                 self.marker = marker
(8)                 self.bubble3D = bubble3D
(8)                 self.explosion = explosion
(8)                 if spPr is None:
(12)                    spPr = GraphicalProperties()
(8)                 self.spPr = spPr
(8)                 self.pictureOptions = pictureOptions
```

File 26 - reader.py:

```
(0)         """
(0)         Read a chart
(0)         """
(0)         def read_chart(chartspace):
(4)             cs = chartspace
(4)             plot = cs.chart.plotArea
(4)             chart = plot._charts[0]
(4)             chart._charts = plot._charts
(4)             chart.title = cs.chart.title
(4)             chart.display_blanks = cs.chart.dispBlanksAs
(4)             chart.visible_cells_only = cs.chart.plotVisOnly
(4)             chart.layout = plot.layout
(4)             chart.legend = cs.chart.legend
(4)             chart.floor = cs.chart.floor
(4)             chart.sideWall = cs.chart.sideWall
(4)             chart.backWall = cs.chart.backWall
(4)             chart.pivotSource = cs.pivotSource
(4)             chart.pivotFormats = cs.chart.pivotFmts
(4)             chart.idx_base = min((s.idx for s in chart.series), default=0)
(4)             chart._reindex()
```

```
21: (4)                 chart.graphical_properties = cs.graphical_properties
22: (4)                 return chart
```

File 27 - series.py:

```
1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  String,
5: (4)                  Integer,
6: (4)                  Bool,
7: (4)                  Alias,
8: (4)                  Sequence,
9: (0)              )
10: (0)             from openpyxl.descriptors.excel import ExtensionList
11: (0)             from openpyxl.descriptors.nested import (
12: (4)                 NestedInteger,
13: (4)                 NestedBool,
14: (4)                 NestedNoneSet,
15: (4)                 NestedText,
16: (0)             )
17: (0)             from .shapes import GraphicalProperties
18: (0)             from .data_source import (
19: (4)                 AxDataSource,
20: (4)                 NumDataSource,
21: (4)                 NumRef,
22: (4)                 StrRef,
23: (0)             )
24: (0)             from .error_bar import ErrorBars
25: (0)             from .label import DataLabelList
26: (0)             from .marker import DataPoint, PictureOptions, Marker
27: (0)             from .trendline import Trendline
28: (0)             attribute_mapping = {
29: (4)                 'area': ('idx', 'order', 'tx', 'spPr', 'pictureOptions', 'dPt', 'dLbls',
'errBars',
30: (13)                         'trendline', 'cat', 'val',),
31: (4)                 'bar':('idx', 'order','tx', 'spPr', 'invertIfNegative', 'pictureOptions',
'dPt',
32: (11)                      'dLbls', 'trendline', 'errBars', 'cat', 'val', 'shape'),
33: (4)                 'bubble':('idx','order', 'tx', 'spPr', 'invertIfNegative', 'dPt', 'dLbls',
34: (14)                          'trendline', 'errBars', 'xVal', 'yVal', 'bubbleSize',
'bubble3D'),
35: (4)                 'line':('idx', 'order', 'tx', 'spPr', 'marker', 'dPt', 'dLbls',
'trendline',
36: (12)                       'errBars', 'cat', 'val', 'smooth'),
37: (4)                 'pie':('idx', 'order', 'tx', 'spPr', 'explosion', 'dPt', 'dLbls', 'cat',
'val'),
38: (4)                 'radar':('idx', 'order', 'tx', 'spPr', 'marker', 'dPt', 'dLbls', 'cat',
'val'),
39: (4)                 'scatter':('idx', 'order', 'tx', 'spPr', 'marker', 'dPt', 'dLbls',
'trendline',
40: (15)                           'errBars', 'xVal', 'yVal', 'smooth'),
41: (4)                 'surface':('idx', 'order', 'tx', 'spPr', 'cat', 'val'),
42: (21)                                 }
43: (0)             class SeriesLabel(Serialisable):
44: (4)                 tagname = "tx"
45: (4)                 strRef = Typed(expected_type=StrRef, allow_none=True)
46: (4)                 v = NestedText(expected_type=str, allow_none=True)
47: (4)                 value = Alias('v')
48: (4)                 __elements__ = ('strRef', 'v')
49: (4)                 def __init__(self,
50: (17)                             strRef=None,
51: (17)                             v=None):
52: (8)                     self.strRef = strRef
53: (8)                     self.v = v
54: (0)             class Series(Serialisable):
55: (4)                 """
```

```
(4)         Generic series object. Should not be instantiated directly.
(4)         User the chart.Series factory instead.
(4)         """
(4)         tagname = "ser"
(4)         idx = NestedInteger()
(4)         order = NestedInteger()
(4)         tx = Typed(expected_type=SeriesLabel, allow_none=True)
(4)         title = Alias('tx')
(4)         spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)         graphicalProperties = Alias('spPr')
(4)         pictureOptions = Typed(expected_type=PictureOptions, allow_none=True)
(4)         dPt = Sequence(expected_type=DataPoint, allow_none=True)
(4)         data_points = Alias("dPt")
(4)         dLbls = Typed(expected_type=DataLabelList, allow_none=True)
(4)         labels = Alias("dLbls")
(4)         trendline = Typed(expected_type=Trendline, allow_none=True)
(4)         errBars = Typed(expected_type=ErrorBars, allow_none=True)
(4)         cat = Typed(expected_type=AxDataSource, allow_none=True)
(4)         identifiers = Alias("cat")
(4)         val = Typed(expected_type=NumDataSource, allow_none=True)
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         invertIfNegative = NestedBool(allow_none=True)
(4)         shape = NestedNoneSet(values=(['cone', 'coneToMax', 'box', 'cylinder', 
'pyramid', 'pyramidToMax']))
(4)         xVal = Typed(expected_type=AxDataSource, allow_none=True)
(4)         yVal = Typed(expected_type=NumDataSource, allow_none=True)
(4)         bubbleSize = Typed(expected_type=NumDataSource, allow_none=True)
(4)         zVal = Alias("bubbleSize")
(4)         bubble3D = NestedBool(allow_none=True)
(4)         marker = Typed(expected_type=Marker, allow_none=True)
(4)         smooth = NestedBool(allow_none=True)
(4)         explosion = NestedInteger(allow_none=True)
(4)         __elements__ = ()
(4)         def __init__(self,
(17)                     idx=0,
(17)                     order=0,
(17)                     tx=None,
(17)                     spPr=None,
(17)                     pictureOptions=None,
(17)                     dPt=(),
(17)                     dLbls=None,
(17)                     trendline=None,
(17)                     errBars=None,
(17)                     cat=None,
(17)                     val=None,
(17)                     invertIfNegative=None,
(17)                     shape=None,
(17)                     xVal=None,
(17)                     yVal=None,
(17)                     bubbleSize=None,
(17)                     bubble3D=None,
(17)                     marker=None,
(17)                     smooth=None,
(17)                     explosion=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.idx = idx
(8)             self.order = order
(8)             self.tx = tx
(8)             if spPr is None:
(12)                spPr = GraphicalProperties()
(8)             self.spPr = spPr
(8)             self.pictureOptions = pictureOptions
(8)             self.dPt = dPt
(8)             self.dLbls = dLbls
(8)             self.trendline = trendline
(8)             self.errBars = errBars
(8)             self.cat = cat
(8)             self.val = val
```

```
124: (8)                        self.invertIfNegative = invertIfNegative
125: (8)                        self.shape = shape
126: (8)                        self.xVal = xVal
127: (8)                        self.yVal = yVal
128: (8)                        self.bubbleSize = bubbleSize
129: (8)                        self.bubble3D = bubble3D
130: (8)                        if marker is None:
131: (12)                           marker = Marker()
132: (8)                        self.marker = marker
133: (8)                        self.smooth = smooth
134: (8)                        self.explosion = explosion
135: (4)                    def to_tree(self, tagname=None, idx=None):
136: (8)                        """The index can need rebasing"""
137: (8)                        if idx is not None:
138: (12)                           if self.order == self.idx:
139: (16)                               self.order = idx # rebase the order if the index has been
rebased
140: (12)                           self.idx = idx
141: (8)                        return super().to_tree(tagname)
142: (0)                class XYSeries(Series):
143: (4)                    """Dedicated series for charts that have x and y series"""
144: (4)                    idx = Series.idx
145: (4)                    order = Series.order
146: (4)                    tx = Series.tx
147: (4)                    spPr = Series.spPr
148: (4)                    dPt = Series.dPt
149: (4)                    dLbls = Series.dLbls
150: (4)                    trendline = Series.trendline
151: (4)                    errBars = Series.errBars
152: (4)                    xVal = Series.xVal
153: (4)                    yVal = Series.yVal
154: (4)                    invertIfNegative = Series.invertIfNegative
155: (4)                    bubbleSize = Series.bubbleSize
156: (4)                    bubble3D = Series.bubble3D
157: (4)                    marker = Series.marker
158: (4)                    smooth = Series.smooth

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

File 28 - shapes.py:

1: (0)                  from openpyxl.descriptors.serialisable import Serialisable
2: (0)                  from openpyxl.descriptors import (
3: (4)                      Typed,
4: (4)                      Alias
5: (0)                  )
6: (0)                  from openpyxl.descriptors.nested import (
7: (4)                      EmptyTag
8: (0)                  )
9: (0)                  from openpyxl.drawing.colors import ColorChoiceDescriptor
10: (0)                 from openpyxl.drawing.fill import *
11: (0)                 from openpyxl.drawing.line import LineProperties
12: (0)                 from openpyxl.drawing.geometry import (
13: (4)                     Shape3D,
14: (4)                     Scene3D,
15: (4)                     Transform2D,
16: (4)                     CustomGeometry2D,
17: (4)                     PresetGeometry2D,
18: (0)                 )
19: (0)                 class GraphicalProperties(Serialisable):
20: (4)                     """
21: (4)                     Somewhat vaguely 21.2.2.197 says this:
22: (4)                     This element specifies the formatting for the parent chart element. The
23: (4)                     custGeom, prstGeom, scene3d, and xfrm elements are not supported. The
24: (4)                     bwMode attribute is not supported.
25: (4)                     This doesn't leave much. And the element is used in different places.
26: (4)                     """
27: (4)                     tagname = "spPr"
28: (4)                     bwMode = NoneSet(values=(['clr', 'auto', 'gray', 'ltGray', 'invGray',
29: (26)                                    'grayWhite', 'blackGray', 'blackWhite', 'black',
'white', 'hidden']
30: (25)                                   )
31: (17)                           )
32: (4)                 xfrm = Typed(expected_type=Transform2D, allow_none=True)
33: (4)                 transform = Alias('xfrm')
34: (4)                 custGeom = Typed(expected_type=CustomGeometry2D, allow_none=True) # either
or
35: (4)                 prstGeom = Typed(expected_type=PresetGeometry2D, allow_none=True)
36: (4)                 noFill = EmptyTag(namespace=DRAWING_NS)
37: (4)                 solidFill = ColorChoiceDescriptor()
38: (4)                 gradFill = Typed(expected_type=GradientFillProperties, allow_none=True)
39: (4)                 pattFill = Typed(expected_type=PatternFillProperties, allow_none=True)
40: (4)                 ln = Typed(expected_type=LineProperties, allow_none=True)
41: (4)                 line = Alias('ln')
42: (4)                 scene3d = Typed(expected_type=Scene3D, allow_none=True)
43: (4)                 sp3d = Typed(expected_type=Shape3D, allow_none=True)
44: (4)                 shape3D = Alias('sp3d')
45: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
46: (4)                 __elements__ = ('xfrm', 'prstGeom', 'noFill', 'solidFill', 'gradFill',
'pattFill',
47: (20)                             'ln', 'scene3d', 'sp3d')
48: (4)                 def __init__(self,
49: (17)                             bwMode=None,
50: (17)                             xfrm=None,
51: (17)                             noFill=None,
52: (17)                             solidFill=None,
53: (17)                             gradFill=None,
54: (17)                             pattFill=None,
55: (17)                             ln=None,
56: (17)                             scene3d=None,
57: (17)                             custGeom=None,
58: (17)                             prstGeom=None,
59: (17)                             sp3d=None,
60: (17)                             extLst=None,
61: (16)                            ):
62: (8)                     self.bwMode = bwMode
63: (8)                     self.xfrm = xfrm
64: (8)                     self.noFill = noFill
65: (8)                     self.solidFill = solidFill
66: (8)                     self.gradFill = gradFill
67: (8)                     self.pattFill = pattFill
68: (8)                     if ln is None:
69: (12)                        ln = LineProperties()
70: (8)                     self.ln = ln
71: (8)                     self.custGeom = custGeom
72: (8)                     self.prstGeom = prstGeom
73: (8)                     self.scene3d = scene3d
74: (8)                     self.sp3d = sp3d

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

File 29 - picture.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors.nested import (
3: (4)                  NestedBool,
4: (4)                  NestedFloat,
5: (4)                  NestedMinMax,
6: (4)                  NestedNoneSet,
7: (0)              )
8: (0)              class PictureOptions(Serialisable):
9: (4)                  tagname = "pictureOptions"
10: (4)                 applyToFront = NestedBool(allow_none=True, nested=True)
11: (4)                 applyToSides = NestedBool(allow_none=True, nested=True)
12: (4)                 applyToEnd = NestedBool(allow_none=True, nested=True)
13: (4)                 pictureFormat = NestedNoneSet(values=(['stretch', 'stack', 'stackScale']),
nested=True)
14: (4)                 pictureStackUnit = NestedFloat(allow_none=True, nested=True)
```

```
15: (4)                 __elements__ = ('applyToFront', 'applyToSides', 'applyToEnd',
'pictureFormat', 'pictureStackUnit')
16: (4)                 def __init__(self,
17: (17)                             applyToFront=None,
18: (17)                             applyToSides=None,
19: (17)                             applyToEnd=None,
20: (17)                             pictureFormat=None,
21: (17)                             pictureStackUnit=None,
22: (16)                            ):
23: (8)                     self.applyToFront = applyToFront
24: (8)                     self.applyToSides = applyToSides
25: (8)                     self.applyToEnd = applyToEnd
26: (8)                     self.pictureFormat = pictureFormat
27: (8)                     self.pictureStackUnit = pictureStackUnit

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

File 30 - plotarea.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Alias,
5: (0)              )
6: (0)              from openpyxl.descriptors.excel import (
7: (4)                  ExtensionList,
8: (0)              )
9: (0)              from openpyxl.descriptors.sequence import (
10: (4)                 MultiSequence,
11: (4)                 MultiSequencePart,
12: (0)             )
13: (0)             from openpyxl.descriptors.nested import (
14: (4)                 NestedBool,
15: (0)             )
16: (0)             from ._3d import _3DBase
17: (0)             from .area_chart import AreaChart, AreaChart3D
18: (0)             from .bar_chart import BarChart, BarChart3D
19: (0)             from .bubble_chart import BubbleChart
20: (0)             from .line_chart import LineChart, LineChart3D
21: (0)             from .pie_chart import PieChart, PieChart3D, ProjectedPieChart, DoughnutChart
22: (0)             from .radar_chart import RadarChart
23: (0)             from .scatter_chart import ScatterChart
24: (0)             from .stock_chart import StockChart
25: (0)             from .surface_chart import SurfaceChart, SurfaceChart3D
26: (0)             from .layout import Layout
27: (0)             from .shapes import GraphicalProperties
28: (0)             from .text import RichText
29: (0)             from .axis import (
30: (4)                 NumericAxis,
31: (4)                 TextAxis,
32: (4)                 SeriesAxis,
33: (4)                 DateAxis,
34: (0)             )
35: (0)             class DataTable(Serialisable):
36: (4)                 tagname = "dTable"
37: (4)                 showHorzBorder = NestedBool(allow_none=True)
38: (4)                 showVertBorder = NestedBool(allow_none=True)
39: (4)                 showOutline = NestedBool(allow_none=True)
40: (4)                 showKeys = NestedBool(allow_none=True)
41: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
42: (4)                 graphicalProperties = Alias('spPr')
43: (4)                 txPr = Typed(expected_type=RichText, allow_none=True)
44: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
45: (4)                 __elements__ = ('showHorzBorder', 'showVertBorder', 'showOutline',
46: (20)                                'showKeys', 'spPr', 'txPr')
47: (4)                 def __init__(self,
48: (17)                             showHorzBorder=None,
49: (17)                             showVertBorder=None,
50: (17)                             showOutline=None,
(17)                         showKeys=None,
(17)                         spPr=None,
(17)                         txPr=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.showHorzBorder = showHorzBorder
(8)                 self.showVertBorder = showVertBorder
(8)                 self.showOutline = showOutline
(8)                 self.showKeys = showKeys
(8)                 self.spPr = spPr
(8)                 self.txPr = txPr
(0)         class PlotArea(Serialisable):
(4)             tagname = "plotArea"
(4)             layout = Typed(expected_type=Layout, allow_none=True)
(4)             dTable = Typed(expected_type=DataTable, allow_none=True)
(4)             spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)             graphicalProperties = Alias("spPr")
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)             _charts = MultiSequence()
(4)             areaChart = MultiSequencePart(expected_type=AreaChart, store="_charts")
(4)             area3DChart = MultiSequencePart(expected_type=AreaChart3D, store="_charts")
(4)             lineChart = MultiSequencePart(expected_type=LineChart, store="_charts")
(4)             line3DChart = MultiSequencePart(expected_type=LineChart3D, store="_charts")
(4)             stockChart = MultiSequencePart(expected_type=StockChart, store="_charts")
(4)             radarChart = MultiSequencePart(expected_type=RadarChart, store="_charts")
(4)             scatterChart = MultiSequencePart(expected_type=ScatterChart, store="_charts")
(4)             pieChart = MultiSequencePart(expected_type=PieChart, store="_charts")
(4)             pie3DChart = MultiSequencePart(expected_type=PieChart3D, store="_charts")
(4)             doughnutChart = MultiSequencePart(expected_type=DoughnutChart, store="_charts")
(4)             barChart = MultiSequencePart(expected_type=BarChart, store="_charts")
(4)             bar3DChart = MultiSequencePart(expected_type=BarChart3D, store="_charts")
(4)             ofPieChart = MultiSequencePart(expected_type=ProjectedPieChart, store="_charts")
(4)             surfaceChart = MultiSequencePart(expected_type=SurfaceChart, store="_charts")
(4)             surface3DChart = MultiSequencePart(expected_type=SurfaceChart3D, store="_charts")
(4)             bubbleChart = MultiSequencePart(expected_type=BubbleChart, store="_charts")
(4)             _axes = MultiSequence()
(4)             valAx = MultiSequencePart(expected_type=NumericAxis, store="_axes")
(4)             catAx = MultiSequencePart(expected_type=TextAxis, store="_axes")
(4)             dateAx = MultiSequencePart(expected_type=DateAxis, store="_axes")
(4)             serAx = MultiSequencePart(expected_type=SeriesAxis, store="_axes")
(4)             __elements__ = ('layout', '_charts', '_axes', 'dTable', 'spPr')
(4)             def __init__(self,
(17)                         layout=None,
(17)                         dTable=None,
(17)                         spPr=None,
(17)                         _charts=(),
(17)                         _axes=(),
(17)                         extLst=None,
(16)                        ):
(8)                 self.layout = layout
(8)                 self.dTable = dTable
(8)                 self.spPr = spPr
(8)                 self._charts = _charts
(8)                 self._axes = _axes
(4)             def to_tree(self, tagname=None, idx=None, namespace=None):
(8)                 axIds = {ax.axId for ax in self._axes}
(8)                 for chart in self._charts:
(12)                    for id, axis in chart._axes.items():
(16)                        if id not in axIds:
(20)                            setattr(self, axis.tagname, axis)
(20)                            axIds.add(id)
```

```
(8)             return super().to_tree(tagname)
(4)         @classmethod
(4)         def from_tree(cls, node):
(8)             self = super().from_tree(node)
(8)             axes = dict((axis.axId, axis) for axis in self._axes)
(8)             for chart in self._charts:
(12)                if isinstance(chart, (ScatterChart, BubbleChart)):
(16)                    x, y = (axes[axId] for axId in chart.axId)
(16)                    chart.x_axis = x
(16)                    chart.y_axis = y
(16)                    continue
(12)                for axId in chart.axId:
(16)                    axis = axes.get(axId)
(16)                    if axis is None and isinstance(chart, _3DBase):
(20)                        chart.z_axis = None
(20)                        continue
(16)                    if axis.tagname in ("catAx", "dateAx"):
(20)                        chart.x_axis = axis
(16)                    elif axis.tagname == "valAx":
(20)                        chart.y_axis = axis
(16)                    elif axis.tagname == "serAx":
(20)                        chart.z_axis = axis
(8)             return self

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

File 31 - error_bar.py:

(0)         from openpyxl.descriptors.serialisable import Serialisable
(0)         from openpyxl.descriptors import (
(4)             Typed,
(4)             Float,
(4)             Set,
(4)             Alias
(0)         )
(0)         from openpyxl.descriptors.excel import ExtensionList
(0)         from openpyxl.descriptors.nested import (
(4)             NestedNoneSet,
(4)             NestedSet,
(4)             NestedBool,
(4)             NestedFloat,
(0)         )
(0)         from .data_source import NumDataSource
(0)         from .shapes import GraphicalProperties
(0)         class ErrorBars(Serialisable):
(4)             tagname = "errBars"
(4)             errDir = NestedNoneSet(values=(['x', 'y']))
(4)             direction = Alias("errDir")
(4)             errBarType = NestedSet(values=(['both', 'minus', 'plus']))
(4)             style = Alias("errBarType")
(4)             errValType = NestedSet(values=(['cust', 'fixedVal', 'percentage', 'stdDev', 'stdErr']))
(4)             size = Alias("errValType")
(4)             noEndCap = NestedBool(nested=True, allow_none=True)
(4)             plus = Typed(expected_type=NumDataSource, allow_none=True)
(4)             minus = Typed(expected_type=NumDataSource, allow_none=True)
(4)             val = NestedFloat(allow_none=True)
(4)             spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)             graphicalProperties = Alias("spPr")
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)             __elements__ = ('errDir','errBarType', 'errValType', 'noEndCap','minus', 'plus', 'val', 'spPr')
(4)             def __init__(self,
(17)                         errDir=None,
(17)                         errBarType="both",
(17)                         errValType="fixedVal",
(17)                         noEndCap=None,
(17)                         plus=None,
(17)                         minus=None,
40: (17)                             val=None,
41: (17)                             spPr=None,
42: (17)                             extLst=None,
43: (16)                            ):
44: (8)                     self.errDir = errDir
45: (8)                     self.errBarType = errBarType
46: (8)                     self.errValType = errValType
47: (8)                     self.noEndCap = noEndCap
48: (8)                     self.plus = plus
49: (8)                     self.minus = minus
50: (8)                     self.val = val
51: (8)                     self.spPr = spPr
```

File 32 - pie_chart.py:

```
1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Bool,
5: (4)                  MinMax,
6: (4)                  Integer,
7: (4)                  NoneSet,
8: (4)                  Float,
9: (4)                  Alias,
10: (4)                 Sequence,
11: (0)             )
12: (0)             from openpyxl.descriptors.excel import ExtensionList, Percentage
13: (0)             from openpyxl.descriptors.nested import (
14: (4)                 NestedBool,
15: (4)                 NestedMinMax,
16: (4)                 NestedInteger,
17: (4)                 NestedFloat,
18: (4)                 NestedNoneSet,
19: (4)                 NestedSet,
20: (0)             )
21: (0)             from openpyxl.descriptors.sequence import ValueSequence
22: (0)             from ._chart import ChartBase
23: (0)             from .axis import ChartLines
24: (0)             from .descriptors import NestedGapAmount
25: (0)             from .series import Series
26: (0)             from .label import DataLabelList
27: (0)             class _PieChartBase(ChartBase):
28: (4)                 varyColors = NestedBool(allow_none=True)
29: (4)                 ser = Sequence(expected_type=Series, allow_none=True)
30: (4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
31: (4)                 dataLabels = Alias("dLbls")
32: (4)                 _series_type = "pie"
33: (4)                 __elements__ = ('varyColors', 'ser', 'dLbls')
34: (4)                 def __init__(self,
35: (17)                             varyColors=True,
36: (17)                             ser=(),
37: (17)                             dLbls=None,
38: (16)                            ):
39: (8)                     self.varyColors = varyColors
40: (8)                     self.ser = ser
41: (8)                     self.dLbls = dLbls
42: (8)                     super().__init__()
43: (0)             class PieChart(_PieChartBase):
44: (4)                 tagname = "pieChart"
45: (4)                 varyColors = _PieChartBase.varyColors
46: (4)                 ser = _PieChartBase.ser
47: (4)                 dLbls = _PieChartBase.dLbls
48: (4)                 firstSliceAng = NestedMinMax(min=0, max=360)
49: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
50: (4)                 __elements__ = _PieChartBase.__elements__ + ('firstSliceAng', )
51: (4)                 def __init__(self,
52: (17)                             firstSliceAng=0,
(17)                     extLst=None,
(17)                     **kw
(16)                    ):
(8)         self.firstSliceAng = firstSliceAng
(8)         super().__init__(**kw)
(0) class PieChart3D(_PieChartBase):
(4)     tagname = "pie3DChart"
(4)     varyColors = _PieChartBase.varyColors
(4)     ser = _PieChartBase.ser
(4)     dLbls = _PieChartBase.dLbls
(4)     extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)     __elements__ = _PieChartBase.__elements__
(0) class DoughnutChart(_PieChartBase):
(4)     tagname = "doughnutChart"
(4)     varyColors = _PieChartBase.varyColors
(4)     ser = _PieChartBase.ser
(4)     dLbls = _PieChartBase.dLbls
(4)     firstSliceAng = NestedMinMax(min=0, max=360)
(4)     holeSize = NestedMinMax(min=1, max=90, allow_none=True)
(4)     extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)     __elements__ = _PieChartBase.__elements__ + ('firstSliceAng', 'holeSize')
(4)     def __init__(self,
(17)                     firstSliceAng=0,
(17)                     holeSize=10,
(17)                     extLst=None,
(17)                     **kw
(16)                    ):
(8)         self.firstSliceAng = firstSliceAng
(8)         self.holeSize = holeSize
(8)         super().__init__(**kw)
(0) class CustomSplit(Serialisable):
(4)     tagname = "custSplit"
(4)     secondPiePt = ValueSequence(expected_type=int)
(4)     __elements__ = ('secondPiePt',)
(4)     def __init__(self,
(17)                     secondPiePt=(),
(16)                    ):
(8)         self.secondPiePt = secondPiePt
(0) class ProjectedPieChart(_PieChartBase):
(4)     """
(4)     From the spec 21.2.2.126
(4)     This element contains the pie of pie or bar of pie series on this
(4)     chart. Only the first series shall be displayed. The splitType element
(4)     shall determine whether the splitPos and custSplit elements apply.
(4)     """
(4)     tagname = "ofPieChart"
(4)     varyColors = _PieChartBase.varyColors
(4)     ser = _PieChartBase.ser
(4)     dLbls = _PieChartBase.dLbls
(4)     ofPieType = NestedSet(values=(['pie', 'bar']))
(4)     type = Alias('ofPieType')
(4)     gapWidth = NestedGapAmount()
(4)     splitType = NestedNoneSet(values=(['auto', 'cust', 'percent', 'pos', 'val']))
(4)     splitPos = NestedFloat(allow_none=True)
(4)     custSplit = Typed(expected_type=CustomSplit, allow_none=True)
(4)     secondPieSize = NestedMinMax(min=5, max=200, allow_none=True)
(4)     serLines = Typed(expected_type=ChartLines, allow_none=True)
(4)     join_lines = Alias('serLines')
(4)     extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)     __elements__ = _PieChartBase.__elements__ + ('ofPieType', 'gapWidth',
(49)                                                     'splitType', 'splitPos', 'custSplit', 'secondPieSize', 'serLines')
(4)     def __init__(self,
(17)                     ofPieType="pie",
(17)                     gapWidth=None,
(17)                     splitType="auto",
(17)                     splitPos=None,
(17)                     custSplit=None,
(17)                         secondPieSize=75,
(17)                         serLines=None,
(17)                         extLst=None,
(17)                         **kw
(16)                        ):
(8)                 self.ofPieType = ofPieType
(8)                 self.gapWidth = gapWidth
(8)                 self.splitType = splitType
(8)                 self.splitPos = splitPos
(8)                 self.custSplit = custSplit
(8)                 self.secondPieSize = secondPieSize
(8)                 if serLines is None:
(12)                    self.serLines = ChartLines()
(8)                 super().__init__(**kw)

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

File 33 - reference.py:

(0)         from itertools import chain
(0)         from openpyxl.descriptors.serialisable import Serialisable
(0)         from openpyxl.descriptors import (
(4)             MinMax,
(4)             Typed,
(4)             String,
(4)             Strict,
(0)         )
(0)         from openpyxl.worksheet.worksheet import Worksheet
(0)         from openpyxl.utils import (
(4)             get_column_letter,
(4)             range_to_tuple,
(4)             quote_sheetname
(0)         )
(0)         class DummyWorksheet:
(4)             def __init__(self, title):
(8)                 self.title = title
(0)         class Reference(Strict):
(4)             """
(4)             Normalise cell range references
(4)             """
(4)             min_row = MinMax(min=1, max=1000000, expected_type=int)
(4)             max_row = MinMax(min=1, max=1000000, expected_type=int)
(4)             min_col = MinMax(min=1, max=16384, expected_type=int)
(4)             max_col = MinMax(min=1, max=16384, expected_type=int)
(4)             range_string = String(allow_none=True)
(4)             def __init__(self,
(17)                         worksheet=None,
(17)                         min_col=None,
(17)                         min_row=None,
(17)                         max_col=None,
(17)                         max_row=None,
(17)                         range_string=None
(17)                         ):
(8)                 if range_string is not None:
(12)                    sheetname, boundaries = range_to_tuple(range_string)
(12)                    min_col, min_row, max_col, max_row = boundaries
(12)                    worksheet = DummyWorksheet(sheetname)
(8)                 self.worksheet = worksheet
(8)                 self.min_col = min_col
(8)                 self.min_row = min_row
(8)                 if max_col is None:
(12)                    max_col = min_col
(8)                 self.max_col = max_col
(8)                 if max_row is None:
(12)                    max_row = min_row
(8)                 self.max_row = max_row
(4)             def __repr__(self):
(8)                 return str(self)
(4)             def __str__(self):
```

```
51: (8)                     fmt = u"{0}!${1}${2}:${3}${4}"
52: (8)                     if (self.min_col == self.max_col
53: (12)                        and self.min_row == self.max_row):
54: (12)                        fmt = u"{0}!${1}${2}"
55: (8)                     return fmt.format(self.sheetname,
56: (26)                                      get_column_letter(self.min_col), self.min_row,
57: (26)                                      get_column_letter(self.max_col), self.max_row
58: (26)                                      )
59: (4)                 __str__ = __str__
60: (4)                 def __len__(self):
61: (8)                     if self.min_row == self.max_row:
62: (12)                        return 1 + self.max_col - self.min_col
63: (8)                     return 1 + self.max_row - self.min_row
64: (4)                 def __eq__(self, other):
65: (8)                     return str(self) == str(other)
66: (4)                 @property
67: (4)                 def rows(self):
68: (8)                     """
69: (8)                     Return all rows in the range
70: (8)                     """
71: (8)                     for row in range(self.min_row, self.max_row+1):
72: (12)                        yield Reference(self.worksheet, self.min_col, row, self.max_col,
row)
73: (4)                 @property
74: (4)                 def cols(self):
75: (8)                     """
76: (8)                     Return all columns in the range
77: (8)                     """
78: (8)                     for col in range(self.min_col, self.max_col+1):
79: (12)                        yield Reference(self.worksheet, col, self.min_row, col,
self.max_row)
80: (4)                 def pop(self):
81: (8)                     """
82: (8)                     Return and remove the first cell
83: (8)                     """
84: (8)                     cell = "{0}{1}".format(get_column_letter(self.min_col), self.min_row)
85: (8)                     if self.min_row == self.max_row:
86: (12)                        self.min_col += 1
87: (8)                     else:
88: (12)                        self.min_row += 1
89: (8)                     return cell
90: (4)                 @property
91: (4)                 def sheetname(self):
92: (8)                     return quote_sheetname(self.worksheet.title)

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

File 34 - trendline.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  String,
5: (4)                  Alias
6: (0)              )
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from openpyxl.descriptors.nested import (
9: (4)                  NestedBool,
10: (4)                 NestedInteger,
11: (4)                 NestedFloat,
12: (4)                 NestedSet
13: (0)             )
14: (0)             from .data_source import NumFmt
15: (0)             from .shapes import GraphicalProperties
16: (0)             from .text import RichText, Text
17: (0)             from .layout import Layout
18: (0)             class TrendlineLabel(Serialisable):
19: (4)                 tagname = "trendlineLbl"
20: (4)                 layout = Typed(expected_type=Layout, allow_none=True)
```

```
(4)                 tx = Typed(expected_type=Text, allow_none=True)
(4)                 numFmt = Typed(expected_type=NumFmt, allow_none=True)
(4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)                 graphicalProperties = Alias("spPr")
(4)                 txPr = Typed(expected_type=RichText, allow_none=True)
(4)                 textProperties = Alias("txPr")
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ('layout', 'tx', 'numFmt', 'spPr', 'txPr')
(4)                 def __init__(self,
(17)                             layout=None,
(17)                             tx=None,
(17)                             numFmt=None,
(17)                             spPr=None,
(17)                             txPr=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.layout = layout
(8)                     self.tx = tx
(8)                     self.numFmt = numFmt
(8)                     self.spPr = spPr
(8)                     self.txPr = txPr
(0)             class Trendline(Serialisable):
(4)                 tagname = "trendline"
(4)                 name = String(allow_none=True)
(4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
(4)                 graphicalProperties = Alias('spPr')
(4)                 trendlineType = NestedSet(values=(['exp', 'linear', 'log', 'movingAvg', 'poly', 'power']))
(4)                 order = NestedInteger(allow_none=True)
(4)                 period = NestedInteger(allow_none=True)
(4)                 forward = NestedFloat(allow_none=True)
(4)                 backward = NestedFloat(allow_none=True)
(4)                 intercept = NestedFloat(allow_none=True)
(4)                 dispRSqr = NestedBool(allow_none=True)
(4)                 dispEq = NestedBool(allow_none=True)
(4)                 trendlineLbl = Typed(expected_type=TrendlineLabel, allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ('spPr', 'trendlineType', 'order', 'period', 'forward',
(20)                                'backward', 'intercept', 'dispRSqr', 'dispEq', 'trendlineLbl')
(4)                 def __init__(self,
(17)                             name=None,
(17)                             spPr=None,
(17)                             trendlineType='linear',
(17)                             order=None,
(17)                             period=None,
(17)                             forward=None,
(17)                             backward=None,
(17)                             intercept=None,
(17)                             dispRSqr=None,
(17)                             dispEq=None,
(17)                             trendlineLbl=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.name = name
(8)                     self.spPr = spPr
(8)                     self.trendlineType = trendlineType
(8)                     self.order = order
(8)                     self.period = period
(8)                     self.forward = forward
(8)                     self.backward = backward
(8)                     self.intercept = intercept
(8)                     self.dispRSqr = dispRSqr
(8)                     self.dispEq = dispEq
(8)                     self.trendlineLbl = trendlineLbl
```

File 35 - line_chart.py:

```
(0)             from openpyxl.descriptors import (
(4)                 Typed,
(4)                 Sequence,
(4)                 Alias,
(4)                 )
(0)             from openpyxl.descriptors.excel import ExtensionList
(0)             from openpyxl.descriptors.nested import (
(4)                 NestedSet,
(4)                 NestedBool,
(0)             )
(0)             from ._chart import ChartBase
(0)             from .updown_bars import UpDownBars
(0)             from .descriptors import NestedGapAmount
(0)             from .axis import TextAxis, NumericAxis, SeriesAxis, ChartLines, _BaseAxis
(0)             from .label import DataLabelList
(0)             from .series import Series
(0)             class _LineChartBase(ChartBase):
(4)                 grouping = NestedSet(values=(['percentStacked', 'standard', 'stacked']))
(4)                 varyColors = NestedBool(allow_none=True)
(4)                 ser = Sequence(expected_type=Series, allow_none=True)
(4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
(4)                 dataLabels = Alias("dLbls")
(4)                 dropLines = Typed(expected_type=ChartLines, allow_none=True)
(4)                 _series_type = "line"
(4)                 __elements__ = ('grouping', 'varyColors', 'ser', 'dLbls', 'dropLines')
(4)                 def __init__(self,
(17)                             grouping="standard",
(17)                             varyColors=None,
(17)                             ser=(),
(17)                             dLbls=None,
(17)                             dropLines=None,
(17)                             **kw
(16)                            ):
(8)                     self.grouping = grouping
(8)                     self.varyColors = varyColors
(8)                     self.ser = ser
(8)                     self.dLbls = dLbls
(8)                     self.dropLines = dropLines
(8)                     super().__init__(**kw)
(0)             class LineChart(_LineChartBase):
(4)                 tagname = "lineChart"
(4)                 grouping = _LineChartBase.grouping
(4)                 varyColors = _LineChartBase.varyColors
(4)                 ser = _LineChartBase.ser
(4)                 dLbls = _LineChartBase.dLbls
(4)                 dropLines =_LineChartBase.dropLines
(4)                 hiLowLines = Typed(expected_type=ChartLines, allow_none=True)
(4)                 upDownBars = Typed(expected_type=UpDownBars, allow_none=True)
(4)                 marker = NestedBool(allow_none=True)
(4)                 smooth = NestedBool(allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 x_axis = Typed(expected_type=_BaseAxis)
(4)                 y_axis = Typed(expected_type=NumericAxis)
(4)                 __elements__ = _LineChartBase.__elements__ + ('hiLowLines', 'upDownBars', 'marker', 'smooth', 'axId')
(4)                 def __init__(self,
(17)                             hiLowLines=None,
(17)                             upDownBars=None,
(17)                             marker=None,
(17)                             smooth=None,
(17)                             extLst=None,
(17)                             **kw
(16)                            ):
(8)                     self.hiLowLines = hiLowLines
(8)                     self.upDownBars = upDownBars
(8)                     self.marker = marker
(8)                     self.smooth = smooth
(8)                     self.x_axis = TextAxis()
```

```
68: (8)                         self.y_axis = NumericAxis()
69: (8)                         super().__init__(**kw)
70: (0)                 class LineChart3D(_LineChartBase):
71: (4)                     tagname = "line3DChart"
72: (4)                     grouping = _LineChartBase.grouping
73: (4)                     varyColors = _LineChartBase.varyColors
74: (4)                     ser = _LineChartBase.ser
75: (4)                     dLbls = _LineChartBase.dLbls
76: (4)                     dropLines =_LineChartBase.dropLines
77: (4)                     gapDepth = NestedGapAmount()
78: (4)                     hiLowLines = Typed(expected_type=ChartLines, allow_none=True)
79: (4)                     upDownBars = Typed(expected_type=UpDownBars, allow_none=True)
80: (4)                     marker = NestedBool(allow_none=True)
81: (4)                     smooth = NestedBool(allow_none=True)
82: (4)                     extLst = Typed(expected_type=ExtensionList, allow_none=True)
83: (4)                     x_axis = Typed(expected_type=TextAxis)
84: (4)                     y_axis = Typed(expected_type=NumericAxis)
85: (4)                     z_axis = Typed(expected_type=SeriesAxis)
86: (4)                     __elements__ = _LineChartBase.__elements__ + ('gapDepth', 'hiLowLines',
87: (50)                                                                             'upDownBars', 'marker',
'smooth', 'axId')
88: (4)                     def __init__(self,
89: (17)                                 gapDepth=None,
90: (17)                                 hiLowLines=None,
91: (17)                                 upDownBars=None,
92: (17)                                 marker=None,
93: (17)                                 smooth=None,
94: (17)                                 **kw
95: (16)                                ):
96: (8)                         self.gapDepth = gapDepth
97: (8)                         self.hiLowLines = hiLowLines
98: (8)                         self.upDownBars = upDownBars
99: (8)                         self.marker = marker
100: (8)                        self.smooth = smooth
101: (8)                        self.x_axis = TextAxis()
102: (8)                        self.y_axis = NumericAxis()
103: (8)                        self.z_axis = SeriesAxis()
104: (8)                        super(LineChart3D, self).__init__(**kw)

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

File 36 - radar_chart.py:

1: (0)                  from openpyxl.descriptors.serialisable import Serialisable
2: (0)                  from openpyxl.descriptors import (
3: (4)                      Sequence,
4: (4)                      Typed,
5: (4)                      Alias,
6: (0)                  )
7: (0)                  from openpyxl.descriptors.excel import ExtensionList
8: (0)                  from openpyxl.descriptors.nested import (
9: (4)                      NestedBool,
10: (4)                     NestedInteger,
11: (4)                     NestedSet
12: (0)                 )
13: (0)                 from ._chart import ChartBase
14: (0)                 from .axis import TextAxis, NumericAxis
15: (0)                 from .series import Series
16: (0)                 from .label import DataLabelList
17: (0)                 class RadarChart(ChartBase):
18: (4)                     tagname = "radarChart"
19: (4)                     radarStyle = NestedSet(values=(['standard', 'marker', 'filled']))
20: (4)                     type = Alias("radarStyle")
21: (4)                     varyColors = NestedBool(nested=True, allow_none=True)
22: (4)                     ser = Sequence(expected_type=Series, allow_none=True)
23: (4)                     dLbls = Typed(expected_type=DataLabelList, allow_none=True)
24: (4)                     dataLabels = Alias("dLbls")
25: (4)                     extLst = Typed(expected_type=ExtensionList, allow_none=True)
26: (4)                     _series_type = "radar"
```

```
27: (4)             x_axis = Typed(expected_type=TextAxis)
28: (4)             y_axis = Typed(expected_type=NumericAxis)
29: (4)             __elements__ = ('radarStyle', 'varyColors', 'ser', 'dLbls', 'axId')
30: (4)             def __init__(self,
31: (17)                         radarStyle="standard",
32: (17)                         varyColors=None,
33: (17)                         ser=(),
34: (17)                         dLbls=None,
35: (17)                         extLst=None,
36: (17)                         **kw
37: (16)                        ):
38: (8)                 self.radarStyle = radarStyle
39: (8)                 self.varyColors = varyColors
40: (8)                 self.ser = ser
41: (8)                 self.dLbls = dLbls
42: (8)                 self.x_axis = TextAxis()
43: (8)                 self.y_axis = NumericAxis()
44: (8)                 super().__init__(**kw)

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

File 37 - stock_chart.py:

1: (0)          from openpyxl.descriptors.serialisable import Serialisable
2: (0)          from openpyxl.descriptors import (
3: (4)              Typed,
4: (4)              Sequence,
5: (4)              Alias,
6: (0)          )
7: (0)          from openpyxl.descriptors.excel import ExtensionList
8: (0)          from ._chart import ChartBase
9: (0)          from .axis import TextAxis, NumericAxis, ChartLines
10: (0)         from .updown_bars import UpDownBars
11: (0)         from .label import DataLabelList
12: (0)         from .series import Series
13: (0)         class StockChart(ChartBase):
14: (4)             tagname = "stockChart"
15: (4)             ser = Sequence(expected_type=Series) #min 3, max4
16: (4)             dLbls = Typed(expected_type=DataLabelList, allow_none=True)
17: (4)             dataLabels = Alias('dLbls')
18: (4)             dropLines = Typed(expected_type=ChartLines, allow_none=True)
19: (4)             hiLowLines = Typed(expected_type=ChartLines, allow_none=True)
20: (4)             upDownBars = Typed(expected_type=UpDownBars, allow_none=True)
21: (4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
22: (4)             x_axis = Typed(expected_type=TextAxis)
23: (4)             y_axis = Typed(expected_type=NumericAxis)
24: (4)             _series_type = "line"
25: (4)             __elements__ = ('ser', 'dLbls', 'dropLines', 'hiLowLines', 'upDownBars',
26: (20)                            'axId')
27: (4)             def __init__(self,
28: (17)                         ser=(),
29: (17)                         dLbls=None,
30: (17)                         dropLines=None,
31: (17)                         hiLowLines=None,
32: (17)                         upDownBars=None,
33: (17)                         extLst=None,
34: (17)                         **kw
35: (16)                        ):
36: (8)                 self.ser = ser
37: (8)                 self.dLbls = dLbls
38: (8)                 self.dropLines = dropLines
39: (8)                 self.hiLowLines = hiLowLines
40: (8)                 self.upDownBars = upDownBars
41: (8)                 self.x_axis = TextAxis()
42: (8)                 self.y_axis = NumericAxis()
43: (8)                 super().__init__(**kw)

----------------------------------------
```

```
File 38 - updown_bars.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import Typed
3: (0)              from openpyxl.descriptors.excel import ExtensionList
4: (0)              from .shapes import GraphicalProperties
5: (0)              from .axis import ChartLines
6: (0)              from .descriptors import NestedGapAmount
7: (0)              class UpDownBars(Serialisable):
8: (4)                  tagname = "upbars"
9: (4)                  gapWidth = NestedGapAmount()
10: (4)                 upBars = Typed(expected_type=ChartLines, allow_none=True)
11: (4)                 downBars = Typed(expected_type=ChartLines, allow_none=True)
12: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
13: (4)                 __elements__ = ('gapWidth', 'upBars', 'downBars')
14: (4)                 def __init__(self,
15: (17)                             gapWidth=150,
16: (17)                             upBars=None,
17: (17)                             downBars=None,
18: (17)                             extLst=None,
19: (16)                            ):
20: (8)                     self.gapWidth = gapWidth
21: (8)                     self.upBars = upBars
22: (8)                     self.downBars = downBars

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

File 39 - scatter_chart.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Sequence,
5: (4)                  Alias
6: (0)              )
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from openpyxl.descriptors.nested import (
9: (4)                  NestedNoneSet,
10: (4)                 NestedBool,
11: (0)             )
12: (0)             from ._chart import ChartBase
13: (0)             from .axis import NumericAxis, TextAxis
14: (0)             from .series import XYSeries
15: (0)             from .label import DataLabelList
16: (0)             class ScatterChart(ChartBase):
17: (4)                 tagname = "scatterChart"
18: (4)                 scatterStyle = NestedNoneSet(values=(['line', 'lineMarker', 'marker',
'smooth', 'smoothMarker']))
19: (4)                 varyColors = NestedBool(allow_none=True)
20: (4)                 ser = Sequence(expected_type=XYSeries, allow_none=True)
21: (4)                 dLbls = Typed(expected_type=DataLabelList, allow_none=True)
22: (4)                 dataLabels = Alias("dLbls")
23: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
24: (4)                 x_axis = Typed(expected_type=(NumericAxis, TextAxis))
25: (4)                 y_axis = Typed(expected_type=NumericAxis)
26: (4)                 _series_type = "scatter"
27: (4)                 __elements__ = ('scatterStyle', 'varyColors', 'ser', 'dLbls', 'axId',)
28: (4)                 def __init__(self,
29: (17)                             scatterStyle=None,
30: (17)                             varyColors=None,
31: (17)                             ser=(),
32: (17)                             dLbls=None,
33: (17)                             extLst=None,
34: (17)                             **kw
35: (16)                            ):
36: (8)                     self.scatterStyle = scatterStyle
37: (8)                     self.varyColors = varyColors
38: (8)                     self.ser = ser
39: (8)                     self.dLbls = dLbls
```

```
40: (8)                     self.x_axis = NumericAxis(axId=10, crossAx=20)
41: (8)                     self.y_axis = NumericAxis(axId=20, crossAx=10)
42: (8)                     super().__init__(**kw)

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

File 40 - surface_chart.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Integer,
5: (4)                  Bool,
6: (4)                  Alias,
7: (4)                  Sequence,
8: (0)              )
9: (0)              from openpyxl.descriptors.excel import ExtensionList
10: (0)             from openpyxl.descriptors.nested import (
11: (4)                 NestedInteger,
12: (4)                 NestedBool,
13: (0)             )
14: (0)             from ._chart import ChartBase
15: (0)             from ._3d import _3DBase
16: (0)             from .axis import TextAxis, NumericAxis, SeriesAxis
17: (0)             from .shapes import GraphicalProperties
18: (0)             from .series import Series
19: (0)             class BandFormat(Serialisable):
20: (4)                 tagname = "bandFmt"
21: (4)                 idx = NestedInteger()
22: (4)                 spPr = Typed(expected_type=GraphicalProperties, allow_none=True)
23: (4)                 graphicalProperties = Alias("spPr")
24: (4)                 __elements__ = ('idx', 'spPr')
25: (4)                 def __init__(self,
26: (17)                             idx=0,
27: (17)                             spPr=None,
28: (16)                            ):
29: (8)                     self.idx = idx
30: (8)                     self.spPr = spPr
31: (0)             class BandFormatList(Serialisable):
32: (4)                 tagname = "bandFmts"
33: (4)                 bandFmt = Sequence(expected_type=BandFormat, allow_none=True)
34: (4)                 __elements__ = ('bandFmt',)
35: (4)                 def __init__(self,
36: (17)                             bandFmt=(),
37: (16)                            ):
38: (8)                     self.bandFmt = bandFmt
39: (0)             class _SurfaceChartBase(ChartBase):
40: (4)                 wireframe = NestedBool(allow_none=True)
41: (4)                 ser = Sequence(expected_type=Series, allow_none=True)
42: (4)                 bandFmts = Typed(expected_type=BandFormatList, allow_none=True)
43: (4)                 _series_type = "surface"
44: (4)                 __elements__ = ('wireframe', 'ser', 'bandFmts')
45: (4)                 def __init__(self,
46: (17)                             wireframe=None,
47: (17)                             ser=(),
48: (17)                             bandFmts=None,
49: (17)                             **kw
50: (16)                            ):
51: (8)                     self.wireframe = wireframe
52: (8)                     self.ser = ser
53: (8)                     self.bandFmts = bandFmts
54: (8)                     super().__init__(**kw)
55: (0)             class SurfaceChart3D(_SurfaceChartBase, _3DBase):
56: (4)                 tagname = "surface3DChart"
57: (4)                 wireframe = _SurfaceChartBase.wireframe
58: (4)                 ser = _SurfaceChartBase.ser
59: (4)                 bandFmts = _SurfaceChartBase.bandFmts
60: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
61: (4)                 x_axis = Typed(expected_type=TextAxis)
```

```
62: (4)                 y_axis = Typed(expected_type=NumericAxis)
63: (4)                 z_axis = Typed(expected_type=SeriesAxis)
64: (4)                 __elements__ = _SurfaceChartBase.__elements__ + ('axId',)
65: (4)                 def __init__(self, **kw):
66: (8)                     self.x_axis = TextAxis()
67: (8)                     self.y_axis = NumericAxis()
68: (8)                     self.z_axis = SeriesAxis()
69: (8)                     super(SurfaceChart3D, self).__init__(**kw)
70: (0)             class SurfaceChart(SurfaceChart3D):
71: (4)                 tagname = "surfaceChart"
72: (4)                 wireframe = _SurfaceChartBase.wireframe
73: (4)                 ser = _SurfaceChartBase.ser
74: (4)                 bandFmts = _SurfaceChartBase.bandFmts
75: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
76: (4)                 __elements__ = SurfaceChart3D.__elements__
77: (4)                 def __init__(self, **kw):
78: (8)                     super().__init__(**kw)
79: (8)                     self.y_axis.delete = True
80: (8)                     self.view3D.x_rotation = 90
81: (8)                     self.view3D.y_rotation = 0
82: (8)                     self.view3D.perspective = False
83: (8)                     self.view3D.right_angle_axes = False

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

File 41 - print_settings.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Float,
4: (4)                  Typed,
5: (4)                  Alias,
6: (0)              )
7: (0)              from openpyxl.worksheet.page import PrintPageSetup
8: (0)              from openpyxl.worksheet.header_footer import HeaderFooter
9: (0)              class PageMargins(Serialisable):
10: (4)                 """
11: (4)                 Identical to openpyxl.worksheet.page.Pagemargins but element names are
different :-/
12: (4)                 """
13: (4)                 tagname = "pageMargins"
14: (4)                 l = Float()
15: (4)                 left = Alias('l')
16: (4)                 r = Float()
17: (4)                 right = Alias('r')
18: (4)                 t = Float()
19: (4)                 top = Alias('t')
20: (4)                 b = Float()
21: (4)                 bottom = Alias('b')
22: (4)                 header = Float()
23: (4)                 footer = Float()
24: (4)                 def __init__(self, l=0.75, r=0.75, t=1, b=1, header=0.5, footer=0.5):
25: (8)                     self.l = l
26: (8)                     self.r = r
27: (8)                     self.t = t
28: (8)                     self.b = b
29: (8)                     self.header = header
30: (8)                     self.footer = footer
31: (0)             class PrintSettings(Serialisable):
32: (4)                 tagname = "printSettings"
33: (4)                 headerFooter = Typed(expected_type=HeaderFooter, allow_none=True)
34: (4)                 pageMargins = Typed(expected_type=PageMargins, allow_none=True)
35: (4)                 pageSetup = Typed(expected_type=PrintPageSetup, allow_none=True)
36: (4)                 __elements__ = ("headerFooter", "pageMargins", "pageMargins")
37: (4)                 def __init__(self,
38: (17)                             headerFooter=None,
39: (17)                             pageMargins=None,
40: (17)                             pageSetup=None,
41: (16)                            ):
```

```
42: (8)                     self.headerFooter = headerFooter
43: (8)                     self.pageMargins = pageMargins
44: (8)                     self.pageSetup = pageSetup
```

File 42 - series_factory.py:

```
1: (0)              from .data_source import NumDataSource, NumRef, AxDataSource
2: (0)              from .reference import Reference
3: (0)              from .series import Series, XYSeries, SeriesLabel, StrRef
4: (0)              from  openpyxl.utils import rows_from_range, quote_sheetname
5: (0)              def SeriesFactory(values, xvalues=None, zvalues=None, title=None,
title_from_data=False):
6: (4)                  """
7: (4)                  Convenience Factory for creating chart data series.
8: (4)                  """
9: (4)                  if not isinstance(values, Reference):
10: (8)                     values = Reference(range_string=values)
11: (4)                 if title_from_data:
12: (8)                     cell = values.pop()
13: (8)                     title = u"{0}!{1}".format(values.sheetname, cell)
14: (8)                     title = SeriesLabel(strRef=StrRef(title))
15: (4)                 elif title is not None:
16: (8)                     title = SeriesLabel(v=title)
17: (4)                 source = NumDataSource(numRef=NumRef(f=values))
18: (4)                 if xvalues is not None:
19: (8)                     if not isinstance(xvalues, Reference):
20: (12)                        xvalues = Reference(range_string=xvalues)
21: (8)                     series = XYSeries()
22: (8)                     series.yVal = source
23: (8)                     series.xVal = AxDataSource(numRef=NumRef(f=xvalues))
24: (8)                     if zvalues is not None:
25: (12)                        if not isinstance(zvalues, Reference):
26: (16)                            zvalues = Reference(range_string=zvalues)
27: (12)                        series.zVal = NumDataSource(NumRef(f=zvalues))
28: (4)                 else:
29: (8)                     series = Series()
30: (8)                     series.val = source
31: (4)                 if title is not None:
32: (8)                     series.title = title
33: (4)                 return series
```

File 43 - views.py:

```
1: (0)              from openpyxl.descriptors import (
2: (4)                  Bool,
3: (4)                  Integer,
4: (4)                  Typed,
5: (4)                  Sequence
6: (0)              )
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from openpyxl.descriptors.serialisable import Serialisable
9: (0)              class ChartsheetView(Serialisable):
10: (4)                 tagname = "sheetView"
11: (4)                 tabSelected = Bool(allow_none=True)
12: (4)                 zoomScale = Integer(allow_none=True)
13: (4)                 workbookViewId = Integer()
14: (4)                 zoomToFit = Bool(allow_none=True)
15: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
16: (4)                 __elements__ = ()
17: (4)                 def __init__(self,
18: (17)                             tabSelected=None,
19: (17)                             zoomScale=None,
20: (17)                             workbookViewId=0,
21: (17)                             zoomToFit=True,
22: (17)                             extLst=None,
23: (17)                        ):
24: (8)                     self.tabSelected = tabSelected
25: (8)                     self.zoomScale = zoomScale
26: (8)                     self.workbookViewId = workbookViewId
27: (8)                     self.zoomToFit = zoomToFit
28: (0)             class ChartsheetViewList(Serialisable):
29: (4)                 tagname = "sheetViews"
30: (4)                 sheetView = Sequence(expected_type=ChartsheetView, )
31: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
32: (4)                 __elements__ = ('sheetView',)
33: (4)                 def __init__(self,
34: (17)                             sheetView=None,
35: (17)                             extLst=None,
36: (17)                             ):
37: (8)                     if sheetView is None:
38: (12)                        sheetView = [ChartsheetView()]
39: (8)                     self.sheetView = sheetView

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

File 44 - custom.py:

1: (0)              from openpyxl.worksheet.header_footer import HeaderFooter
2: (0)              from openpyxl.descriptors import (
3: (4)                  Bool,
4: (4)                  Integer,
5: (4)                  Set,
6: (4)                  Typed,
7: (4)                  Sequence
8: (0)              )
9: (0)              from openpyxl.descriptors.excel import Guid
10: (0)             from openpyxl.descriptors.serialisable import Serialisable
11: (0)             from openpyxl.worksheet.page import (
12: (4)                 PageMargins,
13: (4)                 PrintPageSetup
14: (0)             )
15: (0)             class CustomChartsheetView(Serialisable):
16: (4)                 tagname = "customSheetView"
17: (4)                 guid = Guid()
18: (4)                 scale = Integer()
19: (4)                 state = Set(values=(['visible', 'hidden', 'veryHidden']))
20: (4)                 zoomToFit = Bool(allow_none=True)
21: (4)                 pageMargins = Typed(expected_type=PageMargins, allow_none=True)
22: (4)                 pageSetup = Typed(expected_type=PrintPageSetup, allow_none=True)
23: (4)                 headerFooter = Typed(expected_type=HeaderFooter, allow_none=True)
24: (4)                 __elements__ = ('pageMargins', 'pageSetup', 'headerFooter')
25: (4)                 def __init__(self,
26: (17)                             guid=None,
27: (17)                             scale=None,
28: (17)                             state='visible',
29: (17)                             zoomToFit=None,
30: (17)                             pageMargins=None,
31: (17)                             pageSetup=None,
32: (17)                             headerFooter=None,
33: (17)                             ):
34: (8)                     self.guid = guid
35: (8)                     self.scale = scale
36: (8)                     self.state = state
37: (8)                     self.zoomToFit = zoomToFit
38: (8)                     self.pageMargins = pageMargins
39: (8)                     self.pageSetup = pageSetup
40: (8)                     self.headerFooter = headerFooter
41: (0)             class CustomChartsheetViews(Serialisable):
42: (4)                 tagname = "customSheetViews"
43: (4)                 customSheetView = Sequence(expected_type=CustomChartsheetView,
allow_none=True)
44: (4)                 __elements__ = ('customSheetView',)
45: (4)                 def __init__(self,
46: (17)                             customSheetView=None,
47: (17)                            ):
48: (8)                     self.customSheetView = customSheetView
----------------------------------------
File 45 - author.py:
1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence,
4: (4)                  Alias
5: (0)              )
6: (0)              class AuthorList(Serialisable):
7: (4)                  tagname = "authors"
8: (4)                  author = Sequence(expected_type=str)
9: (4)                  authors = Alias("author")
10: (4)                 def __init__(self,
11: (17)                             author=(),
12: (16)                            ):
13: (8)                     self.author = author
----------------------------------------
File 46 - publish.py:
1: (0)              from openpyxl.descriptors import (
2: (4)                  Bool,
3: (4)                  Integer,
4: (4)                  String,
5: (4)                  Set,
6: (4)                  Sequence
7: (0)              )
8: (0)              from openpyxl.descriptors.serialisable import Serialisable
9: (0)              class WebPublishItem(Serialisable):
10: (4)                 tagname = "webPublishItem"
11: (4)                 id = Integer()
12: (4)                 divId = String()
13: (4)                 sourceType = Set(values=(['sheet', 'printArea', 'autoFilter', 'range',
'chart', 'pivotTable', 'query', 'label']))
14: (4)                 sourceRef = String()
15: (4)                 sourceObject = String(allow_none=True)
16: (4)                 destinationFile = String()
17: (4)                 title = String(allow_none=True)
18: (4)                 autoRepublish = Bool(allow_none=True)
19: (4)                 def __init__(self,
20: (17)                             id=None,
21: (17)                             divId=None,
22: (17)                             sourceType=None,
23: (17)                             sourceRef=None,
24: (17)                             sourceObject=None,
25: (17)                             destinationFile=None,
26: (17)                             title=None,
27: (17)                             autoRepublish=None,
28: (17)                             ):
29: (8)                     self.id = id
30: (8)                     self.divId = divId
31: (8)                     self.sourceType = sourceType
32: (8)                     self.sourceRef = sourceRef
33: (8)                     self.sourceObject = sourceObject
34: (8)                     self.destinationFile = destinationFile
35: (8)                     self.title = title
36: (8)                     self.autoRepublish = autoRepublish
37: (0)              class WebPublishItems(Serialisable):
38: (4)                 tagname = "WebPublishItems"
39: (4)                 count = Integer(allow_none=True)
40: (4)                 webPublishItem = Sequence(expected_type=WebPublishItem, )
41: (4)                 __elements__ = ('webPublishItem',)
42: (4)                 def __init__(self,
43: (17)                             count=None,
44: (17)                        webPublishItem=None,
45: (17)                        ):
46: (8)                     self.count = len(webPublishItem)
47: (8)                     self.webPublishItem = webPublishItem

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

File 47 - relation.py:

1: (0)              from openpyxl.descriptors import (
2: (4)                  Integer,
3: (4)                  Alias
4: (0)              )
5: (0)              from openpyxl.descriptors.excel import Relation
6: (0)              from openpyxl.descriptors.serialisable import Serialisable
7: (0)              class SheetBackgroundPicture(Serialisable):
8: (4)                  tagname = "picture"
9: (4)                  id = Relation()
10: (4)                 def __init__(self, id):
11: (8)                     self.id = id
12: (0)             class DrawingHF(Serialisable):
13: (4)                 id = Relation()
14: (4)                 lho = Integer(allow_none=True)
15: (4)                 leftHeaderOddPages = Alias('lho')
16: (4)                 lhe = Integer(allow_none=True)
17: (4)                 leftHeaderEvenPages = Alias('lhe')
18: (4)                 lhf = Integer(allow_none=True)
19: (4)                 leftHeaderFirstPage = Alias('lhf')
20: (4)                 cho = Integer(allow_none=True)
21: (4)                 centerHeaderOddPages = Alias('cho')
22: (4)                 che = Integer(allow_none=True)
23: (4)                 centerHeaderEvenPages = Alias('che')
24: (4)                 chf = Integer(allow_none=True)
25: (4)                 centerHeaderFirstPage = Alias('chf')
26: (4)                 rho = Integer(allow_none=True)
27: (4)                 rightHeaderOddPages = Alias('rho')
28: (4)                 rhe = Integer(allow_none=True)
29: (4)                 rightHeaderEvenPages = Alias('rhe')
30: (4)                 rhf = Integer(allow_none=True)
31: (4)                 rightHeaderFirstPage = Alias('rhf')
32: (4)                 lfo = Integer(allow_none=True)
33: (4)                 leftFooterOddPages = Alias('lfo')
34: (4)                 lfe = Integer(allow_none=True)
35: (4)                 leftFooterEvenPages = Alias('lfe')
36: (4)                 lff = Integer(allow_none=True)
37: (4)                 leftFooterFirstPage = Alias('lff')
38: (4)                 cfo = Integer(allow_none=True)
39: (4)                 centerFooterOddPages = Alias('cfo')
40: (4)                 cfe = Integer(allow_none=True)
41: (4)                 centerFooterEvenPages = Alias('cfe')
42: (4)                 cff = Integer(allow_none=True)
43: (4)                 centerFooterFirstPage = Alias('cff')
44: (4)                 rfo = Integer(allow_none=True)
45: (4)                 rightFooterOddPages = Alias('rfo')
46: (4)                 rfe = Integer(allow_none=True)
47: (4)                 rightFooterEvenPages = Alias('rfe')
48: (4)                 rff = Integer(allow_none=True)
49: (4)                 rightFooterFirstPage = Alias('rff')
50: (4)                 def __init__(self,
51: (17)                             id=None,
52: (17)                             lho=None,
53: (17)                             lhe=None,
54: (17)                             lhf=None,
55: (17)                             cho=None,
56: (17)                             che=None,
57: (17)                             chf=None,
58: (17)                             rho=None,
59: (17)                             rhe=None,
60: (17)                             rhf=None,
61: (17)                         lfo=None,
62: (17)                         lfe=None,
63: (17)                         lff=None,
64: (17)                         cfo=None,
65: (17)                         cfe=None,
66: (17)                         cff=None,
67: (17)                         rfo=None,
68: (17)                         rfe=None,
69: (17)                         rff=None,
70: (17)                         ):
71: (8)                 self.id = id
72: (8)                 self.lho = lho
73: (8)                 self.lhe = lhe
74: (8)                 self.lhf = lhf
75: (8)                 self.cho = cho
76: (8)                 self.che = che
77: (8)                 self.chf = chf
78: (8)                 self.rho = rho
79: (8)                 self.rhe = rhe
80: (8)                 self.rhf = rhf
81: (8)                 self.lfo = lfo
82: (8)                 self.lfe = lfe
83: (8)                 self.lff = lff
84: (8)                 self.cfo = cfo
85: (8)                 self.cfe = cfe
86: (8)                 self.cff = cff
87: (8)                 self.rfo = rfo
88: (8)                 self.rfe = rfe
89: (8)                 self.rff = rff

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

File 48 - __init__.py:

1: (0)              from .chartsheet import Chartsheet

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

File 49 - comments.py:

1: (0)              class Comment:
2: (4)                  _parent = None
3: (4)                  def __init__(self, text, author, height=79, width=144):
4: (8)                      self.content = text
5: (8)                      self.author = author
6: (8)                      self.height = height
7: (8)                      self.width = width
8: (4)                  @property
9: (4)                  def parent(self):
10: (8)                     return self._parent
11: (4)                 def __eq__(self, other):
12: (8)                     return (
13: (12)                        self.content == other.content
14: (12)                        and self.author == other.author
15: (8)                     )
16: (4)                 def __repr__(self):
17: (8)                     return "Comment: {0} by {1}".format(self.content, self.author)
18: (4)                 def __copy__(self):
19: (8)                     """Create a detached copy of this comment."""
20: (8)                     clone = self.__class__(self.content, self.author, self.height,
self.width)
21: (8)                     return clone
22: (4)                 def bind(self, cell):
23: (8)                     """
24: (8)                     Bind comment to a particular cell
25: (8)                     """
26: (8)                     if cell is not None and self._parent is not None and self._parent !=
cell:
27: (12)                        fmt = "Comment already assigned to {0} in worksheet {1}. Cannot
```

```
assign a comment to more than one cell"
28: (12)                        raise AttributeError(fmt.format(cell.coordinate,
cell.parent.title))
29: (8)                     self._parent = cell
30: (4)                 def unbind(self):
31: (8)                     """
32: (8)                     Unbind a comment from a cell
33: (8)                     """
34: (8)                     self._parent = None
35: (4)                 @property
36: (4)                 def text(self):
37: (8)                     """
38: (8)                     Any comment text stripped of all formatting.
39: (8)                     """
40: (8)                     return self.content
41: (4)                 @text.setter
42: (4)                 def text(self, value):
43: (8)                     self.content = value

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

File 50 - __init__.py:

1: (0)              from .comments import Comment

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

File 51 - chartsheet.py:

1: (0)              from openpyxl.descriptors import Typed, Set, Alias
2: (0)              from openpyxl.descriptors.excel import ExtensionList
3: (0)              from openpyxl.descriptors.serialisable import Serialisable
4: (0)              from openpyxl.drawing.spreadsheet_drawing import (
5: (4)                  AbsoluteAnchor,
6: (4)                  SpreadsheetDrawing,
7: (0)              )
8: (0)              from openpyxl.worksheet.page import (
9: (4)                  PageMargins,
10: (4)                 PrintPageSetup
11: (0)             )
12: (0)             from openpyxl.worksheet.drawing import Drawing
13: (0)             from openpyxl.worksheet.header_footer import HeaderFooter
14: (0)             from openpyxl.workbook.child import _WorkbookChild
15: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS, REL_NS
16: (0)             from .relation import DrawingHF, SheetBackgroundPicture
17: (0)             from .properties import ChartsheetProperties
18: (0)             from .protection import ChartsheetProtection
19: (0)             from .views import ChartsheetViewList
20: (0)             from .custom import CustomChartsheetViews
21: (0)             from .publish import WebPublishItems
22: (0)             class Chartsheet(_WorkbookChild, Serialisable):
23: (4)                 tagname = "chartsheet"
24: (4)                 _default_title = "Chart"
25: (4)                 _rel_type = "chartsheet"
26: (4)                 _path = "/xl/chartsheets/sheet{0}.xml"
27: (4)                 mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.chartsheet+xml"
28: (4)                 sheetPr = Typed(expected_type=ChartsheetProperties, allow_none=True)
29: (4)                 sheetViews = Typed(expected_type=ChartsheetViewList)
30: (4)                 sheetProtection = Typed(expected_type=ChartsheetProtection,
allow_none=True)
31: (4)                 customSheetViews = Typed(expected_type=CustomChartsheetViews,
allow_none=True)
32: (4)                 pageMargins = Typed(expected_type=PageMargins, allow_none=True)
33: (4)                 pageSetup = Typed(expected_type=PrintPageSetup, allow_none=True)
34: (4)                 drawing = Typed(expected_type=Drawing, allow_none=True)
35: (4)                 drawingHF = Typed(expected_type=DrawingHF, allow_none=True)
36: (4)                 picture = Typed(expected_type=SheetBackgroundPicture, allow_none=True)
37: (4)                 webPublishItems = Typed(expected_type=WebPublishItems, allow_none=True)
```

```
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         sheet_state = Set(values=('visible', 'hidden', 'veryHidden'))
(4)         headerFooter = Typed(expected_type=HeaderFooter)
(4)         HeaderFooter = Alias('headerFooter')
(4)         __elements__ = (
(8)             'sheetPr', 'sheetViews', 'sheetProtection', 'customSheetViews',
(8)             'pageMargins', 'pageSetup', 'headerFooter', 'drawing', 'drawingHF',
(8)             'picture', 'webPublishItems')
(4)         __attrs__ = ()
(4)         def __init__(self,
(17)                     sheetPr=None,
(17)                     sheetViews=None,
(17)                     sheetProtection=None,
(17)                     customSheetViews=None,
(17)                     pageMargins=None,
(17)                     pageSetup=None,
(17)                     headerFooter=None,
(17)                     drawing=None,
(17)                     drawingHF=None,
(17)                     picture=None,
(17)                     webPublishItems=None,
(17)                     extLst=None,
(17)                     parent=None,
(17)                     title="",
(17)                     sheet_state='visible',
(17)                     ):
(8)             super().__init__(parent, title)
(8)             self._charts = []
(8)             self.sheetPr = sheetPr
(8)             if sheetViews is None:
(12)                sheetViews = ChartsheetViewList()
(8)             self.sheetViews = sheetViews
(8)             self.sheetProtection = sheetProtection
(8)             self.customSheetViews = customSheetViews
(8)             self.pageMargins = pageMargins
(8)             self.pageSetup = pageSetup
(8)             if headerFooter is not None:
(12)                self.headerFooter = headerFooter
(8)             self.drawing = Drawing("rId1")
(8)             self.drawingHF = drawingHF
(8)             self.picture = picture
(8)             self.webPublishItems = webPublishItems
(8)             self.sheet_state = sheet_state
(4)         def add_chart(self, chart):
(8)             chart.anchor = AbsoluteAnchor()
(8)             self._charts.append(chart)
(4)         def to_tree(self):
(8)             self._drawing = SpreadsheetDrawing()
(8)             self._drawing.charts = self._charts
(8)             tree = super().to_tree()
(8)             if not self.headerFooter:
(12)                el = tree.find('headerFooter')
(12)                tree.remove(el)
(8)             tree.set("xmlns", SHEET_MAIN_NS)
(8)             return tree
```

File 52 - properties.py:

```
(0)     from openpyxl.descriptors import (
(4)         Bool,
(4)         String,
(4)         Typed
(0)     )
(0)     from openpyxl.descriptors.serialisable import Serialisable
(0)     from openpyxl.styles import Color
(0)     class ChartsheetProperties(Serialisable):
(4)         tagname = "sheetPr"
```

```
10: (4)                 published = Bool(allow_none=True)
11: (4)                 codeName = String(allow_none=True)
12: (4)                 tabColor = Typed(expected_type=Color, allow_none=True)
13: (4)                 __elements__ = ('tabColor',)
14: (4)                 def __init__(self,
15: (17)                             published=None,
16: (17)                             codeName=None,
17: (17)                             tabColor=None,
18: (17)                             ):
19: (8)                     self.published = published
20: (8)                     self.codeName = codeName
21: (8)                     self.tabColor = tabColor

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

File 53 - protection.py:

1: (0)              import hashlib
2: (0)              from openpyxl.descriptors import (Bool, Integer, String)
3: (0)              from openpyxl.descriptors.excel import Base64Binary
4: (0)              from openpyxl.descriptors.serialisable import Serialisable
5: (0)              from openpyxl.worksheet.protection import (
6: (4)                  hash_password,
7: (4)                  _Protected
8: (0)              )
9: (0)              class ChartsheetProtection(Serialisable, _Protected):
10: (4)                 tagname = "sheetProtection"
11: (4)                 algorithmName = String(allow_none=True)
12: (4)                 hashValue = Base64Binary(allow_none=True)
13: (4)                 saltValue = Base64Binary(allow_none=True)
14: (4)                 spinCount = Integer(allow_none=True)
15: (4)                 content = Bool(allow_none=True)
16: (4)                 objects = Bool(allow_none=True)
17: (4)                 __attrs__ = ("content", "objects", "password", "hashValue", "spinCount",
"saltValue", "algorithmName")
18: (4)                 def __init__(self,
19: (17)                             content=None,
20: (17)                             objects=None,
21: (17)                             hashValue=None,
22: (17)                             spinCount=None,
23: (17)                             saltValue=None,
24: (17)                             algorithmName=None,
25: (17)                             password=None,
26: (17)                             ):
27: (8)                     self.content = content
28: (8)                     self.objects = objects
29: (8)                     self.hashValue = hashValue
30: (8)                     self.spinCount = spinCount
31: (8)                     self.saltValue = saltValue
32: (8)                     self.algorithmName = algorithmName
33: (8)                     if password is not None:
34: (12)                        self.password = password

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

File 54 - shape_writer.py:

1: (0)              from openpyxl.xml.functions import (
2: (4)                  Element,
3: (4)                  SubElement,
4: (4)                  tostring,
5: (0)              )
6: (0)              from openpyxl.utils import coordinate_to_tuple
7: (0)              vmlns = "urn:schemas-microsoft-com:vml"
8: (0)              officens = "urn:schemas-microsoft-com:office:office"
9: (0)              excelns = "urn:schemas-microsoft-com:office:excel"
10: (0)             class ShapeWriter:
11: (4)                 """
12: (4)                 Create VML for comments
```

```
13: (4)                 """
14: (4)                 vml = None
15: (4)                 vml_path = None
16: (4)                 def __init__(self, comments):
17: (8)                     self.comments = comments
18: (4)                 def add_comment_shapetype(self, root):
19: (8)                     shape_layout = SubElement(root, "{%s}shapelayout" % officens,
20: (34)                                              {"{%s}ext" % vmlns: "edit"})
21: (8)                     SubElement(shape_layout,
22: (19)                               "{%s}idmap" % officens,
23: (19)                               {"{%s}ext" % vmlns: "edit", "data": "1"})
24: (8)                     shape_type = SubElement(root,
25: (32)                                            "{%s}shapetype" % vmlns,
26: (32)                                            {"id": "_x0000_t202",
27: (33)                                             "coordsize": "21600,21600",
28: (33)                                             "{%s}spt" % officens: "202",
29: (33)                                             "path": "m,l,21600r21600,l21600,xe"})
30: (8)                     SubElement(shape_type, "{%s}stroke" % vmlns, {"joinstyle": "miter"})
31: (8)                     SubElement(shape_type,
32: (19)                               "{%s}path" % vmlns,
33: (19)                               {"gradientshapeok": "t",
34: (20)                                "{%s}connecttype" % officens: "rect"})
35: (4)                 def add_comment_shape(self, root, idx, coord, height, width):
36: (8)                     row, col = coordinate_to_tuple(coord)
37: (8)                     row -= 1
38: (8)                     col -= 1
39: (8)                     shape = _shape_factory(row, col, height, width)
40: (8)                     shape.set('id', "_x0000_s%04d" % idx)
41: (8)                     root.append(shape)
42: (4)                 def write(self, root):
43: (8)                     if not hasattr(root, "findall"):
44: (12)                        root = Element("xml")
45: (8)                     comments = root.findall("{%s}shape[@type='#_x0000_t202']" % vmlns)
46: (8)                     for c in comments:
47: (12)                        root.remove(c)
48: (8)                     shape_types = root.find("{%s}shapetype[@id='_x0000_t202']" % vmlns)
49: (8)                     if shape_types is None:
50: (12)                        self.add_comment_shapetype(root)
51: (8)                     for idx, (coord, comment) in enumerate(self.comments, 1026):
52: (12)                        self.add_comment_shape(root, idx, coord, comment.height,
comment.width)
53: (8)                     return tostring(root)
54: (0)             def _shape_factory(row, column, height, width):
55: (4)                 style = ("position:absolute; "
56: (13)                         "margin-left:59.25pt;"
57: (13)                         "margin-top:1.5pt;"
58: (13)                         "width:{width}px;"
59: (13)                         "height:{height}px;"
60: (13)                         "z-index:1;"
61: (13)                         "visibility:hidden").format(height=height,
62: (41)                                                     width=width)
63: (4)                 attrs = {
64: (8)                     "type": "#_x0000_t202",
65: (8)                     "style": style,
66: (8)                     "fillcolor": "#ffffe1",
67: (8)                     "{%s}insetmode" % officens: "auto"
68: (4)                 }
69: (4)                 shape = Element("{%s}shape" % vmlns, attrs)
70: (4)                 SubElement(shape, "{%s}fill" % vmlns,
71: (15)                           {"color2": "#ffffe1"})
72: (4)                 SubElement(shape, "{%s}shadow" % vmlns,
73: (15)                           {"color": "black", "obscured": "t"})
74: (4)                 SubElement(shape, "{%s}path" % vmlns,
75: (15)                           {"{%s}connecttype" % officens: "none"})
76: (4)                 textbox = SubElement(shape, "{%s}textbox" % vmlns,
77: (25)                                     {"style": "mso-direction-alt:auto"})
78: (4)                 SubElement(textbox, "div", {"style": "text-align:left"})
79: (4)                 client_data = SubElement(shape, "{%s}ClientData" % excelns,
80: (29)                                         {"ObjectType": "Note"})
```

```
81: (4)                 SubElement(client_data, "{%s}MoveWithCells" % excelns)
82: (4)                 SubElement(client_data, "{%s}SizeWithCells" % excelns)
83: (4)                 SubElement(client_data, "{%s}AutoFill" % excelns).text = "False"
84: (4)                 SubElement(client_data, "{%s}Row" % excelns).text = str(row)
85: (4)                 SubElement(client_data, "{%s}Column" % excelns).text = str(column)
86: (4)                 return shape

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

File 55 - comment_sheet.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Integer,
5: (4)                  Set,
6: (4)                  String,
7: (4)                  Bool,
8: (0)              )
9: (0)              from openpyxl.descriptors.excel import Guid, ExtensionList
10: (0)             from openpyxl.descriptors.sequence import NestedSequence
11: (0)             from openpyxl.utils.indexed_list import IndexedList
12: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
13: (0)             from openpyxl.cell.text import Text
14: (0)             from .author import AuthorList
15: (0)             from .comments import Comment
16: (0)             from .shape_writer import ShapeWriter
17: (0)             class Properties(Serialisable):
18: (4)                 locked = Bool(allow_none=True)
19: (4)                 defaultSize = Bool(allow_none=True)
20: (4)                 _print = Bool(allow_none=True)
21: (4)                 disabled = Bool(allow_none=True)
22: (4)                 uiObject = Bool(allow_none=True)
23: (4)                 autoFill = Bool(allow_none=True)
24: (4)                 autoLine = Bool(allow_none=True)
25: (4)                 altText = String(allow_none=True)
26: (4)                 textHAlign = Set(values=(['left', 'center', 'right', 'justify',
'distributed']))
27: (4)                 textVAlign = Set(values=(['top', 'center', 'bottom', 'justify',
'distributed']))
28: (4)                 lockText = Bool(allow_none=True)
29: (4)                 justLastX = Bool(allow_none=True)
30: (4)                 autoScale = Bool(allow_none=True)
31: (4)                 rowHidden = Bool(allow_none=True)
32: (4)                 colHidden = Bool(allow_none=True)
33: (4)                 __elements__ = ('anchor',)
34: (4)                 def __init__(self,
35: (17)                             locked=None,
36: (17)                             defaultSize=None,
37: (17)                             _print=None,
38: (17)                             disabled=None,
39: (17)                             uiObject=None,
40: (17)                             autoFill=None,
41: (17)                             autoLine=None,
42: (17)                             altText=None,
43: (17)                             textHAlign=None,
44: (17)                             textVAlign=None,
45: (17)                             lockText=None,
46: (17)                             justLastX=None,
47: (17)                             autoScale=None,
48: (17)                             rowHidden=None,
49: (17)                             colHidden=None,
50: (17)                             anchor=None,
51: (16)                            ):
52: (8)                     self.locked = locked
53: (8)                     self.defaultSize = defaultSize
54: (8)                     self._print = _print
55: (8)                     self.disabled = disabled
56: (8)                     self.uiObject = uiObject
```

```
(8)                     self.autoFill = autoFill
(8)                     self.autoLine = autoLine
(8)                     self.altText = altText
(8)                     self.textHAlign = textHAlign
(8)                     self.textVAlign = textVAlign
(8)                     self.lockText = lockText
(8)                     self.justLastX = justLastX
(8)                     self.autoScale = autoScale
(8)                     self.rowHidden = rowHidden
(8)                     self.colHidden = colHidden
(8)                     self.anchor = anchor
(0)             class CommentRecord(Serialisable):
(4)                 tagname = "comment"
(4)                 ref = String()
(4)                 authorId = Integer()
(4)                 guid = Guid(allow_none=True)
(4)                 shapeId = Integer(allow_none=True)
(4)                 text = Typed(expected_type=Text)
(4)                 commentPr = Typed(expected_type=Properties, allow_none=True)
(4)                 author = String(allow_none=True)
(4)                 __elements__ = ('text', 'commentPr')
(4)                 __attrs__ = ('ref', 'authorId', 'guid', 'shapeId')
(4)                 def __init__(self,
(17)                             ref="",
(17)                             authorId=0,
(17)                             guid=None,
(17)                             shapeId=0,
(17)                             text=None,
(17)                             commentPr=None,
(17)                             author=None,
(17)                             height=79,
(17)                             width=144
(16)                            ):
(8)                     self.ref = ref
(8)                     self.authorId = authorId
(8)                     self.guid = guid
(8)                     self.shapeId = shapeId
(8)                     if text is None:
(12)                        text = Text()
(8)                     self.text = text
(8)                     self.commentPr = commentPr
(8)                     self.author = author
(8)                     self.height = height
(8)                     self.width = width
(4)                 @classmethod
(4)                 def from_cell(cls, cell):
(8)                     """
(8)                     Class method to convert cell comment
(8)                     """
(8)                     comment = cell._comment
(8)                     ref = cell.coordinate
(8)                     self = cls(ref=ref, author=comment.author)
(8)                     self.text.t = comment.content
(8)                     self.height = comment.height
(8)                     self.width = comment.width
(8)                     return self
(4)                 @property
(4)                 def content(self):
(8)                     """
(8)                     Remove all inline formatting and stuff
(8)                     """
(8)                     return self.text.content
(0)             class CommentSheet(Serialisable):
(4)                 tagname = "comments"
(4)                 authors = Typed(expected_type=AuthorList)
(4)                 commentList = NestedSequence(expected_type=CommentRecord, count=0)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 _id = None
(4)                 _path = "/xl/comments/comment{0}.xml"
```

```
126: (4)                mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.comments+xml"
127: (4)                _rel_type = "comments"
128: (4)                _rel_id = None
129: (4)                __elements__ = ('authors', 'commentList')
130: (4)                def __init__(self,
131: (17)                            authors=None,
132: (17)                            commentList=None,
133: (17)                            extLst=None,
134: (16)                           ):
135: (8)                    self.authors = authors
136: (8)                    self.commentList = commentList
137: (4)                def to_tree(self):
138: (8)                    tree = super().to_tree()
139: (8)                    tree.set("xmlns", SHEET_MAIN_NS)
140: (8)                    return tree
141: (4)                @property
142: (4)                def comments(self):
143: (8)                    """
144: (8)                    Return a dictionary of comments keyed by coord
145: (8)                    """
146: (8)                    authors = self.authors.author
147: (8)                    for c in self.commentList:
148: (12)                       yield c.ref, Comment(c.content, authors[c.authorId], c.height,
c.width)
149: (4)                @classmethod
150: (4)                def from_comments(cls, comments):
151: (8)                    """
152: (8)                    Create a comment sheet from a list of comments for a particular
worksheet
153: (8)                    """
154: (8)                    authors = IndexedList()
155: (8)                    for comment in comments:
156: (12)                       comment.authorId = authors.add(comment.author)
157: (8)                    return cls(authors=AuthorList(authors), commentList=comments)
158: (4)                def write_shapes(self, vml=None):
159: (8)                    """
160: (8)                    Create the VML for comments
161: (8)                    """
162: (8)                    sw = ShapeWriter(self.comments)
163: (8)                    return sw.write(vml)
164: (4)                @property
165: (4)                def path(self):
166: (8)                    """
167: (8)                    Return path within the archive
168: (8)                    """
169: (8)                    return self._path.format(self._id)

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

File 56 - abc.py:

1: (0)              try:
2: (4)                  from abc import ABC
3: (0)              except ImportError:
4: (4)                  from abc import ABCMeta
5: (4)                  ABC = ABCMeta('ABC', (object, ), {})

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

File 57 - base.py:

1: (0)              """
2: (0)              Based on Python Cookbook 3rd Edition, 8.13
3: (0)
http://chimera.labs.oreilly.com/books/1230000000393/ch08.html#_discussiuncion_130
4: (0)              """
5: (0)              import datetime
6: (0)              import re
```

```python
from openpyxl import DEBUG
from openpyxl.utils.datetime import from_ISO8601
from .namespace import namespaced
class Descriptor:
    def __init__(self, name=None, **kw):
        self.name = name
        for k, v in kw.items():
            setattr(self, k, v)
    def __set__(self, instance, value):
        instance.__dict__[self.name] = value
class Typed(Descriptor):
    """Values must of a particular type"""
    expected_type = type(None)
    allow_none = False
    nested = False
    def __init__(self, *args, **kw):
        super().__init__(*args, **kw)
        self.__doc__ = f"Values must be of type {self.expected_type}"
    def __set__(self, instance, value):
        if not isinstance(value, self.expected_type):
            if (not self.allow_none
                or (self.allow_none and value is not None)):
                msg = f"{instance.__class__}.{self.name} should be {self.expected_type} but value is {type(value)}"
                if DEBUG:
                    msg = f"{instance.__class__}.{self.name} should be {self.expected_type} but {value} is {type(value)}"
                raise TypeError(msg)
        super().__set__(instance, value)
    def __repr__(self):
        return  self.__doc__
def _convert(expected_type, value):
    """
    Check value is of or can be converted to expected type.
    """
    if not isinstance(value, expected_type):
        try:
            value = expected_type(value)
        except:
            raise TypeError('expected ' + str(expected_type))
    return value
class Convertible(Typed):
    """Values must be convertible to a particular type"""
    def __set__(self, instance, value):
        if ((self.allow_none and value is not None)
            or not self.allow_none):
            value = _convert(self.expected_type, value)
        super().__set__(instance, value)
class Max(Convertible):
    """Values must be less than a `max` value"""
    expected_type = float
    allow_none = False
    def __init__(self, **kw):
        if 'max' not in kw and not hasattr(self, 'max'):
            raise TypeError('missing max value')
        super().__init__(**kw)
    def __set__(self, instance, value):
        if ((self.allow_none and value is not None)
            or not self.allow_none):
            value = _convert(self.expected_type, value)
            if value > self.max:
                raise ValueError('Max value is {0}'.format(self.max))
        super().__set__(instance, value)
class Min(Convertible):
    """Values must be greater than a `min` value"""
    expected_type = float
    allow_none = False
    def __init__(self, **kw):
        if 'min' not in kw and not hasattr(self, 'min'):
```

```
74: (12)                    raise TypeError('missing min value')
75: (8)                 super().__init__(**kw)
76: (4)             def __set__(self, instance, value):
77: (8)                 if ((self.allow_none and value is not None)
78: (12)                    or not self.allow_none):
79: (12)                    value = _convert(self.expected_type, value)
80: (12)                    if value < self.min:
81: (16)                        raise ValueError('Min value is {0}'.format(self.min))
82: (8)                 super().__set__(instance, value)
83: (0)         class MinMax(Min, Max):
84: (4)             """Values must be greater than `min` value and less than a `max` one"""
85: (4)             pass
86: (0)         class Set(Descriptor):
87: (4)             """Value can only be from a set of know values"""
88: (4)             def __init__(self, name=None, **kw):
89: (8)                 if not 'values' in kw:
90: (12)                    raise TypeError("missing set of values")
91: (8)                 kw['values'] = set(kw['values'])
92: (8)                 super().__init__(name, **kw)
93: (8)                 self.__doc__ = "Value must be one of {0}".format(self.values)
94: (4)             def __set__(self, instance, value):
95: (8)                 if value not in self.values:
96: (12)                    raise ValueError(self.__doc__)
97: (8)                 super().__set__(instance, value)
98: (0)         class NoneSet(Set):
99: (4)             """'none' will be treated as None"""
100: (4)            def __init__(self, name=None, **kw):
101: (8)                super().__init__(name, **kw)
102: (8)                self.values.add(None)
103: (4)            def __set__(self, instance, value):
104: (8)                if value == 'none':
105: (12)                   value = None
106: (8)                super().__set__(instance, value)
107: (0)        class Integer(Convertible):
108: (4)            expected_type = int
109: (0)        class Float(Convertible):
110: (4)            expected_type = float
111: (0)        class Bool(Convertible):
112: (4)            expected_type = bool
113: (4)            def __set__(self, instance, value):
114: (8)                if isinstance(value, str):
115: (12)                   if value in ('false', 'f', '0'):
116: (16)                       value = False
117: (8)                super().__set__(instance, value)
118: (0)        class String(Typed):
119: (4)            expected_type = str
120: (0)        class Text(String, Convertible):
121: (4)            pass
122: (0)        class ASCII(Typed):
123: (4)            expected_type = bytes
124: (0)        class Tuple(Typed):
125: (4)            expected_type = tuple
126: (0)        class Length(Descriptor):
127: (4)            def __init__(self, name=None, **kw):
128: (8)                if "length" not in kw:
129: (12)                   raise TypeError("value length must be supplied")
130: (8)                super().__init__(**kw)
131: (4)            def __set__(self, instance, value):
132: (8)                if len(value) != self.length:
133: (12)                   raise ValueError("Value must be length {0}".format(self.length))
134: (8)                super().__set__(instance, value)
135: (0)        class Default(Typed):
136: (4)            """
137: (4)            When called returns an instance of the expected type.
138: (4)            Additional default values can be passed in to the descriptor
139: (4)            """
140: (4)            def __init__(self, name=None, **kw):
141: (8)                if "defaults" not in kw:
142: (12)                   kw['defaults'] = {}
```

```
143: (8)                    super().__init__(**kw)
144: (4)                def __call__(self):
145: (8)                    return self.expected_type()
146: (0)            class Alias(Descriptor):
147: (4)                """
148: (4)                Aliases can be used when either the desired attribute name is not allowed
149: (4)                or confusing in Python (eg. "type") or a more descriptive name is desired
150: (4)                (eg. "underline" for "u")
151: (4)                """
152: (4)                def __init__(self, alias):
153: (8)                    self.alias = alias
154: (4)                def __set__(self, instance, value):
155: (8)                    setattr(instance, self.alias, value)
156: (4)                def __get__(self, instance, cls):
157: (8)                    return getattr(instance, self.alias)
158: (0)            class MatchPattern(Descriptor):
159: (4)                """Values must match a regex pattern """
160: (4)                allow_none = False
161: (4)                def __init__(self, name=None, **kw):
162: (8)                    if 'pattern' not in kw and not hasattr(self, 'pattern'):
163: (12)                       raise TypeError('missing pattern value')
164: (8)                    super().__init__(name, **kw)
165: (8)                    self.test_pattern = re.compile(self.pattern, re.VERBOSE)
166: (4)                def __set__(self, instance, value):
167: (8)                    if value is None and not self.allow_none:
168: (12)                       raise ValueError("Value must not be none")
169: (8)                    if ((self.allow_none and value is not None)
170: (12)                       or not self.allow_none):
171: (12)                       if not self.test_pattern.match(value):
172: (16)                           raise ValueError('Value does not match pattern
{0}'.format(self.pattern))
173: (8)                    super().__set__(instance, value)
174: (0)            class DateTime(Typed):
175: (4)                expected_type = datetime.datetime
176: (4)                def __set__(self, instance, value):
177: (8)                    if value is not None and isinstance(value, str):
178: (12)                       try:
179: (16)                           value = from_ISO8601(value)
180: (12)                       except ValueError:
181: (16)                           raise ValueError("Value must be ISO datetime format")
182: (8)                    super().__set__(instance, value)

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

File 58 - numbers.py:

1: (0)              from decimal import Decimal
2: (0)              NUMERIC_TYPES = (int, float, Decimal)
3: (0)              try:
4: (4)                  import numpy
5: (4)                  NUMPY = True
6: (0)              except ImportError:
7: (4)                  NUMPY = False
8: (0)              if NUMPY:
9: (4)                  NUMERIC_TYPES = NUMERIC_TYPES + (numpy.short,
10: (37)                                                 numpy.ushort,
11: (37)                                                 numpy.intc,
12: (37)                                                 numpy.uintc,
13: (37)                                                 numpy.int_,
14: (37)                                                 numpy.uint,
15: (37)                                                 numpy.longlong,
16: (37)                                                 numpy.ulonglong,
17: (37)                                                 numpy.half,
18: (37)                                                 numpy.float16,
19: (37)                                                 numpy.single,
20: (37)                                                 numpy.double,
21: (37)                                                 numpy.longdouble,
22: (37)                                                 numpy.int8,
23: (37)                                                 numpy.int16,
24: (37)                                                 numpy.int32,
25: (37)                                                 numpy.int64,
26: (37)                                                 numpy.uint8,
27: (37)                                                 numpy.uint16,
28: (37)                                                 numpy.uint32,
29: (37)                                                 numpy.uint64,
30: (37)                                                 numpy.intp,
31: (37)                                                 numpy.uintp,
32: (37)                                                 numpy.float32,
33: (37)                                                 numpy.float64,
34: (37)                                                 numpy.bool_,
35: (37)                                                 numpy.floating,
36: (37)                                                 numpy.integer)

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

File 59 - product.py:

1: (0)              """
2: (0)              math.prod equivalent for < Python 3.8
3: (0)              """
4: (0)              import functools
5: (0)              import operator
6: (0)              def product(sequence):
7: (4)                  return functools.reduce(operator.mul, sequence)
8: (0)              try:
9: (4)                  from math import prod
10: (0)             except ImportError:
11: (4)                 prod = product

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

File 60 - strings.py:

1: (0)              from datetime import datetime
2: (0)              from math import isnan, isinf
3: (0)              import sys
4: (0)              VER = sys.version_info
5: (0)              from .numbers import NUMERIC_TYPES
6: (0)              def safe_string(value):
7: (4)                  """Safely and consistently format numeric values"""
8: (4)                  if isinstance(value, NUMERIC_TYPES):
9: (8)                      if isnan(value) or isinf(value):
10: (12)                        value = ""
11: (8)                     else:
12: (12)                        value = "%.16g" % value
13: (4)                 elif value is None:
14: (8)                     value = "none"
15: (4)                 elif isinstance(value, datetime):
16: (8)                     value = value.isoformat()
17: (4)                 elif not isinstance(value, str):
18: (8)                     value = str(value)
19: (4)                 return value

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

File 61 - __init__.py:

1: (0)              from .numbers import NUMERIC_TYPES
2: (0)              from .strings import safe_string
3: (0)              import warnings
4: (0)              from functools import wraps
5: (0)              import inspect
6: (0)              class DummyCode:
7: (4)                  pass
8: (0)              string_types = (type(b''), type(u''))
9: (0)              def deprecated(reason):
10: (4)                 if isinstance(reason, string_types):
11: (8)                     def decorator(func1):
```

```
12: (12)                    if inspect.isclass(func1):
13: (16)                        fmt1 = "Call to deprecated class {name} ({reason})."
14: (12)                    else:
15: (16)                        fmt1 = "Call to deprecated function {name} ({reason})."
16: (12)                    @wraps(func1)
17: (12)                    def new_func1(*args, **kwargs):
18: (16)                        warnings.warn(
19: (20)                            fmt1.format(name=func1.__name__, reason=reason),
20: (20)                            category=DeprecationWarning,
21: (20)                            stacklevel=2
22: (16)                        )
23: (16)                        return func1(*args, **kwargs)
24: (12)                    deprecationNote = "\n\n.. note::\n    Deprecated: " + reason
25: (12)                    if new_func1.__doc__:
26: (16)                        new_func1.__doc__ += deprecationNote
27: (12)                    else:
28: (16)                        new_func1.__doc__ = deprecationNote
29: (12)                    return new_func1
30: (8)                 return decorator
31: (4)             elif inspect.isclass(reason) or inspect.isfunction(reason):
32: (8)                 raise TypeError("Reason for deprecation must be supplied")
33: (4)             else:
34: (8)                 raise TypeError(repr(type(reason)))

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

File 62 - __init__.py:

1: (0)            from .base import *
2: (0)            from .sequence import Sequence
3: (0)            class MetaStrict(type):
4: (4)                def __new__(cls, clsname, bases, methods):
5: (8)                    for k, v in methods.items():
6: (12)                       if isinstance(v, Descriptor):
7: (16)                           v.name = k
8: (8)                    return type.__new__(cls, clsname, bases, methods)
9: (0)            class Strict(metaclass=MetaStrict):
10: (4)               pass
11: (0)           class MetaSerialisable(type):
12: (4)               def __new__(cls, clsname, bases, methods):
13: (8)                   attrs = []
14: (8)                   nested = []
15: (8)                   elements = []
16: (8)                   namespaced = []
17: (8)                   for k, v in methods.items():
18: (12)                      if isinstance(v, Descriptor):
19: (16)                          ns= getattr(v, 'namespace', None)
20: (16)                          if ns:
21: (20)                              namespaced.append((k, "{%s}%s" % (ns, k)))
22: (16)                          if getattr(v, 'nested', False):
23: (20)                              nested.append(k)
24: (20)                              elements.append(k)
25: (16)                          elif isinstance(v, Sequence):
26: (20)                              elements.append(k)
27: (16)                          elif isinstance(v, Typed):
28: (20)                              if hasattr(v.expected_type, 'to_tree'):
29: (24)                                  elements.append(k)
30: (20)                              elif isinstance(v.expected_type, tuple):
31: (24)                                  if any((hasattr(el, "to_tree") for el in
v.expected_type)):
32: (28)                                      continue
33: (20)                              else:
34: (24)                                  attrs.append(k)
35: (16)                          else:
36: (20)                              if not isinstance(v, Alias):
37: (24)                                  attrs.append(k)
38: (8)                   if methods.get('__attrs__') is None:
39: (12)                      methods['__attrs__'] = tuple(attrs)
40: (8)                   methods['__namespaced__'] = tuple(namespaced)
```

```
41: (8)                     if methods.get('__nested__') is None:
42: (12)                        methods['__nested__'] = tuple(sorted(nested))
43: (8)                     if methods.get('__elements__') is None:
44: (12)                        methods['__elements__'] = tuple(sorted(elements))
45: (8)                     return MetaStrict.__new__(cls, clsname, bases, methods)

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

File 63 - singleton.py:

1: (0)              import weakref
2: (0)              class Singleton(type):
3: (4)                  """
4: (4)                  Singleton metaclass
5: (4)                  Based on Python Cookbook 3rd Edition Recipe 9.13
6: (4)                  Only one instance of a class can exist. Does not work with __slots__
7: (4)                  """
8: (4)                  def __init__(self, *args, **kw):
9: (8)                      super().__init__(*args, **kw)
10: (8)                     self.__instance = None
11: (4)                 def __call__(self, *args, **kw):
12: (8)                     if self.__instance is None:
13: (12)                        self.__instance = super().__call__(*args, **kw)
14: (8)                     return self.__instance
15: (0)             class Cached(type):
16: (4)                 """
17: (4)                 Caching metaclass
18: (4)                 Child classes will only create new instances of themselves if
19: (4)                 one doesn't already exist. Does not work with __slots__
20: (4)                 """
21: (4)                 def __init__(self, *args, **kw):
22: (8)                     super().__init__(*args, **kw)
23: (8)                     self.__cache = weakref.WeakValueDictionary()
24: (4)                 def __call__(self, *args):
25: (8)                     if args in self.__cache:
26: (12)                        return self.__cache[args]
27: (8)                     obj = super().__call__(*args)
28: (8)                     self.__cache[args] = obj
29: (8)                     return obj

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

File 64 - container.py:

1: (0)              """
2: (0)              Utility list for top level containers that contain one type of element
3: (0)              Provides the necessary API to read and write XML
4: (0)              """
5: (0)              from openpyxl.xml.functions import Element
6: (0)              class ElementList(list):
7: (4)                  @property
8: (4)                  def tagname(self):
9: (8)                      raise NotImplementedError
10: (4)                 @property
11: (4)                 def expected_type(self):
12: (8)                     raise NotImplementedError
13: (4)                 @classmethod
14: (4)                 def from_tree(cls, tree):
15: (8)                     l = [cls.expected_type.from_tree(el) for el in tree]
16: (8)                     return cls(l)
17: (4)                 def to_tree(self):
18: (8)                     container = Element(self.tagname)
19: (8)                     for el in self:
20: (12)                        container.append(el.to_tree())
21: (8)                     return container
22: (4)                 def append(self, value):
23: (8)                     if not isinstance(value, self.expected_type):
24: (12)                        raise TypeError(f"Value must of type {self.expected_type}
{type(value)} provided")
```

```
25: (8)                         super().append(value)

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

File 65 - xdr.py:

1: (0)                  """
2: (0)                  Spreadsheet Drawing has some copies of Drawing ML elements
3: (0)                  """
4: (0)                  from .geometry import Point2D, PositiveSize2D, Transform2D
5: (0)                  class XDRPoint2D(Point2D):
6: (4)                      namespace = None
7: (4)                      x = Point2D.x
8: (4)                      y = Point2D.y
9: (0)                  class XDRPositiveSize2D(PositiveSize2D):
10: (4)                     namespace = None
11: (4)                     cx = PositiveSize2D.cx
12: (4)                     cy = PositiveSize2D.cy
13: (0)                 class XDRTransform2D(Transform2D):
14: (4)                     namespace = None
15: (4)                     rot = Transform2D.rot
16: (4)                     flipH = Transform2D.flipH
17: (4)                     flipV = Transform2D.flipV
18: (4)                     off = Transform2D.off
19: (4)                     ext = Transform2D.ext
20: (4)                     chOff = Transform2D.chOff
21: (4)                     chExt = Transform2D.chExt

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

File 66 - fill.py:

1: (0)                  from openpyxl.descriptors.serialisable import Serialisable
2: (0)                  from openpyxl.descriptors import (
3: (4)                      Alias,
4: (4)                      Bool,
5: (4)                      Integer,
6: (4)                      Set,
7: (4)                      NoneSet,
8: (4)                      Typed,
9: (4)                      MinMax,
10: (0)                 )
11: (0)                 from openpyxl.descriptors.excel import (
12: (4)                     Relation,
13: (4)                     Percentage,
14: (0)                 )
15: (0)                 from openpyxl.descriptors.nested import NestedNoneSet, NestedValue
16: (0)                 from openpyxl.descriptors.sequence import NestedSequence
17: (0)                 from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
18: (0)                 from openpyxl.xml.constants import DRAWING_NS
19: (0)                 from .colors import (
20: (4)                     ColorChoice,
21: (4)                     HSLColor,
22: (4)                     SystemColor,
23: (4)                     SchemeColor,
24: (4)                     PRESET_COLORS,
25: (4)                     RGBPercent,
26: (0)                 )
27: (0)                 from .effect import (
28: (4)                     AlphaBiLevelEffect,
29: (4)                     AlphaCeilingEffect,
30: (4)                     AlphaFloorEffect,
31: (4)                     AlphaInverseEffect,
32: (4)                     AlphaModulateEffect,
33: (4)                     AlphaModulateFixedEffect,
34: (4)                     AlphaReplaceEffect,
35: (4)                     BiLevelEffect,
36: (4)                     BlurEffect,
37: (4)                     ColorChangeEffect,
38: (4)                 ColorReplaceEffect,
39: (4)                 DuotoneEffect,
40: (4)                 FillOverlayEffect,
41: (4)                 GrayscaleEffect,
42: (4)                 HSLEffect,
43: (4)                 LuminanceEffect,
44: (4)                 TintEffect,
45: (0)             )
46: (0)             """
47: (0)             Fill elements from drawing main schema
48: (0)             """
49: (0)             class PatternFillProperties(Serialisable):
50: (4)                 tagname = "pattFill"
51: (4)                 namespace = DRAWING_NS
52: (4)                 prst = NoneSet(values=(['pct5', 'pct10', 'pct20', 'pct25', 'pct30',
53: (28)                                        'pct40', 'pct50', 'pct60', 'pct70', 'pct75',
'pct80', 'pct90', 'horz',
54: (28)                                        'vert', 'ltHorz', 'ltVert', 'dkHorz', 'dkVert',
'narHorz', 'narVert',
55: (28)                                        'dashHorz', 'dashVert', 'cross', 'dnDiag',
'upDiag', 'ltDnDiag',
56: (28)                                        'ltUpDiag', 'dkDnDiag', 'dkUpDiag', 'wdDnDiag',
'wdUpDiag', 'dashDnDiag',
57: (28)                                        'dashUpDiag', 'diagCross', 'smCheck', 'lgCheck',
'smGrid', 'lgGrid',
58: (28)                                        'dotGrid', 'smConfetti', 'lgConfetti',
'horzBrick', 'diagBrick',
59: (28)                                        'solidDmnd', 'openDmnd', 'dotDmnd', 'plaid',
'sphere', 'weave', 'divot',
60: (28)                                        'shingle', 'wave', 'trellis', 'zigZag']))
61: (4)                 preset = Alias("prst")
62: (4)                 fgClr = Typed(expected_type=ColorChoice, allow_none=True)
63: (4)                 foreground = Alias("fgClr")
64: (4)                 bgClr = Typed(expected_type=ColorChoice, allow_none=True)
65: (4)                 background = Alias("bgClr")
66: (4)                 __elements__ = ("fgClr", "bgClr")
67: (4)                 def __init__(self,
68: (17)                             prst=None,
69: (17)                             fgClr=None,
70: (17)                             bgClr=None,
71: (16)                            ):
72: (8)                     self.prst = prst
73: (8)                     self.fgClr = fgClr
74: (8)                     self.bgClr = bgClr
75: (0)             class RelativeRect(Serialisable):
76: (4)                 tagname = "rect"
77: (4)                 namespace = DRAWING_NS
78: (4)                 l = Percentage(allow_none=True)
79: (4)                 left = Alias('l')
80: (4)                 t = Percentage(allow_none=True)
81: (4)                 top = Alias('t')
82: (4)                 r = Percentage(allow_none=True)
83: (4)                 right = Alias('r')
84: (4)                 b = Percentage(allow_none=True)
85: (4)                 bottom = Alias('b')
86: (4)                 def __init__(self,
87: (17)                             l=None,
88: (17)                             t=None,
89: (17)                             r=None,
90: (17)                             b=None,
91: (16)                            ):
92: (8)                     self.l = l
93: (8)                     self.t = t
94: (8)                     self.r = r
95: (8)                     self.b = b
96: (0)             class StretchInfoProperties(Serialisable):
97: (4)                 tagname = "stretch"
98: (4)                 namespace = DRAWING_NS
99: (4)                 fillRect = Typed(expected_type=RelativeRect, allow_none=True)
```

```
100: (4)                def __init__(self,
101: (17)                            fillRect=RelativeRect(),
102: (16)                           ):
103: (8)                    self.fillRect = fillRect
104: (0)            class GradientStop(Serialisable):
105: (4)                tagname = "gs"
106: (4)                namespace = DRAWING_NS
107: (4)                pos = MinMax(min=0, max=100000, allow_none=True)
108: (4)                scrgbClr = Typed(expected_type=RGBPercent, allow_none=True)
109: (4)                RGBPercent = Alias('scrgbClr')
110: (4)                srgbClr = NestedValue(expected_type=str, allow_none=True) # needs pattern
and can have transform
111: (4)                RGB = Alias('srgbClr')
112: (4)                hslClr = Typed(expected_type=HSLColor, allow_none=True)
113: (4)                sysClr = Typed(expected_type=SystemColor, allow_none=True)
114: (4)                schemeClr = Typed(expected_type=SchemeColor, allow_none=True)
115: (4)                prstClr = NestedNoneSet(values=PRESET_COLORS)
116: (4)                __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
117: (4)                def __init__(self,
118: (17)                            pos=None,
119: (17)                            scrgbClr=None,
120: (17)                            srgbClr=None,
121: (17)                            hslClr=None,
122: (17)                            sysClr=None,
123: (17)                            schemeClr=None,
124: (17)                            prstClr=None,
125: (16)                           ):
126: (8)                    if pos is None:
127: (12)                       pos = 0
128: (8)                    self.pos = pos
129: (8)                    self.scrgbClr = scrgbClr
130: (8)                    self.srgbClr = srgbClr
131: (8)                    self.hslClr = hslClr
132: (8)                    self.sysClr = sysClr
133: (8)                    self.schemeClr = schemeClr
134: (8)                    self.prstClr = prstClr
135: (0)            class LinearShadeProperties(Serialisable):
136: (4)                tagname = "lin"
137: (4)                namespace = DRAWING_NS
138: (4)                ang = Integer()
139: (4)                scaled = Bool(allow_none=True)
140: (4)                def __init__(self,
141: (17)                            ang=None,
142: (17)                            scaled=None,
143: (16)                           ):
144: (8)                    self.ang = ang
145: (8)                    self.scaled = scaled
146: (0)            class PathShadeProperties(Serialisable):
147: (4)                tagname = "path"
148: (4)                namespace = DRAWING_NS
149: (4)                path = Set(values=(['shape', 'circle', 'rect']))
150: (4)                fillToRect = Typed(expected_type=RelativeRect, allow_none=True)
151: (4)                def __init__(self,
152: (17)                            path=None,
153: (17)                            fillToRect=None,
154: (16)                           ):
155: (8)                    self.path = path
156: (8)                    self.fillToRect = fillToRect
157: (0)            class GradientFillProperties(Serialisable):
158: (4)                tagname = "gradFill"
159: (4)                namespace = DRAWING_NS
160: (4)                flip = NoneSet(values=(['x', 'y', 'xy']))
161: (4)                rotWithShape = Bool(allow_none=True)
162: (4)                gsLst = NestedSequence(expected_type=GradientStop, count=False)
163: (4)                stop_list = Alias("gsLst")
164: (4)                lin = Typed(expected_type=LinearShadeProperties, allow_none=True)
165: (4)                linear = Alias("lin")
166: (4)                path = Typed(expected_type=PathShadeProperties, allow_none=True)
```

```
(4)     tileRect = Typed(expected_type=RelativeRect, allow_none=True)
(4)     __elements__ = ('gsLst', 'lin', 'path', 'tileRect')
(4)     def __init__(self,
(17)                 flip=None,
(17)                 rotWithShape=None,
(17)                 gsLst=(),
(17)                 lin=None,
(17)                 path=None,
(17)                 tileRect=None,
(16)                ):
(8)         self.flip = flip
(8)         self.rotWithShape = rotWithShape
(8)         self.gsLst = gsLst
(8)         self.lin = lin
(8)         self.path = path
(8)         self.tileRect = tileRect
(0) class SolidColorFillProperties(Serialisable):
(4)     tagname = "solidFill"
(4)     scrgbClr = Typed(expected_type=RGBPercent, allow_none=True)
(4)     RGBPercent = Alias('scrgbClr')
(4)     srgbClr = NestedValue(expected_type=str, allow_none=True) # needs pattern and can have transform
(4)     RGB = Alias('srgbClr')
(4)     hslClr = Typed(expected_type=HSLColor, allow_none=True)
(4)     sysClr = Typed(expected_type=SystemColor, allow_none=True)
(4)     schemeClr = Typed(expected_type=SchemeColor, allow_none=True)
(4)     prstClr = NestedNoneSet(values=PRESET_COLORS)
(4)     __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr', 'prstClr')
(4)     def __init__(self,
(17)                 scrgbClr=None,
(17)                 srgbClr=None,
(17)                 hslClr=None,
(17)                 sysClr=None,
(17)                 schemeClr=None,
(17)                 prstClr=None,
(16)                ):
(8)         self.scrgbClr = scrgbClr
(8)         self.srgbClr = srgbClr
(8)         self.hslClr = hslClr
(8)         self.sysClr = sysClr
(8)         self.schemeClr = schemeClr
(8)         self.prstClr = prstClr
(0) class Blip(Serialisable):
(4)     tagname = "blip"
(4)     namespace = DRAWING_NS
(4)     cstate = NoneSet(values=(['email', 'screen', 'print', 'hqprint']))
(4)     embed = Relation() # rId
(4)     link = Relation() # hyperlink
(4)     noGrp = Bool(allow_none=True)
(4)     noSelect = Bool(allow_none=True)
(4)     noRot = Bool(allow_none=True)
(4)     noChangeAspect = Bool(allow_none=True)
(4)     noMove = Bool(allow_none=True)
(4)     noResize = Bool(allow_none=True)
(4)     noEditPoints = Bool(allow_none=True)
(4)     noAdjustHandles = Bool(allow_none=True)
(4)     noChangeArrowheads = Bool(allow_none=True)
(4)     noChangeShapeType = Bool(allow_none=True)
(4)     extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)     alphaBiLevel = Typed(expected_type=AlphaBiLevelEffect, allow_none=True)
(4)     alphaCeiling = Typed(expected_type=AlphaCeilingEffect, allow_none=True)
(4)     alphaFloor = Typed(expected_type=AlphaFloorEffect, allow_none=True)
(4)     alphaInv = Typed(expected_type=AlphaInverseEffect, allow_none=True)
(4)     alphaMod = Typed(expected_type=AlphaModulateEffect, allow_none=True)
(4)     alphaModFix = Typed(expected_type=AlphaModulateFixedEffect, allow_none=True)
(4)     alphaRepl = Typed(expected_type=AlphaReplaceEffect, allow_none=True)
(4)     biLevel = Typed(expected_type=BiLevelEffect, allow_none=True)
```

```
(4)                 blur = Typed(expected_type=BlurEffect, allow_none=True)
(4)                 clrChange = Typed(expected_type=ColorChangeEffect, allow_none=True)
(4)                 clrRepl = Typed(expected_type=ColorReplaceEffect, allow_none=True)
(4)                 duotone = Typed(expected_type=DuotoneEffect, allow_none=True)
(4)                 fillOverlay = Typed(expected_type=FillOverlayEffect, allow_none=True)
(4)                 grayscl = Typed(expected_type=GrayscaleEffect, allow_none=True)
(4)                 hsl = Typed(expected_type=HSLEffect, allow_none=True)
(4)                 lum = Typed(expected_type=LuminanceEffect, allow_none=True)
(4)                 tint = Typed(expected_type=TintEffect, allow_none=True)
(4)                 __elements__ = ('alphaBiLevel', 'alphaCeiling', 'alphaFloor', 'alphaInv',
(20)                                'alphaMod', 'alphaModFix', 'alphaRepl', 'biLevel', 'blur',
'clrChange',
(20)                                'clrRepl', 'duotone', 'fillOverlay', 'grayscl', 'hsl',
'lum', 'tint')
(4)                 def __init__(self,
(17)                             cstate=None,
(17)                             embed=None,
(17)                             link=None,
(17)                             noGrp=None,
(17)                             noSelect=None,
(17)                             noRot=None,
(17)                             noChangeAspect=None,
(17)                             noMove=None,
(17)                             noResize=None,
(17)                             noEditPoints=None,
(17)                             noAdjustHandles=None,
(17)                             noChangeArrowheads=None,
(17)                             noChangeShapeType=None,
(17)                             extLst=None,
(17)                             alphaBiLevel=None,
(17)                             alphaCeiling=None,
(17)                             alphaFloor=None,
(17)                             alphaInv=None,
(17)                             alphaMod=None,
(17)                             alphaModFix=None,
(17)                             alphaRepl=None,
(17)                             biLevel=None,
(17)                             blur=None,
(17)                             clrChange=None,
(17)                             clrRepl=None,
(17)                             duotone=None,
(17)                             fillOverlay=None,
(17)                             grayscl=None,
(17)                             hsl=None,
(17)                             lum=None,
(17)                             tint=None,
(16)                            ):
(8)                     self.cstate = cstate
(8)                     self.embed = embed
(8)                     self.link = link
(8)                     self.noGrp = noGrp
(8)                     self.noSelect = noSelect
(8)                     self.noRot = noRot
(8)                     self.noChangeAspect = noChangeAspect
(8)                     self.noMove = noMove
(8)                     self.noResize = noResize
(8)                     self.noEditPoints = noEditPoints
(8)                     self.noAdjustHandles = noAdjustHandles
(8)                     self.noChangeArrowheads = noChangeArrowheads
(8)                     self.noChangeShapeType = noChangeShapeType
(8)                     self.extLst = extLst
(8)                     self.alphaBiLevel = alphaBiLevel
(8)                     self.alphaCeiling = alphaCeiling
(8)                     self.alphaFloor = alphaFloor
(8)                     self.alphaInv = alphaInv
(8)                     self.alphaMod = alphaMod
(8)                     self.alphaModFix = alphaModFix
(8)                     self.alphaRepl = alphaRepl
(8)                     self.biLevel = biLevel
```

```
(8)                     self.blur = blur
(8)                     self.clrChange = clrChange
(8)                     self.clrRepl = clrRepl
(8)                     self.duotone = duotone
(8)                     self.fillOverlay = fillOverlay
(8)                     self.grayscl = grayscl
(8)                     self.hsl = hsl
(8)                     self.lum = lum
(8)                     self.tint = tint
(0)             class TileInfoProperties(Serialisable):
(4)                 tx = Integer(allow_none=True)
(4)                 ty = Integer(allow_none=True)
(4)                 sx = Integer(allow_none=True)
(4)                 sy = Integer(allow_none=True)
(4)                 flip = NoneSet(values=(['x', 'y', 'xy']))
(4)                 algn = Set(values=(['tl', 't', 'tr', 'l', 'ctr', 'r', 'bl', 'b', 'br']))
(4)                 def __init__(self,
(17)                             tx=None,
(17)                             ty=None,
(17)                             sx=None,
(17)                             sy=None,
(17)                             flip=None,
(17)                             algn=None,
(16)                            ):
(8)                     self.tx = tx
(8)                     self.ty = ty
(8)                     self.sx = sx
(8)                     self.sy = sy
(8)                     self.flip = flip
(8)                     self.algn = algn
(0)             class BlipFillProperties(Serialisable):
(4)                 tagname = "blipFill"
(4)                 dpi = Integer(allow_none=True)
(4)                 rotWithShape = Bool(allow_none=True)
(4)                 blip = Typed(expected_type=Blip, allow_none=True)
(4)                 srcRect = Typed(expected_type=RelativeRect, allow_none=True)
(4)                 tile = Typed(expected_type=TileInfoProperties, allow_none=True)
(4)                 stretch = Typed(expected_type=StretchInfoProperties, allow_none=True)
(4)                 __elements__ = ("blip", "srcRect", "tile", "stretch")
(4)                 def __init__(self,
(17)                             dpi=None,
(17)                             rotWithShape=None,
(17)                             blip=None,
(17)                             tile=None,
(17)                             stretch=StretchInfoProperties(),
(17)                             srcRect=None,
(16)                            ):
(8)                     self.dpi = dpi
(8)                     self.rotWithShape = rotWithShape
(8)                     self.blip = blip
(8)                     self.tile = tile
(8)                     self.stretch = stretch
(8)                     self.srcRect = srcRect

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

File 67 - line.py:

(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 Typed,
(4)                 Integer,
(4)                 MinMax,
(4)                 NoneSet,
(4)                 Alias,
(4)                 Sequence
(0)             )
(0)             from openpyxl.descriptors.nested import (
(4)                 NestedInteger,
(4)                 NestedNoneSet,
(4)                 EmptyTag,
(0)             )
(0)             from openpyxl.xml.constants import DRAWING_NS
(0)             from .colors import ColorChoiceDescriptor
(0)             from .fill import GradientFillProperties, PatternFillProperties
(0)             from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
(0)             """
(0)             Line elements from drawing main schema
(0)             """
(0)             class LineEndProperties(Serialisable):
(4)                 tagname = "end"
(4)                 namespace = DRAWING_NS
(4)                 type = NoneSet(values=(['none', 'triangle', 'stealth', 'diamond', 'oval', 'arrow']))
(4)                 w = NoneSet(values=(['sm', 'med', 'lg']))
(4)                 len = NoneSet(values=(['sm', 'med', 'lg']))
(4)                 def __init__(self,
(17)                             type=None,
(17)                             w=None,
(17)                             len=None,
(16)                            ):
(8)                     self.type = type
(8)                     self.w = w
(8)                     self.len = len
(0)             class DashStop(Serialisable):
(4)                 tagname = "ds"
(4)                 namespace = DRAWING_NS
(4)                 d = Integer()
(4)                 length = Alias('d')
(4)                 sp = Integer()
(4)                 space = Alias('sp')
(4)                 def __init__(self,
(17)                             d=0,
(17)                             sp=0,
(16)                            ):
(8)                     self.d = d
(8)                     self.sp = sp
(0)             class DashStopList(Serialisable):
(4)                 ds = Sequence(expected_type=DashStop, allow_none=True)
(4)                 def __init__(self,
(17)                             ds=None,
(16)                            ):
(8)                     self.ds = ds
(0)             class LineProperties(Serialisable):
(4)                 tagname = "ln"
(4)                 namespace = DRAWING_NS
(4)                 w = MinMax(min=0, max=20116800, allow_none=True) # EMU
(4)                 width = Alias('w')
(4)                 cap = NoneSet(values=(['rnd', 'sq', 'flat']))
(4)                 cmpd = NoneSet(values=(['sng', 'dbl', 'thickThin', 'thinThick', 'tri']))
(4)                 algn = NoneSet(values=(['ctr', 'in']))
(4)                 noFill = EmptyTag()
(4)                 solidFill = ColorChoiceDescriptor()
(4)                 gradFill = Typed(expected_type=GradientFillProperties, allow_none=True)
(4)                 pattFill = Typed(expected_type=PatternFillProperties, allow_none=True)
(4)                 prstDash = NestedNoneSet(values=(['solid', 'dot', 'dash', 'lgDash', 'dashDot',
(23)                                  'lgDashDot', 'lgDashDotDot', 'sysDash', 'sysDot', 'sysDashDot',
(23)                                  'sysDashDotDot']), namespace=namespace)
(4)                 dashStyle = Alias('prstDash')
(4)                 custDash = Typed(expected_type=DashStop, allow_none=True)
(4)                 round = EmptyTag()
(4)                 bevel = EmptyTag()
(4)                 miter = NestedInteger(allow_none=True, attribute="lim")
(4)                 headEnd = Typed(expected_type=LineEndProperties, allow_none=True)
(4)                 tailEnd = Typed(expected_type=LineEndProperties, allow_none=True)
(4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
```

```
78: (4)                 __elements__ = ('noFill', 'solidFill', 'gradFill', 'pattFill',
79: (20)                                'prstDash', 'custDash', 'round', 'bevel', 'miter',
'headEnd', 'tailEnd')
80: (4)                 def __init__(self,
81: (17)                             w=None,
82: (17)                             cap=None,
83: (17)                             cmpd=None,
84: (17)                             algn=None,
85: (17)                             noFill=None,
86: (17)                             solidFill=None,
87: (17)                             gradFill=None,
88: (17)                             pattFill=None,
89: (17)                             prstDash=None,
90: (17)                             custDash=None,
91: (17)                             round=None,
92: (17)                             bevel=None,
93: (17)                             miter=None,
94: (17)                             headEnd=None,
95: (17)                             tailEnd=None,
96: (17)                             extLst=None,
97: (16)                            ):
98: (8)                     self.w = w
99: (8)                     self.cap = cap
100: (8)                    self.cmpd = cmpd
101: (8)                    self.algn = algn
102: (8)                    self.noFill = noFill
103: (8)                    self.solidFill = solidFill
104: (8)                    self.gradFill = gradFill
105: (8)                    self.pattFill = pattFill
106: (8)                    if prstDash is None:
107: (12)                       prstDash = "solid"
108: (8)                    self.prstDash = prstDash
109: (8)                    self.custDash = custDash
110: (8)                    self.round = round
111: (8)                    self.bevel = bevel
112: (8)                    self.miter = miter
113: (8)                    self.headEnd = headEnd
114: (8)                    self.tailEnd = tailEnd

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

File 68 - text.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Alias,
4: (4)                  Typed,
5: (4)                  Set,
6: (4)                  NoneSet,
7: (4)                  Sequence,
8: (4)                  String,
9: (4)                  Bool,
10: (4)                 MinMax,
11: (4)                 Integer
12: (0)             )
13: (0)             from openpyxl.descriptors.excel import (
14: (4)                 HexBinary,
15: (4)                 Coordinate,
16: (4)                 Relation,
17: (0)             )
18: (0)             from openpyxl.descriptors.nested import (
19: (4)                 NestedInteger,
20: (4)                 NestedText,
21: (4)                 NestedValue,
22: (4)                 EmptyTag
23: (0)             )
24: (0)             from openpyxl.xml.constants import DRAWING_NS
25: (0)             from .colors import ColorChoiceDescriptor
26: (0)             from .effect import (
(4)                 EffectList,
(4)                 EffectContainer,
(0)             )
(0)             from .fill import(
(4)                 GradientFillProperties,
(4)                 BlipFillProperties,
(4)                 PatternFillProperties,
(4)                 Blip
(0)             )
(0)             from .geometry import (
(4)                 LineProperties,
(4)                 Color,
(4)                 Scene3D
(0)             )
(0)             from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
(0)             from openpyxl.descriptors.nested import NestedBool
(0)             class EmbeddedWAVAudioFile(Serialisable):
(4)                 name = String(allow_none=True)
(4)                 def __init__(self,
(17)                             name=None,
(16)                            ):
(8)                     self.name = name
(0)             class Hyperlink(Serialisable):
(4)                 tagname = "hlinkClick"
(4)                 namespace = DRAWING_NS
(4)                 invalidUrl = String(allow_none=True)
(4)                 action = String(allow_none=True)
(4)                 tgtFrame = String(allow_none=True)
(4)                 tooltip = String(allow_none=True)
(4)                 history = Bool(allow_none=True)
(4)                 highlightClick = Bool(allow_none=True)
(4)                 endSnd = Bool(allow_none=True)
(4)                 snd = Typed(expected_type=EmbeddedWAVAudioFile, allow_none=True)
(4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)                 id = Relation(allow_none=True)
(4)                 __elements__ = ('snd',)
(4)                 def __init__(self,
(17)                             invalidUrl=None,
(17)                             action=None,
(17)                             tgtFrame=None,
(17)                             tooltip=None,
(17)                             history=None,
(17)                             highlightClick=None,
(17)                             endSnd=None,
(17)                             snd=None,
(17)                             extLst=None,
(17)                             id=None,
(16)                            ):
(8)                     self.invalidUrl = invalidUrl
(8)                     self.action = action
(8)                     self.tgtFrame = tgtFrame
(8)                     self.tooltip = tooltip
(8)                     self.history = history
(8)                     self.highlightClick = highlightClick
(8)                     self.endSnd = endSnd
(8)                     self.snd = snd
(8)                     self.id = id
(0)             class Font(Serialisable):
(4)                 tagname = "latin"
(4)                 namespace = DRAWING_NS
(4)                 typeface = String()
(4)                 panose = HexBinary(allow_none=True)
(4)                 pitchFamily = MinMax(min=0, max=52, allow_none=True)
(4)                 charset = Integer(allow_none=True)
(4)                 def __init__(self,
(17)                             typeface=None,
(17)                             panose=None,
(17)                             pitchFamily=None,
(17)                             charset=None,
96: (16)                            ):
97: (8)                     self.typeface = typeface
98: (8)                     self.panose = panose
99: (8)                     self.pitchFamily = pitchFamily
100: (8)                    self.charset = charset
101: (0)            class CharacterProperties(Serialisable):
102: (4)                tagname = "defRPr"
103: (4)                namespace = DRAWING_NS
104: (4)                kumimoji = Bool(allow_none=True)
105: (4)                lang = String(allow_none=True)
106: (4)                altLang = String(allow_none=True)
107: (4)                sz = MinMax(allow_none=True, min=100, max=400000) # 100ths of a point
108: (4)                b = Bool(allow_none=True)
109: (4)                i = Bool(allow_none=True)
110: (4)                u = NoneSet(values=(['words', 'sng', 'dbl', 'heavy', 'dotted',
111: (25)                                    'dottedHeavy', 'dash', 'dashHeavy', 'dashLong',
'dashLongHeavy',
112: (25)                                    'dotDash', 'dotDashHeavy', 'dotDotDash',
'dotDotDashHeavy', 'wavy',
113: (25)                                    'wavyHeavy', 'wavyDbl']))
114: (4)                strike = NoneSet(values=(['noStrike', 'sngStrike', 'dblStrike']))
115: (4)                kern = Integer(allow_none=True)
116: (4)                cap = NoneSet(values=(['small', 'all']))
117: (4)                spc = Integer(allow_none=True)
118: (4)                normalizeH = Bool(allow_none=True)
119: (4)                baseline = Integer(allow_none=True)
120: (4)                noProof = Bool(allow_none=True)
121: (4)                dirty = Bool(allow_none=True)
122: (4)                err = Bool(allow_none=True)
123: (4)                smtClean = Bool(allow_none=True)
124: (4)                smtId = Integer(allow_none=True)
125: (4)                bmk = String(allow_none=True)
126: (4)                ln = Typed(expected_type=LineProperties, allow_none=True)
127: (4)                highlight = Typed(expected_type=Color, allow_none=True)
128: (4)                latin = Typed(expected_type=Font, allow_none=True)
129: (4)                ea = Typed(expected_type=Font, allow_none=True)
130: (4)                cs = Typed(expected_type=Font, allow_none=True)
131: (4)                sym = Typed(expected_type=Font, allow_none=True)
132: (4)                hlinkClick = Typed(expected_type=Hyperlink, allow_none=True)
133: (4)                hlinkMouseOver = Typed(expected_type=Hyperlink, allow_none=True)
134: (4)                rtl = NestedBool(allow_none=True)
135: (4)                extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
136: (4)                noFill = EmptyTag(namespace=DRAWING_NS)
137: (4)                solidFill = ColorChoiceDescriptor()
138: (4)                gradFill = Typed(expected_type=GradientFillProperties, allow_none=True)
139: (4)                blipFill = Typed(expected_type=BlipFillProperties, allow_none=True)
140: (4)                pattFill = Typed(expected_type=PatternFillProperties, allow_none=True)
141: (4)                grpFill = EmptyTag(namespace=DRAWING_NS)
142: (4)                effectLst = Typed(expected_type=EffectList, allow_none=True)
143: (4)                effectDag = Typed(expected_type=EffectContainer, allow_none=True)
144: (4)                uLnTx = EmptyTag()
145: (4)                uLn = Typed(expected_type=LineProperties, allow_none=True)
146: (4)                uFillTx = EmptyTag()
147: (4)                uFill = EmptyTag()
148: (4)                __elements__ = ('ln', 'noFill', 'solidFill', 'gradFill', 'blipFill',
149: (20)                               'pattFill', 'grpFill', 'effectLst', 'effectDag',
'highlight','uLnTx',
150: (20)                               'uLn', 'uFillTx', 'uFill', 'latin', 'ea', 'cs', 'sym',
'hlinkClick',
151: (20)                               'hlinkMouseOver', 'rtl', )
152: (4)                def __init__(self,
153: (17)                            kumimoji=None,
154: (17)                            lang=None,
155: (17)                            altLang=None,
156: (17)                            sz=None,
157: (17)                            b=None,
158: (17)                            i=None,
159: (17)                            u=None,
160: (17)                            strike=None,
(17)                                kern=None,
(17)                                cap=None,
(17)                                spc=None,
(17)                                normalizeH=None,
(17)                                baseline=None,
(17)                                noProof=None,
(17)                                dirty=None,
(17)                                err=None,
(17)                                smtClean=None,
(17)                                smtId=None,
(17)                                bmk=None,
(17)                                ln=None,
(17)                                highlight=None,
(17)                                latin=None,
(17)                                ea=None,
(17)                                cs=None,
(17)                                sym=None,
(17)                                hlinkClick=None,
(17)                                hlinkMouseOver=None,
(17)                                rtl=None,
(17)                                extLst=None,
(17)                                noFill=None,
(17)                                solidFill=None,
(17)                                gradFill=None,
(17)                                blipFill=None,
(17)                                pattFill=None,
(17)                                grpFill=None,
(17)                                effectLst=None,
(17)                                effectDag=None,
(17)                                uLnTx=None,
(17)                                uLn=None,
(17)                                uFillTx=None,
(17)                                uFill=None,
(16)                               ):
(8)                         self.kumimoji = kumimoji
(8)                         self.lang = lang
(8)                         self.altLang = altLang
(8)                         self.sz = sz
(8)                         self.b = b
(8)                         self.i = i
(8)                         self.u = u
(8)                         self.strike = strike
(8)                         self.kern = kern
(8)                         self.cap = cap
(8)                         self.spc = spc
(8)                         self.normalizeH = normalizeH
(8)                         self.baseline = baseline
(8)                         self.noProof = noProof
(8)                         self.dirty = dirty
(8)                         self.err = err
(8)                         self.smtClean = smtClean
(8)                         self.smtId = smtId
(8)                         self.bmk = bmk
(8)                         self.ln = ln
(8)                         self.highlight = highlight
(8)                         self.latin = latin
(8)                         self.ea = ea
(8)                         self.cs = cs
(8)                         self.sym = sym
(8)                         self.hlinkClick = hlinkClick
(8)                         self.hlinkMouseOver = hlinkMouseOver
(8)                         self.rtl = rtl
(8)                         self.noFill = noFill
(8)                         self.solidFill = solidFill
(8)                         self.gradFill = gradFill
(8)                         self.blipFill = blipFill
(8)                         self.pattFill = pattFill
(8)                         self.grpFill = grpFill
(8)                         self.effectLst = effectLst
```

```
230: (8)                    self.effectDag = effectDag
231: (8)                    self.uLnTx = uLnTx
232: (8)                    self.uLn = uLn
233: (8)                    self.uFillTx = uFillTx
234: (8)                    self.uFill = uFill
235: (0)            class TabStop(Serialisable):
236: (4)                pos = Typed(expected_type=Coordinate, allow_none=True)
237: (4)                algn = Typed(expected_type=Set(values=(['l', 'ctr', 'r', 'dec'])))
238: (4)                def __init__(self,
239: (17)                            pos=None,
240: (17)                            algn=None,
241: (16)                           ):
242: (8)                    self.pos = pos
243: (8)                    self.algn = algn
244: (0)            class TabStopList(Serialisable):
245: (4)                tab = Typed(expected_type=TabStop, allow_none=True)
246: (4)                def __init__(self,
247: (17)                            tab=None,
248: (16)                           ):
249: (8)                    self.tab = tab
250: (0)            class Spacing(Serialisable):
251: (4)                spcPct = NestedInteger(allow_none=True)
252: (4)                spcPts = NestedInteger(allow_none=True)
253: (4)                __elements__ = ('spcPct', 'spcPts')
254: (4)                def __init__(self,
255: (17)                            spcPct=None,
256: (17)                            spcPts=None,
257: (17)                            ):
258: (8)                    self.spcPct = spcPct
259: (8)                    self.spcPts = spcPts
260: (0)            class AutonumberBullet(Serialisable):
261: (4)                type = Set(values=(['alphaLcParenBoth', 'alphaUcParenBoth',
262: (24)                                   'alphaLcParenR', 'alphaUcParenR', 'alphaLcPeriod',
'alphaUcPeriod',
263: (24)                                   'arabicParenBoth', 'arabicParenR', 'arabicPeriod',
'arabicPlain',
264: (24)                                   'romanLcParenBoth', 'romanUcParenBoth',
'romanLcParenR', 'romanUcParenR',
265: (24)                                   'romanLcPeriod', 'romanUcPeriod', 'circleNumDbPlain',
266: (24)                                   'circleNumWdBlackPlain', 'circleNumWdWhitePlain',
'arabicDbPeriod',
267: (24)                                   'arabicDbPlain', 'ea1ChsPeriod', 'ea1ChsPlain',
'ea1ChtPeriod',
268: (24)                                   'ea1ChtPlain', 'ea1JpnChsDbPeriod', 'ea1JpnKorPlain',
'ea1JpnKorPeriod',
269: (24)                                   'arabic1Minus', 'arabic2Minus', 'hebrew2Minus',
'thaiAlphaPeriod',
270: (24)                                   'thaiAlphaParenR', 'thaiAlphaParenBoth',
'thaiNumPeriod',
271: (24)                                   'thaiNumParenR', 'thaiNumParenBoth',
'hindiAlphaPeriod',
272: (24)                                   'hindiNumPeriod', 'hindiNumParenR',
'hindiAlpha1Period']))
273: (4)                startAt = Integer()
274: (4)                def __init__(self,
275: (17)                            type=None,
276: (17)                            startAt=None,
277: (16)                           ):
278: (8)                    self.type = type
279: (8)                    self.startAt = startAt
280: (0)            class ParagraphProperties(Serialisable):
281: (4)                tagname = "pPr"
282: (4)                namespace = DRAWING_NS
283: (4)                marL = Integer(allow_none=True)
284: (4)                marR = Integer(allow_none=True)
285: (4)                lvl = Integer(allow_none=True)
286: (4)                indent = Integer(allow_none=True)
287: (4)                algn = NoneSet(values=(['l', 'ctr', 'r', 'just', 'justLow', 'dist',
'thaiDist']))
```

```
(4)                defTabSz = Integer(allow_none=True)
(4)                rtl = Bool(allow_none=True)
(4)                eaLnBrk = Bool(allow_none=True)
(4)                fontAlgn = NoneSet(values=(['auto', 't', 'ctr', 'base', 'b']))
(4)                latinLnBrk = Bool(allow_none=True)
(4)                hangingPunct = Bool(allow_none=True)
(4)                lnSpc = Typed(expected_type=Spacing, allow_none=True)
(4)                spcBef = Typed(expected_type=Spacing, allow_none=True)
(4)                spcAft = Typed(expected_type=Spacing, allow_none=True)
(4)                tabLst = Typed(expected_type=TabStopList, allow_none=True)
(4)                defRPr = Typed(expected_type=CharacterProperties, allow_none=True)
(4)                extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)                buClrTx = EmptyTag()
(4)                buClr = Typed(expected_type=Color, allow_none=True)
(4)                buSzTx = EmptyTag()
(4)                buSzPct = NestedInteger(allow_none=True)
(4)                buSzPts = NestedInteger(allow_none=True)
(4)                buFontTx = EmptyTag()
(4)                buFont = Typed(expected_type=Font, allow_none=True)
(4)                buNone = EmptyTag()
(4)                buAutoNum = EmptyTag()
(4)                buChar = NestedValue(expected_type=str, attribute="char", allow_none=True)
(4)                buBlip = NestedValue(expected_type=Blip, attribute="blip", allow_none=True)
(4)                __elements__ = ('lnSpc', 'spcBef', 'spcAft', 'tabLst', 'defRPr',
(20)                               'buClrTx', 'buClr', 'buSzTx', 'buSzPct', 'buSzPts', 'buFontTx', 'buFont',
(20)                               'buNone', 'buAutoNum', 'buChar', 'buBlip')
(4)                def __init__(self,
(17)                            marL=None,
(17)                            marR=None,
(17)                            lvl=None,
(17)                            indent=None,
(17)                            algn=None,
(17)                            defTabSz=None,
(17)                            rtl=None,
(17)                            eaLnBrk=None,
(17)                            fontAlgn=None,
(17)                            latinLnBrk=None,
(17)                            hangingPunct=None,
(17)                            lnSpc=None,
(17)                            spcBef=None,
(17)                            spcAft=None,
(17)                            tabLst=None,
(17)                            defRPr=None,
(17)                            extLst=None,
(17)                            buClrTx=None,
(17)                            buClr=None,
(17)                            buSzTx=None,
(17)                            buSzPct=None,
(17)                            buSzPts=None,
(17)                            buFontTx=None,
(17)                            buFont=None,
(17)                            buNone=None,
(17)                            buAutoNum=None,
(17)                            buChar=None,
(17)                            buBlip=None,
(17)                            ):
(8)                    self.marL = marL
(8)                    self.marR = marR
(8)                    self.lvl = lvl
(8)                    self.indent = indent
(8)                    self.algn = algn
(8)                    self.defTabSz = defTabSz
(8)                    self.rtl = rtl
(8)                    self.eaLnBrk = eaLnBrk
(8)                    self.fontAlgn = fontAlgn
(8)                    self.latinLnBrk = latinLnBrk
(8)                    self.hangingPunct = hangingPunct
```

```
(8)           self.lnSpc = lnSpc
(8)           self.spcBef = spcBef
(8)           self.spcAft = spcAft
(8)           self.tabLst = tabLst
(8)           self.defRPr = defRPr
(8)           self.buClrTx = buClrTx
(8)           self.buClr = buClr
(8)           self.buSzTx = buSzTx
(8)           self.buSzPct = buSzPct
(8)           self.buSzPts = buSzPts
(8)           self.buFontTx = buFontTx
(8)           self.buFont = buFont
(8)           self.buNone = buNone
(8)           self.buAutoNum = buAutoNum
(8)           self.buChar = buChar
(8)           self.buBlip = buBlip
(8)           self.defRPr = defRPr
(0)   class ListStyle(Serialisable):
(4)       tagname = "lstStyle"
(4)       namespace = DRAWING_NS
(4)       defPPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl1pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl2pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl3pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl4pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl5pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl6pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl7pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl8pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       lvl9pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)       extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)       __elements__ = ("defPPr", "lvl1pPr", "lvl2pPr", "lvl3pPr", "lvl4pPr",
(20)                      "lvl5pPr", "lvl6pPr", "lvl7pPr", "lvl8pPr", "lvl9pPr")
(4)       def __init__(self,
(17)                   defPPr=None,
(17)                   lvl1pPr=None,
(17)                   lvl2pPr=None,
(17)                   lvl3pPr=None,
(17)                   lvl4pPr=None,
(17)                   lvl5pPr=None,
(17)                   lvl6pPr=None,
(17)                   lvl7pPr=None,
(17)                   lvl8pPr=None,
(17)                   lvl9pPr=None,
(17)                   extLst=None,
(16)                  ):
(8)           self.defPPr = defPPr
(8)           self.lvl1pPr = lvl1pPr
(8)           self.lvl2pPr = lvl2pPr
(8)           self.lvl3pPr = lvl3pPr
(8)           self.lvl4pPr = lvl4pPr
(8)           self.lvl5pPr = lvl5pPr
(8)           self.lvl6pPr = lvl6pPr
(8)           self.lvl7pPr = lvl7pPr
(8)           self.lvl8pPr = lvl8pPr
(8)           self.lvl9pPr = lvl9pPr
(0)   class RegularTextRun(Serialisable):
(4)       tagname = "r"
(4)       namespace = DRAWING_NS
(4)       rPr = Typed(expected_type=CharacterProperties, allow_none=True)
(4)       properties = Alias("rPr")
(4)       t = NestedText(expected_type=str)
(4)       value = Alias("t")
(4)       __elements__ = ('rPr', 't')
(4)       def __init__(self,
(17)                   rPr=None,
(17)                   t="",
(16)                  ):
(8)           self.rPr = rPr
```

```
(8)                 self.t = t
(0)         class LineBreak(Serialisable):
(4)             tagname = "br"
(4)             namespace = DRAWING_NS
(4)             rPr = Typed(expected_type=CharacterProperties, allow_none=True)
(4)             __elements__ = ('rPr',)
(4)             def __init__(self,
(17)                             rPr=None,
(16)                            ):
(8)                 self.rPr = rPr
(0)         class TextField(Serialisable):
(4)             id = String()
(4)             type = String(allow_none=True)
(4)             rPr = Typed(expected_type=CharacterProperties, allow_none=True)
(4)             pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)             t = String(allow_none=True)
(4)             __elements__ = ('rPr', 'pPr')
(4)             def __init__(self,
(17)                             id=None,
(17)                             type=None,
(17)                             rPr=None,
(17)                             pPr=None,
(17)                             t=None,
(16)                            ):
(8)                 self.id = id
(8)                 self.type = type
(8)                 self.rPr = rPr
(8)                 self.pPr = pPr
(8)                 self.t = t
(0)         class Paragraph(Serialisable):
(4)             tagname = "p"
(4)             namespace = DRAWING_NS
(4)             pPr = Typed(expected_type=ParagraphProperties, allow_none=True)
(4)             properties = Alias("pPr")
(4)             endParaRPr = Typed(expected_type=CharacterProperties, allow_none=True)
(4)             r = Sequence(expected_type=RegularTextRun)
(4)             text = Alias('r')
(4)             br = Typed(expected_type=LineBreak, allow_none=True)
(4)             fld = Typed(expected_type=TextField, allow_none=True)
(4)             __elements__ = ('pPr', 'r', 'br', 'fld', 'endParaRPr')
(4)             def __init__(self,
(17)                             pPr=None,
(17)                             endParaRPr=None,
(17)                             r=None,
(17)                             br=None,
(17)                             fld=None,
(17)                             ):
(8)                 self.pPr = pPr
(8)                 self.endParaRPr = endParaRPr
(8)                 if r is None:
(12)                    r = [RegularTextRun()]
(8)                 self.r = r
(8)                 self.br = br
(8)                 self.fld = fld
(0)         class GeomGuide(Serialisable):
(4)             name = String(())
(4)             fmla = String(())
(4)             def __init__(self,
(17)                             name=None,
(17)                             fmla=None,
(16)                            ):
(8)                 self.name = name
(8)                 self.fmla = fmla
(0)         class GeomGuideList(Serialisable):
(4)             gd = Sequence(expected_type=GeomGuide, allow_none=True)
(4)             def __init__(self,
(17)                             gd=None,
(16)                            ):
(8)                 self.gd = gd
```

```
(0)            class PresetTextShape(Serialisable):
(4)                prst = Typed(expected_type=Set(values=(
(8)                    ['textNoShape', 'textPlain','textStop', 'textTriangle', 'textTriangleInverted', 'textChevron',
(9)                     'textChevronInverted', 'textRingInside', 'textRingOutside', 'textArchUp',
(9)                     'textArchDown', 'textCircle', 'textButton', 'textArchUpPour',
(9)                     'textArchDownPour', 'textCirclePour', 'textButtonPour', 'textCurveUp',
(9)                     'textCurveDown', 'textCanUp', 'textCanDown', 'textWave1', 'textWave2',
(9)                     'textDoubleWave1', 'textWave4', 'textInflate', 'textDeflate',
(9)                     'textInflateBottom', 'textDeflateBottom', 'textInflateTop',
(9)                     'textDeflateTop', 'textDeflateInflate', 'textDeflateInflateDeflate',
(9)                     'textFadeRight', 'textFadeLeft', 'textFadeUp', 'textFadeDown',
(9)                     'textSlantUp', 'textSlantDown', 'textCascadeUp', 'textCascadeDown'
(9)                     ]
(4)                )))
(4)                avLst = Typed(expected_type=GeomGuideList, allow_none=True)
(4)                def __init__(self,
(17)                            prst=None,
(17)                            avLst=None,
(16)                           ):
(8)                    self.prst = prst
(8)                    self.avLst = avLst
(0)            class TextNormalAutofit(Serialisable):
(4)                fontScale = Integer()
(4)                lnSpcReduction = Integer()
(4)                def __init__(self,
(17)                            fontScale=None,
(17)                            lnSpcReduction=None,
(16)                           ):
(8)                    self.fontScale = fontScale
(8)                    self.lnSpcReduction = lnSpcReduction
(0)            class RichTextProperties(Serialisable):
(4)                tagname = "bodyPr"
(4)                namespace = DRAWING_NS
(4)                rot = Integer(allow_none=True)
(4)                spcFirstLastPara = Bool(allow_none=True)
(4)                vertOverflow = NoneSet(values=(['overflow', 'ellipsis', 'clip']))
(4)                horzOverflow = NoneSet(values=(['overflow', 'clip']))
(4)                vert = NoneSet(values=(['horz', 'vert', 'vert270', 'wordArtVert',
(28)                                       'eaVert', 'mongolianVert', 'wordArtVertRtl']))
(4)                wrap = NoneSet(values=(['none', 'square']))
(4)                lIns = Integer(allow_none=True)
(4)                tIns = Integer(allow_none=True)
(4)                rIns = Integer(allow_none=True)
(4)                bIns = Integer(allow_none=True)
(4)                numCol = Integer(allow_none=True)
(4)                spcCol = Integer(allow_none=True)
(4)                rtlCol = Bool(allow_none=True)
(4)                fromWordArt = Bool(allow_none=True)
(4)                anchor = NoneSet(values=(['t', 'ctr', 'b', 'just', 'dist']))
(4)                anchorCtr = Bool(allow_none=True)
(4)                forceAA = Bool(allow_none=True)
(4)                upright = Bool(allow_none=True)
(4)                compatLnSpc = Bool(allow_none=True)
(4)                prstTxWarp = Typed(expected_type=PresetTextShape, allow_none=True)
(4)                scene3d = Typed(expected_type=Scene3D, allow_none=True)
(4)                extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)                noAutofit = EmptyTag()
(4)                normAutofit = EmptyTag()
(4)                spAutoFit = EmptyTag()
(4)                flatTx = NestedInteger(attribute="z", allow_none=True)
(4)                __elements__ = ('prstTxWarp', 'scene3d', 'noAutofit', 'normAutofit', 'spAutoFit')
(4)                def __init__(self,
(17)                            rot=None,
(17)                            spcFirstLastPara=None,
557: (17)                           vertOverflow=None,
558: (17)                           horzOverflow=None,
559: (17)                           vert=None,
560: (17)                           wrap=None,
561: (17)                           lIns=None,
562: (17)                           tIns=None,
563: (17)                           rIns=None,
564: (17)                           bIns=None,
565: (17)                           numCol=None,
566: (17)                           spcCol=None,
567: (17)                           rtlCol=None,
568: (17)                           fromWordArt=None,
569: (17)                           anchor=None,
570: (17)                           anchorCtr=None,
571: (17)                           forceAA=None,
572: (17)                           upright=None,
573: (17)                           compatLnSpc=None,
574: (17)                           prstTxWarp=None,
575: (17)                           scene3d=None,
576: (17)                           extLst=None,
577: (17)                           noAutofit=None,
578: (17)                           normAutofit=None,
579: (17)                           spAutoFit=None,
580: (17)                           flatTx=None,
581: (16)                          ):
582: (8)                   self.rot = rot
583: (8)                   self.spcFirstLastPara = spcFirstLastPara
584: (8)                   self.vertOverflow = vertOverflow
585: (8)                   self.horzOverflow = horzOverflow
586: (8)                   self.vert = vert
587: (8)                   self.wrap = wrap
588: (8)                   self.lIns = lIns
589: (8)                   self.tIns = tIns
590: (8)                   self.rIns = rIns
591: (8)                   self.bIns = bIns
592: (8)                   self.numCol = numCol
593: (8)                   self.spcCol = spcCol
594: (8)                   self.rtlCol = rtlCol
595: (8)                   self.fromWordArt = fromWordArt
596: (8)                   self.anchor = anchor
597: (8)                   self.anchorCtr = anchorCtr
598: (8)                   self.forceAA = forceAA
599: (8)                   self.upright = upright
600: (8)                   self.compatLnSpc = compatLnSpc
601: (8)                   self.prstTxWarp = prstTxWarp
602: (8)                   self.scene3d = scene3d
603: (8)                   self.noAutofit = noAutofit
604: (8)                   self.normAutofit = normAutofit
605: (8)                   self.spAutoFit = spAutoFit
606: (8)                   self.flatTx = flatTx

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

File 69 - excel.py:

1: (0)              """
2: (0)              Excel specific descriptors
3: (0)              """
4: (0)              from openpyxl.xml.constants import REL_NS
5: (0)              from openpyxl.compat import safe_string
6: (0)              from openpyxl.xml.functions import Element
7: (0)              from . import (
8: (4)                  MatchPattern,
9: (4)                  MinMax,
10: (4)                 Integer,
11: (4)                 String,
12: (4)                 Sequence,
13: (0)             )
14: (0)             from .serialisable import Serialisable
```

```
15: (0)             class HexBinary(MatchPattern):
16: (4)                 pattern = "[0-9a-fA-F]+$"
17: (0)             class UniversalMeasure(MatchPattern):
18: (4)                 pattern = r"[0-9]+(\.[0-9]+)?(mm|cm|in|pt|pc|pi)"
19: (0)             class TextPoint(MinMax):
20: (4)                 """
21: (4)                 Size in hundredths of points.
22: (4)                 In theory other units of measurement can be used but these are unbounded
23: (4)                 """
24: (4)                 expected_type = int
25: (4)                 min = -400000
26: (4)                 max = 400000
27: (0)             Coordinate = Integer
28: (0)             class Percentage(MinMax):
29: (4)                 pattern = r"((100)|([0-9][0-9]?))(\.[0-9][0-9]?)?%" # strict
30: (4)                 min = -1000000
31: (4)                 max = 1000000
32: (4)                 def __set__(self, instance, value):
33: (8)                     if isinstance(value, str) and "%" in value:
34: (12)                        value = value.replace("%", "")
35: (12)                        value = int(float(value) * 1000)
36: (8)                     super().__set__(instance, value)
37: (0)             class Extension(Serialisable):
38: (4)                 uri = String()
39: (4)                 def __init__(self,
40: (17)                             uri=None,
41: (16)                            ):
42: (8)                     self.uri = uri
43: (0)             class ExtensionList(Serialisable):
44: (4)                 ext = Sequence(expected_type=Extension)
45: (4)                 def __init__(self,
46: (17)                             ext=(),
47: (16)                            ):
48: (8)                     self.ext = ext
49: (0)             class Relation(String):
50: (4)                 namespace = REL_NS
51: (4)                 allow_none = True
52: (0)             class Base64Binary(MatchPattern):
53: (4)                 pattern = "^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=|
[A-Za-z0-9+/]{4})$"
54: (0)             class Guid(MatchPattern):
55: (4)                 pattern = r"{[0-9A-F]{8}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]
{12}\}"
56: (0)             class CellRange(MatchPattern):
57: (4)                 pattern = r"^[$]?([A-Za-z]{1,3})[$]?(\d+)(:[$]?([A-Za-z]{1,3})[$]?(\d+)?)?
$|^[A-Za-z]{1,3}:[A-Za-z]{1,3}$"
58: (4)                 allow_none = True
59: (4)                 def __set__(self, instance, value):
60: (8)                     if value is not None:
61: (12)                        value = value.upper()
62: (8)                     super().__set__(instance, value)
63: (0)             def _explicit_none(tagname, value, namespace=None):
64: (4)                 """
65: (4)                 Override serialisation because explicit none required
66: (4)                 """
67: (4)                 if namespace is not None:
68: (8)                     tagname = "{%s}%s" % (namespace, tagname)
69: (4)                 return Element(tagname, val=safe_string(value))
```

File 70 - slots.py:

```
1: (0)              class AutoSlotProperties(type):
2: (4)                  def __new__(mcl, classname, bases, dictionary):
3: (8)                      slots = list(dictionary.get("__slots__", []))
4: (8)                      for getter_name in [key for key in dictionary if
key.startswith("get_")]:
5: (12)                         name = getter_name
```

```
6: (12)                         slots.append("__" + name)
7: (12)                         getter = dictionary.pop(getter_name)
8: (12)                         setter = dictionary.get(setter_name, None)
9: (12)                         if (setter is not None
10: (16)                            and isinstance(setter, collections.Callable)):
11: (16)                            del dictionary[setter_name]
12: (12)                        dictionary[name] = property(getter. setter)
13: (12)                        dictionary["__slots__"] = tuple(slots)
14: (12)                        return super().__new__(mcl, classname, bases, dictionary)

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

File 71 - image.py:

1: (0)              from io import BytesIO
2: (0)              try:
3: (4)                  from PIL import Image as PILImage
4: (0)              except ImportError:
5: (4)                  PILImage = False
6: (0)              def _import_image(img):
7: (4)                  if not PILImage:
8: (8)                      raise ImportError('You must install Pillow to fetch image objects')
9: (4)                  if not isinstance(img, PILImage.Image):
10: (8)                     img = PILImage.open(img)
11: (4)                 return img
12: (0)             class Image:
13: (4)                 """Image in a spreadsheet"""
14: (4)                 _id = 1
15: (4)                 _path = "/xl/media/image{0}.{1}"
16: (4)                 anchor = "A1"
17: (4)                 def __init__(self, img):
18: (8)                     self.ref = img
19: (8)                     mark_to_close = isinstance(img, str)
20: (8)                     image = _import_image(img)
21: (8)                     self.width, self.height = image.size
22: (8)                     try:
23: (12)                        self.format = image.format.lower()
24: (8)                     except AttributeError:
25: (12)                        self.format = "png"
26: (8)                     if mark_to_close:
27: (12)                        image.close()
28: (4)                 def _data(self):
29: (8)                     """
30: (8)                     Return image data, convert to supported types if necessary
31: (8)                     """
32: (8)                     img = _import_image(self.ref)
33: (8)                     if self.format in ['gif', 'jpeg', 'png']:
34: (12)                        img.fp.seek(0)
35: (12)                        fp = img.fp
36: (8)                     else:
37: (12)                        fp = BytesIO()
38: (12)                        img.save(fp, format="png")
39: (12)                        fp.seek(0)
40: (8)                     data = fp.read()
41: (8)                     fp.close()
42: (8)                     return data
43: (4)                 @property
44: (4)                 def path(self):
45: (8)                     return self._path.format(self._id, self.format)

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

File 72 - nested.py:

1: (0)              """
2: (0)              Generic serialisable classes
3: (0)              """
4: (0)              from .base import (
5: (4)                  Convertible,
(4)             Bool,
(4)             Descriptor,
(4)             NoneSet,
(4)             MinMax,
(4)             Set,
(4)             Float,
(4)             Integer,
(4)             String,
(4)             )
(0)         from openpyxl.compat import safe_string
(0)         from openpyxl.xml.functions import Element, localname, whitespace
(0)         class Nested(Descriptor):
(4)             nested = True
(4)             attribute = "val"
(4)             def __set__(self, instance, value):
(8)                 if hasattr(value, "tag"):
(12)                    tag = localname(value)
(12)                    if tag != self.name:
(16)                        raise ValueError("Tag does not match attribute")
(12)                    value = self.from_tree(value)
(8)                 super().__set__(instance, value)
(4)             def from_tree(self, node):
(8)                 return node.get(self.attribute)
(4)             def to_tree(self, tagname=None, value=None, namespace=None):
(8)                 namespace = getattr(self, "namespace", namespace)
(8)                 if value is not None:
(12)                    if namespace is not None:
(16)                        tagname = "{%s}%s" % (namespace, tagname)
(12)                    value = safe_string(value)
(12)                    return Element(tagname, {self.attribute:value})
(0)         class NestedValue(Nested, Convertible):
(4)             """
(4)             Nested tag storing the value on the 'val' attribute
(4)             """
(4)             pass
(0)         class NestedText(NestedValue):
(4)             """
(4)             Represents any nested tag with the value as the contents of the tag
(4)             """
(4)             def from_tree(self, node):
(8)                 return node.text
(4)             def to_tree(self, tagname=None, value=None, namespace=None):
(8)                 namespace = getattr(self, "namespace", namespace)
(8)                 if value is not None:
(12)                    if namespace is not None:
(16)                        tagname = "{%s}%s" % (namespace, tagname)
(12)                    el = Element(tagname)
(12)                    el.text = safe_string(value)
(12)                    whitespace(el)
(12)                    return el
(0)         class NestedFloat(NestedValue, Float):
(4)             pass
(0)         class NestedInteger(NestedValue, Integer):
(4)             pass
(0)         class NestedString(NestedValue, String):
(4)             pass
(0)         class NestedBool(NestedValue, Bool):
(4)             def from_tree(self, node):
(8)                 return node.get("val", True)
(0)         class NestedNoneSet(Nested, NoneSet):
(4)             pass
(0)         class NestedSet(Nested, Set):
(4)             pass
(0)         class NestedMinMax(Nested, MinMax):
(4)             pass
(0)         class EmptyTag(Nested, Bool):
(4)             """
(4)             Boolean if a tag exists or not.
(4)             """
```

```
75: (4)                 def from_tree(self, node):
76: (8)                     return True
77: (4)                 def to_tree(self, tagname=None, value=None, namespace=None):
78: (8)                     if value:
79: (12)                        namespace = getattr(self, "namespace", namespace)
80: (12)                        if namespace is not None:
81: (16)                            tagname = "{%s}%s" % (namespace, tagname)
82: (12)                        return Element(tagname)

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

File 73 - colors.py:

1: (0)                from openpyxl.descriptors.serialisable import Serialisable
2: (0)                from openpyxl.descriptors import (
3: (4)                    Alias,
4: (4)                    Typed,
5: (4)                    Integer,
6: (4)                    Set,
7: (4)                    MinMax,
8: (0)                )
9: (0)                from openpyxl.descriptors.excel import Percentage
10: (0)               from openpyxl.descriptors.nested import (
11: (4)                   NestedNoneSet,
12: (4)                   NestedValue,
13: (4)                   NestedInteger,
14: (4)                   EmptyTag,
15: (0)               )
16: (0)               from openpyxl.styles.colors import RGB
17: (0)               from openpyxl.xml.constants import DRAWING_NS
18: (0)               from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
19: (0)               PRESET_COLORS = [
20: (8)                       'aliceBlue', 'antiqueWhite', 'aqua', 'aquamarine',
21: (8)                       'azure', 'beige', 'bisque', 'black', 'blanchedAlmond', 'blue',
22: (8)                       'blueViolet', 'brown', 'burlyWood', 'cadetBlue', 'chartreuse',
23: (8)                       'chocolate', 'coral', 'cornflowerBlue', 'cornsilk', 'crimson', 'cyan',
24: (8)                       'darkBlue', 'darkCyan', 'darkGoldenrod', 'darkGray', 'darkGrey',
25: (8)                       'darkGreen', 'darkKhaki', 'darkMagenta', 'darkOliveGreen',
'darkOrange',
26: (8)                       'darkOrchid', 'darkRed', 'darkSalmon', 'darkSeaGreen',
'darkSlateBlue',
27: (8)                       'darkSlateGray', 'darkSlateGrey', 'darkTurquoise', 'darkViolet',
28: (8)                       'dkBlue', 'dkCyan', 'dkGoldenrod', 'dkGray', 'dkGrey', 'dkGreen',
29: (8)                       'dkKhaki', 'dkMagenta', 'dkOliveGreen', 'dkOrange', 'dkOrchid',
'dkRed',
30: (8)                       'dkSalmon', 'dkSeaGreen', 'dkSlateBlue', 'dkSlateGray', 'dkSlateGrey',
31: (8)                       'dkTurquoise', 'dkViolet', 'deepPink', 'deepSkyBlue', 'dimGray',
32: (8)                       'dimGrey', 'dodgerBlue', 'firebrick', 'floralWhite', 'forestGreen',
33: (8)                       'fuchsia', 'gainsboro', 'ghostWhite', 'gold', 'goldenrod', 'gray',
34: (8)                       'grey', 'green', 'greenYellow', 'honeydew', 'hotPink', 'indianRed',
35: (8)                       'indigo', 'ivory', 'khaki', 'lavender', 'lavenderBlush', 'lawnGreen',
36: (8)                       'lemonChiffon', 'lightBlue', 'lightCoral', 'lightCyan',
37: (8)                       'lightGoldenrodYellow', 'lightGray', 'lightGrey', 'lightGreen',
38: (8)                       'lightPink', 'lightSalmon', 'lightSeaGreen', 'lightSkyBlue',
39: (8)                       'lightSlateGray', 'lightSlateGrey', 'lightSteelBlue', 'lightYellow',
40: (8)                       'ltBlue', 'ltCoral', 'ltCyan', 'ltGoldenrodYellow', 'ltGray',
'ltGrey',
41: (8)                       'ltGreen', 'ltPink', 'ltSalmon', 'ltSeaGreen', 'ltSkyBlue',
42: (8)                       'ltSlateGray', 'ltSlateGrey', 'ltSteelBlue', 'ltYellow', 'lime',
43: (8)                       'limeGreen', 'linen', 'magenta', 'maroon', 'medAquamarine', 'medBlue',
44: (8)                       'medOrchid', 'medPurple', 'medSeaGreen', 'medSlateBlue',
45: (8)                       'medSpringGreen', 'medTurquoise', 'medVioletRed', 'mediumAquamarine',
46: (8)                       'mediumBlue', 'mediumOrchid', 'mediumPurple', 'mediumSeaGreen',
47: (8)                       'mediumSlateBlue', 'mediumSpringGreen', 'mediumTurquoise',
48: (8)                       'mediumVioletRed', 'midnightBlue', 'mintCream', 'mistyRose',
'moccasin',
49: (8)                       'navajoWhite', 'navy', 'oldLace', 'olive', 'oliveDrab', 'orange',
50: (8)                       'orangeRed', 'orchid', 'paleGoldenrod', 'paleGreen', 'paleTurquoise',
51: (8)                       'paleVioletRed', 'papayaWhip', 'peachPuff', 'peru', 'pink', 'plum',
52: (8)                     'powderBlue', 'purple', 'red', 'rosyBrown', 'royalBlue',
'saddleBrown',
53: (8)                     'salmon', 'sandyBrown', 'seaGreen', 'seaShell', 'sienna', 'silver',
54: (8)                     'skyBlue', 'slateBlue', 'slateGray', 'slateGrey', 'snow',
'springGreen',
55: (8)                     'steelBlue', 'tan', 'teal', 'thistle', 'tomato', 'turquoise',
'violet',
56: (8)                     'wheat', 'white', 'whiteSmoke', 'yellow', 'yellowGreen'
57: (4)                 ]
58: (0)             SCHEME_COLORS= ['bg1', 'tx1', 'bg2', 'tx2', 'accent1', 'accent2', 'accent3',
59: (16)                            'accent4', 'accent5', 'accent6', 'hlink', 'folHlink', 'phClr',
'dk1', 'lt1',
60: (16)                            'dk2', 'lt2'
61: (16)                            ]
62: (0)             class Transform(Serialisable):
63: (4)                 pass
64: (0)             class SystemColor(Serialisable):
65: (4)                 tagname = "sysClr"
66: (4)                 namespace = DRAWING_NS
67: (4)                 tint = NestedInteger(allow_none=True)
68: (4)                 shade = NestedInteger(allow_none=True)
69: (4)                 comp = Typed(expected_type=Transform, allow_none=True)
70: (4)                 inv = Typed(expected_type=Transform, allow_none=True)
71: (4)                 gray = Typed(expected_type=Transform, allow_none=True)
72: (4)                 alpha = NestedInteger(allow_none=True)
73: (4)                 alphaOff = NestedInteger(allow_none=True)
74: (4)                 alphaMod = NestedInteger(allow_none=True)
75: (4)                 hue = NestedInteger(allow_none=True)
76: (4)                 hueOff = NestedInteger(allow_none=True)
77: (4)                 hueMod = NestedInteger(allow_none=True)
78: (4)                 sat = NestedInteger(allow_none=True)
79: (4)                 satOff = NestedInteger(allow_none=True)
80: (4)                 satMod = NestedInteger(allow_none=True)
81: (4)                 lum = NestedInteger(allow_none=True)
82: (4)                 lumOff = NestedInteger(allow_none=True)
83: (4)                 lumMod = NestedInteger(allow_none=True)
84: (4)                 red = NestedInteger(allow_none=True)
85: (4)                 redOff = NestedInteger(allow_none=True)
86: (4)                 redMod = NestedInteger(allow_none=True)
87: (4)                 green = NestedInteger(allow_none=True)
88: (4)                 greenOff = NestedInteger(allow_none=True)
89: (4)                 greenMod = NestedInteger(allow_none=True)
90: (4)                 blue = NestedInteger(allow_none=True)
91: (4)                 blueOff = NestedInteger(allow_none=True)
92: (4)                 blueMod = NestedInteger(allow_none=True)
93: (4)                 gamma = Typed(expected_type=Transform, allow_none=True)
94: (4)                 invGamma = Typed(expected_type=Transform, allow_none=True)
95: (4)                 val = Set(values=( ['scrollBar', 'background', 'activeCaption',
96: (24)                                    'inactiveCaption', 'menu', 'window', 'windowFrame',
'menuText',
97: (24)                                    'windowText', 'captionText', 'activeBorder',
'inactiveBorder',
98: (24)                                    'appWorkspace', 'highlight', 'highlightText',
'btnFace', 'btnShadow',
99: (24)                                    'grayText', 'btnText', 'inactiveCaptionText',
'btnHighlight',
100: (24)                                   '3dDkShadow', '3dLight', 'infoText', 'infoBk',
'hotLight',
101: (24)                                   'gradientActiveCaption', 'gradientInactiveCaption',
'menuHighlight',
102: (24)                                   'menuBar'] )
103: (14)                          )
104: (4)                lastClr = RGB(allow_none=True)
105: (4)                __elements__ = ('tint', 'shade', 'comp', 'inv', 'gray', "alpha",
106: (20)                               "alphaOff", "alphaMod", "hue", "hueOff", "hueMod",
"hueOff", "sat",
107: (20)                               "satOff", "satMod", "lum", "lumOff", "lumMod", "red",
"redOff", "redMod",
108: (20)                               "green", "greenOff", "greenMod", "blue", "blueOff",
"blueMod", "gamma",
109: (20)                               "invGamma")
110: (4)                def __init__(self,
111: (17)                            val="windowText",
112: (17)                            lastClr=None,
113: (17)                            tint=None,
114: (17)                            shade=None,
115: (17)                            comp=None,
116: (17)                            inv=None,
117: (17)                            gray=None,
118: (17)                            alpha=None,
119: (17)                            alphaOff=None,
120: (17)                            alphaMod=None,
121: (17)                            hue=None,
122: (17)                            hueOff=None,
123: (17)                            hueMod=None,
124: (17)                            sat=None,
125: (17)                            satOff=None,
126: (17)                            satMod=None,
127: (17)                            lum=None,
128: (17)                            lumOff=None,
129: (17)                            lumMod=None,
130: (17)                            red=None,
131: (17)                            redOff=None,
132: (17)                            redMod=None,
133: (17)                            green=None,
134: (17)                            greenOff=None,
135: (17)                            greenMod=None,
136: (17)                            blue=None,
137: (17)                            blueOff=None,
138: (17)                            blueMod=None,
139: (17)                            gamma=None,
140: (17)                            invGamma=None
141: (16)                           ):
142: (8)                    self.val = val
143: (8)                    self.lastClr = lastClr
144: (8)                    self.tint = tint
145: (8)                    self.shade = shade
146: (8)                    self.comp = comp
147: (8)                    self.inv = inv
148: (8)                    self.gray = gray
149: (8)                    self.alpha = alpha
150: (8)                    self.alphaOff = alphaOff
151: (8)                    self.alphaMod = alphaMod
152: (8)                    self.hue = hue
153: (8)                    self.hueOff = hueOff
154: (8)                    self.hueMod = hueMod
155: (8)                    self.sat = sat
156: (8)                    self.satOff = satOff
157: (8)                    self.satMod = satMod
158: (8)                    self.lum = lum
159: (8)                    self.lumOff = lumOff
160: (8)                    self.lumMod = lumMod
161: (8)                    self.red = red
162: (8)                    self.redOff = redOff
163: (8)                    self.redMod = redMod
164: (8)                    self.green = green
165: (8)                    self.greenOff = greenOff
166: (8)                    self.greenMod = greenMod
167: (8)                    self.blue = blue
168: (8)                    self.blueOff = blueOff
169: (8)                    self.blueMod = blueMod
170: (8)                    self.gamma = gamma
171: (8)                    self.invGamma = invGamma
172: (0)            class HSLColor(Serialisable):
173: (4)                tagname = "hslClr"
174: (4)                hue = Integer()
175: (4)                sat = MinMax(min=0, max=100)
176: (4)                lum = MinMax(min=0, max=100)
```

```
177: (4)                def __init__(self,
178: (17)                            hue=None,
179: (17)                            sat=None,
180: (17)                            lum=None,
181: (16)                           ):
182: (8)                    self.hue = hue
183: (8)                    self.sat = sat
184: (8)                    self.lum = lum
185: (0)            class RGBPercent(Serialisable):
186: (4)                tagname = "rgbClr"
187: (4)                r = MinMax(min=0, max=100)
188: (4)                g = MinMax(min=0, max=100)
189: (4)                b = MinMax(min=0, max=100)
190: (4)                def __init__(self,
191: (17)                            r=None,
192: (17)                            g=None,
193: (17)                            b=None,
194: (16)                           ):
195: (8)                    self.r = r
196: (8)                    self.g = g
197: (8)                    self.b = b
198: (0)            class SchemeColor(Serialisable):
199: (4)                tagname = "schemeClr"
200: (4)                namespace = DRAWING_NS
201: (4)                tint = NestedInteger(allow_none=True)
202: (4)                shade = NestedInteger(allow_none=True)
203: (4)                comp = EmptyTag(allow_none=True)
204: (4)                inv = NestedInteger(allow_none=True)
205: (4)                gray = NestedInteger(allow_none=True)
206: (4)                alpha = NestedInteger(allow_none=True)
207: (4)                alphaOff = NestedInteger(allow_none=True)
208: (4)                alphaMod = NestedInteger(allow_none=True)
209: (4)                hue = NestedInteger(allow_none=True)
210: (4)                hueOff = NestedInteger(allow_none=True)
211: (4)                hueMod = NestedInteger(allow_none=True)
212: (4)                sat = NestedInteger(allow_none=True)
213: (4)                satOff = NestedInteger(allow_none=True)
214: (4)                satMod = NestedInteger(allow_none=True)
215: (4)                lum = NestedInteger(allow_none=True)
216: (4)                lumOff = NestedInteger(allow_none=True)
217: (4)                lumMod = NestedInteger(allow_none=True)
218: (4)                red = NestedInteger(allow_none=True)
219: (4)                redOff = NestedInteger(allow_none=True)
220: (4)                redMod = NestedInteger(allow_none=True)
221: (4)                green = NestedInteger(allow_none=True)
222: (4)                greenOff = NestedInteger(allow_none=True)
223: (4)                greenMod = NestedInteger(allow_none=True)
224: (4)                blue = NestedInteger(allow_none=True)
225: (4)                blueOff = NestedInteger(allow_none=True)
226: (4)                blueMod = NestedInteger(allow_none=True)
227: (4)                gamma = EmptyTag(allow_none=True)
228: (4)                invGamma = EmptyTag(allow_none=True)
229: (4)                val = Set(values=(['bg1', 'tx1', 'bg2', 'tx2', 'accent1', 'accent2',
230: (23)                              'accent3', 'accent4', 'accent5', 'accent6', 'hlink',
'folHlink', 'phClr',
231: (23)                              'dk1', 'lt1', 'dk2', 'lt2']))
232: (4)                __elements__ = ('tint', 'shade', 'comp', 'inv', 'gray', 'alpha',
233: (20)                          'alphaOff', 'alphaMod', 'hue', 'hueOff', 'hueMod', 'sat',
'satOff',
234: (20)                          'satMod', 'lum', 'lumMod', 'lumOff', 'red', 'redOff',
'redMod', 'green',
235: (20)                          'greenOff', 'greenMod', 'blue', 'blueOff', 'blueMod',
'gamma',
236: (20)                          'invGamma')
237: (4)                def __init__(self,
238: (17)                            tint=None,
239: (17)                            shade=None,
240: (17)                            comp=None,
241: (17)                            inv=None,
242: (17)                                gray=None,
243: (17)                                alpha=None,
244: (17)                                alphaOff=None,
245: (17)                                alphaMod=None,
246: (17)                                hue=None,
247: (17)                                hueOff=None,
248: (17)                                hueMod=None,
249: (17)                                sat=None,
250: (17)                                satOff=None,
251: (17)                                satMod=None,
252: (17)                                lum=None,
253: (17)                                lumOff=None,
254: (17)                                lumMod=None,
255: (17)                                red=None,
256: (17)                                redOff=None,
257: (17)                                redMod=None,
258: (17)                                green=None,
259: (17)                                greenOff=None,
260: (17)                                greenMod=None,
261: (17)                                blue=None,
262: (17)                                blueOff=None,
263: (17)                                blueMod=None,
264: (17)                                gamma=None,
265: (17)                                invGamma=None,
266: (17)                                val=None,
267: (16)                               ):
268: (8)                        self.tint = tint
269: (8)                        self.shade = shade
270: (8)                        self.comp = comp
271: (8)                        self.inv = inv
272: (8)                        self.gray = gray
273: (8)                        self.alpha = alpha
274: (8)                        self.alphaOff = alphaOff
275: (8)                        self.alphaMod = alphaMod
276: (8)                        self.hue = hue
277: (8)                        self.hueOff = hueOff
278: (8)                        self.hueMod = hueMod
279: (8)                        self.sat = sat
280: (8)                        self.satOff = satOff
281: (8)                        self.satMod = satMod
282: (8)                        self.lum = lum
283: (8)                        self.lumOff = lumOff
284: (8)                        self.lumMod = lumMod
285: (8)                        self.red = red
286: (8)                        self.redOff = redOff
287: (8)                        self.redMod = redMod
288: (8)                        self.green = green
289: (8)                        self.greenOff = greenOff
290: (8)                        self.greenMod = greenMod
291: (8)                        self.blue = blue
292: (8)                        self.blueOff = blueOff
293: (8)                        self.blueMod = blueMod
294: (8)                        self.gamma = gamma
295: (8)                        self.invGamma = invGamma
296: (8)                        self.val = val
297: (0)                class ColorChoice(Serialisable):
298: (4)                    tagname = "colorChoice"
299: (4)                    namespace = DRAWING_NS
300: (4)                    scrgbClr = Typed(expected_type=RGBPercent, allow_none=True)
301: (4)                    RGBPercent = Alias('scrgbClr')
302: (4)                    srgbClr = NestedValue(expected_type=str, allow_none=True) # needs pattern
and can have transform
303: (4)                    RGB = Alias('srgbClr')
304: (4)                    hslClr = Typed(expected_type=HSLColor, allow_none=True)
305: (4)                    sysClr = Typed(expected_type=SystemColor, allow_none=True)
306: (4)                    schemeClr = Typed(expected_type=SchemeColor, allow_none=True)
307: (4)                    prstClr = NestedNoneSet(values=PRESET_COLORS)
308: (4)                    __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
```

```
(4)             def __init__(self,
(17)                         scrgbClr=None,
(17)                         srgbClr=None,
(17)                         hslClr=None,
(17)                         sysClr=None,
(17)                         schemeClr=None,
(17)                         prstClr=None,
(16)                        ):
(8)                 self.scrgbClr = scrgbClr
(8)                 self.srgbClr = srgbClr
(8)                 self.hslClr = hslClr
(8)                 self.sysClr = sysClr
(8)                 self.schemeClr = schemeClr
(8)                 self.prstClr = prstClr
(0)         _COLOR_SET = ('dk1', 'lt1', 'dk2', 'lt2', 'accent1', 'accent2', 'accent3',
(15)                       'accent4', 'accent5', 'accent6', 'hlink', 'folHlink')
(0)         class ColorMapping(Serialisable):
(4)             tagname = "clrMapOvr"
(4)             bg1 = Set(values=_COLOR_SET)
(4)             tx1 = Set(values=_COLOR_SET)
(4)             bg2 = Set(values=_COLOR_SET)
(4)             tx2 = Set(values=_COLOR_SET)
(4)             accent1 = Set(values=_COLOR_SET)
(4)             accent2 = Set(values=_COLOR_SET)
(4)             accent3 = Set(values=_COLOR_SET)
(4)             accent4 = Set(values=_COLOR_SET)
(4)             accent5 = Set(values=_COLOR_SET)
(4)             accent6 = Set(values=_COLOR_SET)
(4)             hlink = Set(values=_COLOR_SET)
(4)             folHlink = Set(values=_COLOR_SET)
(4)             extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)             def __init__(self,
(17)                         bg1="lt1",
(17)                         tx1="dk1",
(17)                         bg2="lt2",
(17)                         tx2="dk2",
(17)                         accent1="accent1",
(17)                         accent2="accent2",
(17)                         accent3="accent3",
(17)                         accent4="accent4",
(17)                         accent5="accent5",
(17)                         accent6="accent6",
(17)                         hlink="hlink",
(17)                         folHlink="folHlink",
(17)                         extLst=None,
(16)                        ):
(8)                 self.bg1 = bg1
(8)                 self.tx1 = tx1
(8)                 self.bg2 = bg2
(8)                 self.tx2 = tx2
(8)                 self.accent1 = accent1
(8)                 self.accent2 = accent2
(8)                 self.accent3 = accent3
(8)                 self.accent4 = accent4
(8)                 self.accent5 = accent5
(8)                 self.accent6 = accent6
(8)                 self.hlink = hlink
(8)                 self.folHlink = folHlink
(8)                 self.extLst = extLst
(0)         class ColorChoiceDescriptor(Typed):
(4)             """
(4)             Objects can choose from 7 different kinds of color system.
(4)             Assume RGBHex if a string is passed in.
(4)             """
(4)             expected_type = ColorChoice
(4)             allow_none = True
(4)             def __set__(self, instance, value):
(8)                 if isinstance(value, str):
(12)                    value = ColorChoice(srgbClr=value)
```

```
(8)          else:
(12)             if hasattr(self, "namespace") and value is not None:
(16)                 value.namespace = self.namespace
(8)          super().__set__(instance, value)
```

File 74 - effect.py:

```
(0)      from openpyxl.descriptors.serialisable import Serialisable
(0)      from openpyxl.descriptors import (
(4)          Typed,
(4)          String,
(4)          Set,
(4)          Bool,
(4)          Integer,
(4)          Float,
(0)      )
(0)      from .colors import ColorChoice
(0)      class TintEffect(Serialisable):
(4)          tagname = "tint"
(4)          hue = Integer()
(4)          amt = Integer()
(4)          def __init__(self,
(17)                      hue=0,
(17)                      amt=0,
(16)                     ):
(8)              self.hue = hue
(8)              self.amt = amt
(0)      class LuminanceEffect(Serialisable):
(4)          tagname = "lum"
(4)          bright = Integer() #Pct ?
(4)          contrast = Integer() #Pct#
(4)          def __init__(self,
(17)                      bright=0,
(17)                      contrast=0,
(16)                     ):
(8)              self.bright = bright
(8)              self.contrast = contrast
(0)      class HSLEffect(Serialisable):
(4)          hue = Integer()
(4)          sat = Integer()
(4)          lum = Integer()
(4)          def __init__(self,
(17)                      hue=None,
(17)                      sat=None,
(17)                      lum=None,
(16)                     ):
(8)              self.hue = hue
(8)              self.sat = sat
(8)              self.lum = lum
(0)      class GrayscaleEffect(Serialisable):
(4)          tagname = "grayscl"
(0)      class FillOverlayEffect(Serialisable):
(4)          blend = Set(values=(['over', 'mult', 'screen', 'darken', 'lighten']))
(4)          def __init__(self,
(17)                      blend=None,
(16)                     ):
(8)              self.blend = blend
(0)      class DuotoneEffect(Serialisable):
(4)          pass
(0)      class ColorReplaceEffect(Serialisable):
(4)          pass
(0)      class Color(Serialisable):
(4)          pass
(0)      class ColorChangeEffect(Serialisable):
(4)          useA = Bool(allow_none=True)
(4)          clrFrom = Typed(expected_type=Color, )
(4)          clrTo = Typed(expected_type=Color, )
```

```
    def __init__(self,
                 useA=None,
                 clrFrom=None,
                 clrTo=None,
                ):
        self.useA = useA
        self.clrFrom = clrFrom
        self.clrTo = clrTo
class BlurEffect(Serialisable):
    rad = Float()
    grow = Bool(allow_none=True)
    def __init__(self,
                 rad=None,
                 grow=None,
                ):
        self.rad = rad
        self.grow = grow
class BiLevelEffect(Serialisable):
    thresh = Integer()
    def __init__(self,
                 thresh=None,
                ):
        self.thresh = thresh
class AlphaReplaceEffect(Serialisable):
    a = Integer()
    def __init__(self,
                 a=None,
                ):
        self.a = a
class AlphaModulateFixedEffect(Serialisable):
    amt = Integer()
    def __init__(self,
                 amt=None,
                ):
        self.amt = amt
class EffectContainer(Serialisable):
    type = Set(values=(['sib', 'tree']))
    name = String(allow_none=True)
    def __init__(self,
                 type=None,
                 name=None,
                ):
        self.type = type
        self.name = name
class AlphaModulateEffect(Serialisable):
    cont = Typed(expected_type=EffectContainer, )
    def __init__(self,
                 cont=None,
                ):
        self.cont = cont
class AlphaInverseEffect(Serialisable):
    pass
class AlphaFloorEffect(Serialisable):
    pass
class AlphaCeilingEffect(Serialisable):
    pass
class AlphaBiLevelEffect(Serialisable):
    thresh = Integer()
    def __init__(self,
                 thresh=None,
                ):
        self.thresh = thresh
class GlowEffect(ColorChoice):
    rad = Float()
    scrgbClr = ColorChoice.scrgbClr
    srgbClr = ColorChoice.srgbClr
    hslClr = ColorChoice.hslClr
    sysClr = ColorChoice.sysClr
    schemeClr = ColorChoice.schemeClr
```

```
130: (4)            prstClr = ColorChoice.prstClr
131: (4)            __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
132: (4)            def __init__(self,
133: (17)                         rad=None,
134: (17)                         **kw
135: (16)                        ):
136: (8)                self.rad = rad
137: (8)                super().__init__(**kw)
138: (0)        class InnerShadowEffect(ColorChoice):
139: (4)            blurRad = Float()
140: (4)            dist = Float()
141: (4)            dir = Integer()
142: (4)            scrgbClr = ColorChoice.scrgbClr
143: (4)            srgbClr = ColorChoice.srgbClr
144: (4)            hslClr = ColorChoice.hslClr
145: (4)            sysClr = ColorChoice.sysClr
146: (4)            schemeClr = ColorChoice.schemeClr
147: (4)            prstClr = ColorChoice.prstClr
148: (4)            __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
149: (4)            def __init__(self,
150: (17)                         blurRad=None,
151: (17)                         dist=None,
152: (17)                         dir=None,
153: (17)                         **kw
154: (17)                         ):
155: (8)                self.blurRad = blurRad
156: (8)                self.dist = dist
157: (8)                self.dir = dir
158: (8)                super().__init__(**kw)
159: (0)        class OuterShadow(ColorChoice):
160: (4)            tagname = "outerShdw"
161: (4)            blurRad = Float(allow_none=True)
162: (4)            dist = Float(allow_none=True)
163: (4)            dir = Integer(allow_none=True)
164: (4)            sx = Integer(allow_none=True)
165: (4)            sy = Integer(allow_none=True)
166: (4)            kx = Integer(allow_none=True)
167: (4)            ky = Integer(allow_none=True)
168: (4)            algn = Set(values=['tl', 't', 'tr', 'l', 'ctr', 'r', 'bl', 'b', 'br'])
169: (4)            rotWithShape = Bool(allow_none=True)
170: (4)            scrgbClr = ColorChoice.scrgbClr
171: (4)            srgbClr = ColorChoice.srgbClr
172: (4)            hslClr = ColorChoice.hslClr
173: (4)            sysClr = ColorChoice.sysClr
174: (4)            schemeClr = ColorChoice.schemeClr
175: (4)            prstClr = ColorChoice.prstClr
176: (4)            __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
177: (4)            def __init__(self,
178: (17)                         blurRad=None,
179: (17)                         dist=None,
180: (17)                         dir=None,
181: (17)                         sx=None,
182: (17)                         sy=None,
183: (17)                         kx=None,
184: (17)                         ky=None,
185: (17)                         algn=None,
186: (17)                         rotWithShape=None,
187: (17)                         **kw
188: (16)                        ):
189: (8)                self.blurRad = blurRad
190: (8)                self.dist = dist
191: (8)                self.dir = dir
192: (8)                self.sx = sx
193: (8)                self.sy = sy
194: (8)                self.kx = kx
195: (8)                self.ky = ky
```

```
196: (8)                    self.algn = algn
197: (8)                    self.rotWithShape = rotWithShape
198: (8)                    super().__init__(**kw)
199: (0)            class PresetShadowEffect(ColorChoice):
200: (4)                prst = Set(values=(['shdw1', 'shdw2', 'shdw3', 'shdw4', 'shdw5', 'shdw6',
201: (24)                                   'shdw7', 'shdw8', 'shdw9', 'shdw10', 'shdw11',
'shdw12', 'shdw13',
202: (24)                                   'shdw14', 'shdw15', 'shdw16', 'shdw17', 'shdw18',
'shdw19', 'shdw20']))
203: (4)                dist = Float()
204: (4)                dir = Integer()
205: (4)                scrgbClr = ColorChoice.scrgbClr
206: (4)                srgbClr = ColorChoice.srgbClr
207: (4)                hslClr = ColorChoice.hslClr
208: (4)                sysClr = ColorChoice.sysClr
209: (4)                schemeClr = ColorChoice.schemeClr
210: (4)                prstClr = ColorChoice.prstClr
211: (4)                __elements__ = ('scrgbClr', 'srgbClr', 'hslClr', 'sysClr', 'schemeClr',
'prstClr')
212: (4)                def __init__(self,
213: (17)                             prst=None,
214: (17)                             dist=None,
215: (17)                             dir=None,
216: (17)                             **kw
217: (16)                            ):
218: (8)                    self.prst = prst
219: (8)                    self.dist = dist
220: (8)                    self.dir = dir
221: (8)                    super().__init__(**kw)
222: (0)            class ReflectionEffect(Serialisable):
223: (4)                blurRad = Float()
224: (4)                stA = Integer()
225: (4)                stPos = Integer()
226: (4)                endA = Integer()
227: (4)                endPos = Integer()
228: (4)                dist = Float()
229: (4)                dir = Integer()
230: (4)                fadeDir = Integer()
231: (4)                sx = Integer()
232: (4)                sy = Integer()
233: (4)                kx = Integer()
234: (4)                ky = Integer()
235: (4)                algn = Set(values=(['tl', 't', 'tr', 'l', 'ctr', 'r', 'bl', 'b', 'br']))
236: (4)                rotWithShape = Bool(allow_none=True)
237: (4)                def __init__(self,
238: (17)                             blurRad=None,
239: (17)                             stA=None,
240: (17)                             stPos=None,
241: (17)                             endA=None,
242: (17)                             endPos=None,
243: (17)                             dist=None,
244: (17)                             dir=None,
245: (17)                             fadeDir=None,
246: (17)                             sx=None,
247: (17)                             sy=None,
248: (17)                             kx=None,
249: (17)                             ky=None,
250: (17)                             algn=None,
251: (17)                             rotWithShape=None,
252: (16)                            ):
253: (8)                    self.blurRad = blurRad
254: (8)                    self.stA = stA
255: (8)                    self.stPos = stPos
256: (8)                    self.endA = endA
257: (8)                    self.endPos = endPos
258: (8)                    self.dist = dist
259: (8)                    self.dir = dir
260: (8)                    self.fadeDir = fadeDir
261: (8)                    self.sx = sx
```

```
262: (8)                self.sy = sy
263: (8)                self.kx = kx
264: (8)                self.ky = ky
265: (8)                self.algn = algn
266: (8)                self.rotWithShape = rotWithShape
267: (0)        class SoftEdgesEffect(Serialisable):
268: (4)            rad = Float()
269: (4)            def __init__(self,
270: (17)                        rad=None,
271: (16)                       ):
272: (8)                self.rad = rad
273: (0)        class EffectList(Serialisable):
274: (4)            blur = Typed(expected_type=BlurEffect, allow_none=True)
275: (4)            fillOverlay = Typed(expected_type=FillOverlayEffect, allow_none=True)
276: (4)            glow = Typed(expected_type=GlowEffect, allow_none=True)
277: (4)            innerShdw = Typed(expected_type=InnerShadowEffect, allow_none=True)
278: (4)            outerShdw = Typed(expected_type=OuterShadow, allow_none=True)
279: (4)            prstShdw = Typed(expected_type=PresetShadowEffect, allow_none=True)
280: (4)            reflection = Typed(expected_type=ReflectionEffect, allow_none=True)
281: (4)            softEdge = Typed(expected_type=SoftEdgesEffect, allow_none=True)
282: (4)            __elements__ = ('blur', 'fillOverlay', 'glow', 'innerShdw', 'outerShdw',
283: (20)                           'prstShdw', 'reflection', 'softEdge')
284: (4)            def __init__(self,
285: (17)                        blur=None,
286: (17)                        fillOverlay=None,
287: (17)                        glow=None,
288: (17)                        innerShdw=None,
289: (17)                        outerShdw=None,
290: (17)                        prstShdw=None,
291: (17)                        reflection=None,
292: (17)                        softEdge=None,
293: (16)                       ):
294: (8)                self.blur = blur
295: (8)                self.fillOverlay = fillOverlay
296: (8)                self.glow = glow
297: (8)                self.innerShdw = innerShdw
298: (8)                self.outerShdw = outerShdw
299: (8)                self.prstShdw = prstShdw
300: (8)                self.reflection = reflection
301: (8)                self.softEdge = softEdge

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

File 75 - drawing.py:

1: (0)          import math
2: (0)          from openpyxl.utils.units import pixels_to_EMU
3: (0)          class Drawing:
4: (4)              """ a drawing object - eg container for shapes or charts
5: (8)                  we assume user specifies dimensions in pixels; units are
6: (8)                  converted to EMU in the drawing part
7: (4)              """
8: (4)              count = 0
9: (4)              def __init__(self):
10: (8)                 self.name = ''
11: (8)                 self.description = ''
12: (8)                 self.coordinates = ((1, 2), (16, 8))
13: (8)                 self.left = 0
14: (8)                 self.top = 0
15: (8)                 self._width = 21 # default in px
16: (8)                 self._height = 192 #default in px
17: (8)                 self.resize_proportional = False
18: (8)                 self.rotation = 0
19: (8)                 self.anchortype = "absolute"
20: (8)                 self.anchorcol = 0 # left cell
21: (8)                 self.anchorrow = 0 # top row
22: (4)             @property
23: (4)             def width(self):
24: (8)                 return self._width
```

```
(4)                 @width.setter
(4)                 def width(self, w):
(8)                     if self.resize_proportional and w:
(12)                        ratio = self._height / self._width
(12)                        self._height = round(ratio * w)
(8)                     self._width = w
(4)                 @property
(4)                 def height(self):
(8)                     return self._height
(4)                 @height.setter
(4)                 def height(self, h):
(8)                     if self.resize_proportional and h:
(12)                        ratio = self._width / self._height
(12)                        self._width = round(ratio * h)
(8)                     self._height = h
(4)                 def set_dimension(self, w=0, h=0):
(8)                     xratio = w / self._width
(8)                     yratio = h / self._height
(8)                     if self.resize_proportional and w and h:
(12)                        if (xratio * self._height) < h:
(16)                            self._height = math.ceil(xratio * self._height)
(16)                            self._width = w
(12)                        else:
(16)                            self._width = math.ceil(yratio * self._width)
(16)                            self._height = h
(4)                 @property
(4)                 def anchor(self):
(8)                     from .spreadsheet_drawing import (
(12)                        OneCellAnchor,
(12)                        TwoCellAnchor,
(12)                        AbsoluteAnchor)
(8)                     if self.anchortype == "absolute":
(12)                        anchor = AbsoluteAnchor()
(12)                        anchor.pos.x = pixels_to_EMU(self.left)
(12)                        anchor.pos.y = pixels_to_EMU(self.top)
(8)                     elif self.anchortype == "oneCell":
(12)                        anchor = OneCellAnchor()
(12)                        anchor._from.col = self.anchorcol
(12)                        anchor._from.row = self.anchorrow
(8)                     anchor.ext.width = pixels_to_EMU(self._width)
(8)                     anchor.ext.height = pixels_to_EMU(self._height)
(8)                     return anchor
```

File 76 - graphic.py:

```
(0)             from openpyxl.xml.constants import CHART_NS, DRAWING_NS
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 Typed,
(4)                 Bool,
(4)                 String,
(4)                 Alias,
(0)             )
(0)             from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
(0)             from .effect import (
(4)                 EffectList,
(4)                 EffectContainer,
(0)             )
(0)             from .fill import (
(4)                 Blip,
(4)                 GradientFillProperties,
(4)                 BlipFillProperties,
(0)             )
(0)             from .picture import PictureFrame
(0)             from .properties import (
(4)                 NonVisualDrawingProps,
(4)                 NonVisualGroupShape,
23: (4)                     GroupShapeProperties,
24: (0)                 )
25: (0)                 from .relation import ChartRelation
26: (0)                 from .xdr import XDRTransform2D
27: (0)                 class GraphicFrameLocking(Serialisable):
28: (4)                     noGrp = Bool(allow_none=True)
29: (4)                     noDrilldown = Bool(allow_none=True)
30: (4)                     noSelect = Bool(allow_none=True)
31: (4)                     noChangeAspect = Bool(allow_none=True)
32: (4)                     noMove = Bool(allow_none=True)
33: (4)                     noResize = Bool(allow_none=True)
34: (4)                     extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
35: (4)                     def __init__(self,
36: (17)                                 noGrp=None,
37: (17)                                 noDrilldown=None,
38: (17)                                 noSelect=None,
39: (17)                                 noChangeAspect=None,
40: (17)                                 noMove=None,
41: (17)                                 noResize=None,
42: (17)                                 extLst=None,
43: (16)                                ):
44: (8)                         self.noGrp = noGrp
45: (8)                         self.noDrilldown = noDrilldown
46: (8)                         self.noSelect = noSelect
47: (8)                         self.noChangeAspect = noChangeAspect
48: (8)                         self.noMove = noMove
49: (8)                         self.noResize = noResize
50: (8)                         self.extLst = extLst
51: (0)                 class NonVisualGraphicFrameProperties(Serialisable):
52: (4)                     tagname = "cNvGraphicFramePr"
53: (4)                     graphicFrameLocks = Typed(expected_type=GraphicFrameLocking,
allow_none=True)
54: (4)                     extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
55: (4)                     def __init__(self,
56: (17)                                 graphicFrameLocks=None,
57: (17)                                 extLst=None,
58: (16)                                ):
59: (8)                         self.graphicFrameLocks = graphicFrameLocks
60: (8)                         self.extLst = extLst
61: (0)                 class NonVisualGraphicFrame(Serialisable):
62: (4)                     tagname = "nvGraphicFramePr"
63: (4)                     cNvPr = Typed(expected_type=NonVisualDrawingProps)
64: (4)                     cNvGraphicFramePr = Typed(expected_type=NonVisualGraphicFrameProperties)
65: (4)                     __elements__ = ('cNvPr', 'cNvGraphicFramePr')
66: (4)                     def __init__(self,
67: (17)                                 cNvPr=None,
68: (17)                                 cNvGraphicFramePr=None,
69: (16)                                ):
70: (8)                         if cNvPr is None:
71: (12)                            cNvPr = NonVisualDrawingProps(id=0, name="Chart 0")
72: (8)                         self.cNvPr = cNvPr
73: (8)                         if cNvGraphicFramePr is None:
74: (12)                            cNvGraphicFramePr = NonVisualGraphicFrameProperties()
75: (8)                         self.cNvGraphicFramePr = cNvGraphicFramePr
76: (0)                 class GraphicData(Serialisable):
77: (4)                     tagname = "graphicData"
78: (4)                     namespace = DRAWING_NS
79: (4)                     uri = String()
80: (4)                     chart = Typed(expected_type=ChartRelation, allow_none=True)
81: (4)                     def __init__(self,
82: (17)                                 uri=CHART_NS,
83: (17)                                 chart=None,
84: (16)                                ):
85: (8)                         self.uri = uri
86: (8)                         self.chart = chart
87: (0)                 class GraphicObject(Serialisable):
88: (4)                     tagname = "graphic"
89: (4)                     namespace = DRAWING_NS
90: (4)                     graphicData = Typed(expected_type=GraphicData)
```

```
(4)              def __init__(self,
(17)                          graphicData=None,
(16)                         ):
(8)                  if graphicData is None:
(12)                     graphicData = GraphicData()
(8)                  self.graphicData = graphicData
(0)          class GraphicFrame(Serialisable):
(4)              tagname = "graphicFrame"
(4)              nvGraphicFramePr = Typed(expected_type=NonVisualGraphicFrame)
(4)              xfrm = Typed(expected_type=XDRTransform2D)
(4)              graphic = Typed(expected_type=GraphicObject)
(4)              macro = String(allow_none=True)
(4)              fPublished = Bool(allow_none=True)
(4)              __elements__ = ('nvGraphicFramePr', 'xfrm', 'graphic', 'macro', 'fPublished')
(4)              def __init__(self,
(17)                          nvGraphicFramePr=None,
(17)                          xfrm=None,
(17)                          graphic=None,
(17)                          macro=None,
(17)                          fPublished=None,
(17)                          ):
(8)                  if nvGraphicFramePr is None:
(12)                     nvGraphicFramePr = NonVisualGraphicFrame()
(8)                  self.nvGraphicFramePr = nvGraphicFramePr
(8)                  if xfrm is None:
(12)                     xfrm = XDRTransform2D()
(8)                  self.xfrm = xfrm
(8)                  if graphic is None:
(12)                     graphic = GraphicObject()
(8)                  self.graphic = graphic
(8)                  self.macro = macro
(8)                  self.fPublished = fPublished
(0)          class GroupShape(Serialisable):
(4)              nvGrpSpPr = Typed(expected_type=NonVisualGroupShape)
(4)              nonVisualProperties = Alias("nvGrpSpPr")
(4)              grpSpPr = Typed(expected_type=GroupShapeProperties)
(4)              visualProperties = Alias("grpSpPr")
(4)              pic = Typed(expected_type=PictureFrame, allow_none=True)
(4)              __elements__ = ["nvGrpSpPr", "grpSpPr", "pic"]
(4)              def __init__(self,
(17)                          nvGrpSpPr=None,
(17)                          grpSpPr=None,
(17)                          pic=None,
(16)                         ):
(8)                  self.nvGrpSpPr = nvGrpSpPr
(8)                  self.grpSpPr = grpSpPr
(8)                  self.pic = pic

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

File 77 - picture.py:

(0)          from openpyxl.xml.constants import DRAWING_NS
(0)          from openpyxl.descriptors.serialisable import Serialisable
(0)          from openpyxl.descriptors import (
(4)              Typed,
(4)              Bool,
(4)              String,
(4)              Alias,
(0)          )
(0)          from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
(0)          from openpyxl.chart.shapes import GraphicalProperties
(0)          from .fill import BlipFillProperties
(0)          from .properties import NonVisualDrawingProps
(0)          from .geometry import ShapeStyle
(0)          class PictureLocking(Serialisable):
(4)              tagname = "picLocks"
(4)              namespace = DRAWING_NS
```

```
17: (4)                 noCrop = Bool(allow_none=True)
18: (4)                 noGrp = Bool(allow_none=True)
19: (4)                 noSelect = Bool(allow_none=True)
20: (4)                 noRot = Bool(allow_none=True)
21: (4)                 noChangeAspect = Bool(allow_none=True)
22: (4)                 noMove = Bool(allow_none=True)
23: (4)                 noResize = Bool(allow_none=True)
24: (4)                 noEditPoints = Bool(allow_none=True)
25: (4)                 noAdjustHandles = Bool(allow_none=True)
26: (4)                 noChangeArrowheads = Bool(allow_none=True)
27: (4)                 noChangeShapeType = Bool(allow_none=True)
28: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
29: (4)                 __elements__ = ()
30: (4)                 def __init__(self,
31: (17)                             noCrop=None,
32: (17)                             noGrp=None,
33: (17)                             noSelect=None,
34: (17)                             noRot=None,
35: (17)                             noChangeAspect=None,
36: (17)                             noMove=None,
37: (17)                             noResize=None,
38: (17)                             noEditPoints=None,
39: (17)                             noAdjustHandles=None,
40: (17)                             noChangeArrowheads=None,
41: (17)                             noChangeShapeType=None,
42: (17)                             extLst=None,
43: (16)                            ):
44: (8)                     self.noCrop = noCrop
45: (8)                     self.noGrp = noGrp
46: (8)                     self.noSelect = noSelect
47: (8)                     self.noRot = noRot
48: (8)                     self.noChangeAspect = noChangeAspect
49: (8)                     self.noMove = noMove
50: (8)                     self.noResize = noResize
51: (8)                     self.noEditPoints = noEditPoints
52: (8)                     self.noAdjustHandles = noAdjustHandles
53: (8)                     self.noChangeArrowheads = noChangeArrowheads
54: (8)                     self.noChangeShapeType = noChangeShapeType
55: (0)             class NonVisualPictureProperties(Serialisable):
56: (4)                 tagname = "cNvPicPr"
57: (4)                 preferRelativeResize = Bool(allow_none=True)
58: (4)                 picLocks = Typed(expected_type=PictureLocking, allow_none=True)
59: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
60: (4)                 __elements__ = ("picLocks",)
61: (4)                 def __init__(self,
62: (17)                             preferRelativeResize=None,
63: (17)                             picLocks=None,
64: (17)                             extLst=None,
65: (16)                            ):
66: (8)                     self.preferRelativeResize = preferRelativeResize
67: (8)                     self.picLocks = picLocks
68: (0)             class PictureNonVisual(Serialisable):
69: (4)                 tagname = "nvPicPr"
70: (4)                 cNvPr = Typed(expected_type=NonVisualDrawingProps, )
71: (4)                 cNvPicPr = Typed(expected_type=NonVisualPictureProperties, )
72: (4)                 __elements__ = ("cNvPr", "cNvPicPr")
73: (4)                 def __init__(self,
74: (17)                             cNvPr=None,
75: (17)                             cNvPicPr=None,
76: (16)                            ):
77: (8)                     if cNvPr is None:
78: (12)                        cNvPr = NonVisualDrawingProps(id=0, name="Image 1", descr="Name of
file")
79: (8)                     self.cNvPr = cNvPr
80: (8)                     if cNvPicPr is None:
81: (12)                        cNvPicPr = NonVisualPictureProperties()
82: (8)                     self.cNvPicPr = cNvPicPr
83: (0)             class PictureFrame(Serialisable):
84: (4)                 tagname = "pic"
```

```
85: (4)                 macro = String(allow_none=True)
86: (4)                 fPublished = Bool(allow_none=True)
87: (4)                 nvPicPr = Typed(expected_type=PictureNonVisual, )
88: (4)                 blipFill = Typed(expected_type=BlipFillProperties, )
89: (4)                 spPr = Typed(expected_type=GraphicalProperties, )
90: (4)                 graphicalProperties = Alias('spPr')
91: (4)                 style = Typed(expected_type=ShapeStyle, allow_none=True)
92: (4)                 __elements__ = ("nvPicPr", "blipFill", "spPr", "style")
93: (4)                 def __init__(self,
94: (17)                             macro=None,
95: (17)                             fPublished=None,
96: (17)                             nvPicPr=None,
97: (17)                             blipFill=None,
98: (17)                             spPr=None,
99: (17)                             style=None,
100: (16)                           ):
101: (8)                    self.macro = macro
102: (8)                    self.fPublished = fPublished
103: (8)                    if nvPicPr is None:
104: (12)                       nvPicPr = PictureNonVisual()
105: (8)                    self.nvPicPr = nvPicPr
106: (8)                    if blipFill is None:
107: (12)                       blipFill = BlipFillProperties()
108: (8)                    self.blipFill = blipFill
109: (8)                    if spPr is None:
110: (12)                       spPr = GraphicalProperties()
111: (8)                    self.spPr = spPr
112: (8)                    self.style = style

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

File 78 - sequence.py:

1: (0)              from openpyxl.compat import safe_string
2: (0)              from openpyxl.xml.functions import Element
3: (0)              from openpyxl.utils.indexed_list import IndexedList
4: (0)              from .base import Descriptor, Alias, _convert
5: (0)              from .namespace import namespaced
6: (0)              class Sequence(Descriptor):
7: (4)                  """
8: (4)                  A sequence (list or tuple) that may only contain objects of the declared
9: (4)                  type
10: (4)                 """
11: (4)                 expected_type = type(None)
12: (4)                 seq_types = (list, tuple)
13: (4)                 idx_base = 0
14: (4)                 unique = False
15: (4)                 container = list
16: (4)                 def __set__(self, instance, seq):
17: (8)                     if not isinstance(seq, self.seq_types):
18: (12)                        raise TypeError("Value must be a sequence")
19: (8)                     seq = self.container(_convert(self.expected_type, value) for value in
seq)
20: (8)                     if self.unique:
21: (12)                        seq = IndexedList(seq)
22: (8)                     super().__set__(instance, seq)
23: (4)                 def to_tree(self, tagname, obj, namespace=None):
24: (8)                     """
25: (8)                     Convert the sequence represented by the descriptor to an XML element
26: (8)                     """
27: (8)                     for idx, v in enumerate(obj, self.idx_base):
28: (12)                        if hasattr(v, "to_tree"):
29: (16)                            el = v.to_tree(tagname, idx)
30: (12)                        else:
31: (16)                            tagname = namespaced(obj, tagname, namespace)
32: (16)                            el = Element(tagname)
33: (16)                            el.text = safe_string(v)
34: (12)                        yield el
35: (0)             class UniqueSequence(Sequence):
```

```
(4)             """
(4)             Use a set to keep values unique
(4)             """
(4)             seq_types = (list, tuple, set)
(4)             container = set
(0)         class ValueSequence(Sequence):
(4)             """
(4)             A sequence of primitive types that are stored as a single attribute.
(4)             "val" is the default attribute
(4)             """
(4)             attribute = "val"
(4)             def to_tree(self, tagname, obj, namespace=None):
(8)                 tagname = namespaced(self, tagname, namespace)
(8)                 for v in obj:
(12)                    yield Element(tagname, {self.attribute:safe_string(v)})
(4)             def from_tree(self, node):
(8)                 return node.get(self.attribute)
(0)         class NestedSequence(Sequence):
(4)             """
(4)             Wrap a sequence in an containing object
(4)             """
(4)             count = False
(4)             def to_tree(self, tagname, obj, namespace=None):
(8)                 tagname = namespaced(self, tagname, namespace)
(8)                 container = Element(tagname)
(8)                 if self.count:
(12)                    container.set('count', str(len(obj)))
(8)                 for v in obj:
(12)                    container.append(v.to_tree())
(8)                 return container
(4)             def from_tree(self, node):
(8)                 return [self.expected_type.from_tree(el) for el in node]
(0)         class MultiSequence(Sequence):
(4)             """
(4)             Sequences can contain objects with different tags
(4)             """
(4)             def __set__(self, instance, seq):
(8)                 if not isinstance(seq, (tuple, list)):
(12)                    raise ValueError("Value must be a sequence")
(8)                 seq = list(seq)
(8)                 Descriptor.__set__(self, instance, seq)
(4)             def to_tree(self, tagname, obj, namespace=None):
(8)                 """
(8)                 Convert the sequence represented by the descriptor to an XML element
(8)                 """
(8)                 for v in obj:
(12)                    el = v.to_tree(namespace=namespace)
(12)                    yield el
(0)         class MultiSequencePart(Alias):
(4)             """
(4)             Allow a multisequence to be built up from parts
(4)             Excluded from the instance __elements__ or __attrs__ as is effectively an
Alias
(4)             """
(4)             def __init__(self, expected_type, store):
(8)                 self.expected_type = expected_type
(8)                 self.store = store
(4)             def __set__(self, instance, value):
(8)                 value = _convert(self.expected_type, value)
(8)                 instance.__dict__[self.store].append(value)
(4)             def __get__(self, instance, cls):
(8)                 return self
```

File 79 - geometry.py:

```
(0)         from openpyxl.descriptors.serialisable import Serialisable
(0)         from openpyxl.descriptors import (
(4)                 Typed,
(4)                 Float,
(4)                 Integer,
(4)                 Bool,
(4)                 MinMax,
(4)                 Set,
(4)                 NoneSet,
(4)                 String,
(4)                 Alias,
(0)             )
(0)             from openpyxl.descriptors.excel import Coordinate, Percentage
(0)             from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
(0)             from .line import LineProperties
(0)             from openpyxl.styles.colors import Color
(0)             from openpyxl.xml.constants import DRAWING_NS
(0)             class Point2D(Serialisable):
(4)                 tagname = "off"
(4)                 namespace = DRAWING_NS
(4)                 x = Coordinate()
(4)                 y = Coordinate()
(4)                 def __init__(self,
(17)                             x=None,
(17)                             y=None,
(16)                            ):
(8)                     self.x = x
(8)                     self.y = y
(0)             class PositiveSize2D(Serialisable):
(4)                 tagname = "ext"
(4)                 namespace = DRAWING_NS
(4)                 """
(4)                 Dimensions in EMUs
(4)                 """
(4)                 cx = Integer()
(4)                 width = Alias('cx')
(4)                 cy = Integer()
(4)                 height = Alias('cy')
(4)                 def __init__(self,
(17)                             cx=None,
(17)                             cy=None,
(16)                            ):
(8)                     self.cx = cx
(8)                     self.cy = cy
(0)             class Transform2D(Serialisable):
(4)                 tagname = "xfrm"
(4)                 namespace = DRAWING_NS
(4)                 rot = Integer(allow_none=True)
(4)                 flipH = Bool(allow_none=True)
(4)                 flipV = Bool(allow_none=True)
(4)                 off = Typed(expected_type=Point2D, allow_none=True)
(4)                 ext = Typed(expected_type=PositiveSize2D, allow_none=True)
(4)                 chOff = Typed(expected_type=Point2D, allow_none=True)
(4)                 chExt = Typed(expected_type=PositiveSize2D, allow_none=True)
(4)                 __elements__ = ('off', 'ext', 'chOff', 'chExt')
(4)                 def __init__(self,
(17)                             rot=None,
(17)                             flipH=None,
(17)                             flipV=None,
(17)                             off=None,
(17)                             ext=None,
(17)                             chOff=None,
(17)                             chExt=None,
(16)                            ):
(8)                     self.rot = rot
(8)                     self.flipH = flipH
(8)                     self.flipV = flipV
(8)                     self.off = off
(8)                     self.ext = ext
(8)                     self.chOff = chOff
(8)                     self.chExt = chExt
```

```
72: (0)             class GroupTransform2D(Serialisable):
73: (4)                 tagname = "xfrm"
74: (4)                 namespace = DRAWING_NS
75: (4)                 rot = Integer(allow_none=True)
76: (4)                 flipH = Bool(allow_none=True)
77: (4)                 flipV = Bool(allow_none=True)
78: (4)                 off = Typed(expected_type=Point2D, allow_none=True)
79: (4)                 ext = Typed(expected_type=PositiveSize2D, allow_none=True)
80: (4)                 chOff = Typed(expected_type=Point2D, allow_none=True)
81: (4)                 chExt = Typed(expected_type=PositiveSize2D, allow_none=True)
82: (4)                 __elements__ = ("off", "ext", "chOff", "chExt")
83: (4)                 def __init__(self,
84: (17)                             rot=0,
85: (17)                             flipH=None,
86: (17)                             flipV=None,
87: (17)                             off=None,
88: (17)                             ext=None,
89: (17)                             chOff=None,
90: (17)                             chExt=None,
91: (16)                            ):
92: (8)                     self.rot = rot
93: (8)                     self.flipH = flipH
94: (8)                     self.flipV = flipV
95: (8)                     self.off = off
96: (8)                     self.ext = ext
97: (8)                     self.chOff = chOff
98: (8)                     self.chExt = chExt
99: (0)             class SphereCoords(Serialisable):
100: (4)                tagname = "sphereCoords" # usually
101: (4)                lat = Integer()
102: (4)                lon = Integer()
103: (4)                rev = Integer()
104: (4)                def __init__(self,
105: (17)                            lat=None,
106: (17)                            lon=None,
107: (17)                            rev=None,
108: (16)                           ):
109: (8)                    self.lat = lat
110: (8)                    self.lon = lon
111: (8)                    self.rev = rev
112: (0)            class Camera(Serialisable):
113: (4)                tagname = "camera"
114: (4)                prst = Set(values=[
115: (8)                    'legacyObliqueTopLeft', 'legacyObliqueTop', 'legacyObliqueTopRight',
'legacyObliqueLeft',
116: (9)                     'legacyObliqueFront', 'legacyObliqueRight',
'legacyObliqueBottomLeft',
117: (9)                     'legacyObliqueBottom', 'legacyObliqueBottomRight',
'legacyPerspectiveTopLeft',
118: (9)                     'legacyPerspectiveTop', 'legacyPerspectiveTopRight',
'legacyPerspectiveLeft',
119: (9)                     'legacyPerspectiveFront', 'legacyPerspectiveRight',
'legacyPerspectiveBottomLeft',
120: (9)                     'legacyPerspectiveBottom', 'legacyPerspectiveBottomRight',
'orthographicFront',
121: (9)                     'isometricTopUp', 'isometricTopDown', 'isometricBottomUp',
'isometricBottomDown',
122: (9)                     'isometricLeftUp', 'isometricLeftDown', 'isometricRightUp',
'isometricRightDown',
123: (9)                     'isometricOffAxis1Left', 'isometricOffAxis1Right',
'isometricOffAxis1Top',
124: (9)                     'isometricOffAxis2Left', 'isometricOffAxis2Right',
'isometricOffAxis2Top',
125: (9)                     'isometricOffAxis3Left', 'isometricOffAxis3Right',
'isometricOffAxis3Bottom',
126: (9)                     'isometricOffAxis4Left', 'isometricOffAxis4Right',
'isometricOffAxis4Bottom',
127: (9)                     'obliqueTopLeft',  'obliqueTop', 'obliqueTopRight', 'obliqueLeft',
'obliqueRight',
128: (9)                     'obliqueBottomLeft', 'obliqueBottom', 'obliqueBottomRight',
'perspectiveFront',
129: (9)                     'perspectiveLeft', 'perspectiveRight', 'perspectiveAbove',
'perspectiveBelow',
130: (9)                     'perspectiveAboveLeftFacing', 'perspectiveAboveRightFacing',
131: (9)                     'perspectiveContrastingLeftFacing',
'perspectiveContrastingRightFacing',
132: (9)                     'perspectiveHeroicLeftFacing', 'perspectiveHeroicRightFacing',
133: (9)                     'perspectiveHeroicExtremeLeftFacing',
'perspectiveHeroicExtremeRightFacing',
134: (9)                     'perspectiveRelaxed', 'perspectiveRelaxedModerately'])
135: (4)                fov = Integer(allow_none=True)
136: (4)                zoom = Typed(expected_type=Percentage, allow_none=True)
137: (4)                rot = Typed(expected_type=SphereCoords, allow_none=True)
138: (4)                def __init__(self,
139: (17)                            prst=None,
140: (17)                            fov=None,
141: (17)                            zoom=None,
142: (17)                            rot=None,
143: (16)                           ):
144: (8)                    self.prst = prst
145: (8)                    self.fov = fov
146: (8)                    self.zoom = zoom
147: (8)                    self.rot = rot
148: (0)            class LightRig(Serialisable):
149: (4)                tagname = "lightRig"
150: (4)                rig = Set(values=['legacyFlat1', 'legacyFlat2', 'legacyFlat3',
'legacyFlat4', 'legacyNormal1',
151: (9)                     'legacyNormal2', 'legacyNormal3', 'legacyNormal4', 'legacyHarsh1',
152: (9)                     'legacyHarsh2', 'legacyHarsh3', 'legacyHarsh4', 'threePt',
'balanced',
153: (9)                     'soft', 'harsh', 'flood', 'contrasting', 'morning', 'sunrise',
'sunset',
154: (9)                     'chilly', 'freezing', 'flat', 'twoPt', 'glow', 'brightRoom']
155: (4)                )
156: (4)                dir = Set(values=(['tl', 't', 'tr', 'l', 'r', 'bl', 'b', 'br']))
157: (4)                rot = Typed(expected_type=SphereCoords, allow_none=True)
158: (4)                def __init__(self,
159: (17)                            rig=None,
160: (17)                            dir=None,
161: (17)                            rot=None,
162: (16)                           ):
163: (8)                    self.rig = rig
164: (8)                    self.dir = dir
165: (8)                    self.rot = rot
166: (0)            class Vector3D(Serialisable):
167: (4)                tagname = "vector"
168: (4)                dx = Integer() # can be in or universl measure :-/
169: (4)                dy = Integer()
170: (4)                dz = Integer()
171: (4)                def __init__(self,
172: (17)                            dx=None,
173: (17)                            dy=None,
174: (17)                            dz=None,
175: (16)                           ):
176: (8)                    self.dx = dx
177: (8)                    self.dy = dy
178: (8)                    self.dz = dz
179: (0)            class Point3D(Serialisable):
180: (4)                tagname = "anchor"
181: (4)                x = Integer()
182: (4)                y = Integer()
183: (4)                z = Integer()
184: (4)                def __init__(self,
185: (17)                            x=None,
186: (17)                            y=None,
187: (17)                            z=None,
188: (16)                           ):
189: (8)                    self.x = x
```

```
(8)                 self.y = y
(8)                 self.z = z
(0)         class Backdrop(Serialisable):
(4)             anchor = Typed(expected_type=Point3D, )
(4)             norm = Typed(expected_type=Vector3D, )
(4)             up = Typed(expected_type=Vector3D, )
(4)             extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)             def __init__(self,
(17)                         anchor=None,
(17)                         norm=None,
(17)                         up=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.anchor = anchor
(8)                 self.norm = norm
(8)                 self.up = up
(8)                 self.extLst = extLst
(0)         class Scene3D(Serialisable):
(4)             camera = Typed(expected_type=Camera, )
(4)             lightRig = Typed(expected_type=LightRig, )
(4)             backdrop = Typed(expected_type=Backdrop, allow_none=True)
(4)             extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)             def __init__(self,
(17)                         camera=None,
(17)                         lightRig=None,
(17)                         backdrop=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.camera = camera
(8)                 self.lightRig = lightRig
(8)                 self.backdrop = backdrop
(8)                 self.extLst = extLst
(0)         class Bevel(Serialisable):
(4)             tagname = "bevel"
(4)             w = Integer()
(4)             h = Integer()
(4)             prst = NoneSet(values=
(15)                       ['relaxedInset', 'circle', 'slope', 'cross', 'angle',
(16)                        'softRound', 'convex', 'coolSlant', 'divot', 'riblet',
(17)                         'hardEdge', 'artDeco']
(15)                       )
(4)             def __init__(self,
(17)                         w=None,
(17)                         h=None,
(17)                         prst=None,
(16)                        ):
(8)                 self.w = w
(8)                 self.h = h
(8)                 self.prst = prst
(0)         class Shape3D(Serialisable):
(4)             namespace = DRAWING_NS
(4)             z = Typed(expected_type=Coordinate, allow_none=True)
(4)             extrusionH = Integer(allow_none=True)
(4)             contourW = Integer(allow_none=True)
(4)             prstMaterial = NoneSet(values=[
(8)                 'legacyMatte','legacyPlastic', 'legacyMetal', 'legacyWireframe', 'matte', 'plastic',
(8)                 'metal', 'warmMatte', 'translucentPowder', 'powder', 'dkEdge',
(8)                 'softEdge', 'clear', 'flat', 'softmetal']
(23)                               )
(4)             bevelT = Typed(expected_type=Bevel, allow_none=True)
(4)             bevelB = Typed(expected_type=Bevel, allow_none=True)
(4)             extrusionClr = Typed(expected_type=Color, allow_none=True)
(4)             contourClr = Typed(expected_type=Color, allow_none=True)
(4)             extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)             def __init__(self,
(17)                         z=None,
(17)                         extrusionH=None,
(17)                         contourW=None,
(17)                         prstMaterial=None,
(17)                         bevelT=None,
(17)                         bevelB=None,
(17)                         extrusionClr=None,
(17)                         contourClr=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.z = z
(8)                 self.extrusionH = extrusionH
(8)                 self.contourW = contourW
(8)                 self.prstMaterial = prstMaterial
(8)                 self.bevelT = bevelT
(8)                 self.bevelB = bevelB
(8)                 self.extrusionClr = extrusionClr
(8)                 self.contourClr = contourClr
(8)                 self.extLst = extLst
(0)         class Path2D(Serialisable):
(4)             w = Float()
(4)             h = Float()
(4)             fill = NoneSet(values=(['norm', 'lighten', 'lightenLess', 'darken', 'darkenLess']))
(4)             stroke = Bool(allow_none=True)
(4)             extrusionOk = Bool(allow_none=True)
(4)             def __init__(self,
(17)                         w=None,
(17)                         h=None,
(17)                         fill=None,
(17)                         stroke=None,
(17)                         extrusionOk=None,
(16)                        ):
(8)                 self.w = w
(8)                 self.h = h
(8)                 self.fill = fill
(8)                 self.stroke = stroke
(8)                 self.extrusionOk = extrusionOk
(0)         class Path2DList(Serialisable):
(4)             path = Typed(expected_type=Path2D, allow_none=True)
(4)             def __init__(self,
(17)                         path=None,
(16)                        ):
(8)                 self.path = path
(0)         class GeomRect(Serialisable):
(4)             l = Coordinate()
(4)             t = Coordinate()
(4)             r = Coordinate()
(4)             b = Coordinate()
(4)             def __init__(self,
(17)                         l=None,
(17)                         t=None,
(17)                         r=None,
(17)                         b=None,
(16)                        ):
(8)                 self.l = l
(8)                 self.t = t
(8)                 self.r = r
(8)                 self.b = b
(0)         class AdjPoint2D(Serialisable):
(4)             x = Coordinate()
(4)             y = Coordinate()
(4)             def __init__(self,
(17)                         x=None,
(17)                         y=None,
(16)                        ):
(8)                 self.x = x
(8)                 self.y = y
(0)         class ConnectionSite(Serialisable):
(4)             ang = MinMax(min=0, max=360) # guess work, can also be a name
(4)             pos = Typed(expected_type=AdjPoint2D, )
(4)             def __init__(self,
326: (17)                           ang=None,
327: (17)                           pos=None,
328: (16)                          ):
329: (8)                    self.ang = ang
330: (8)                    self.pos = pos
331: (0)            class ConnectionSiteList(Serialisable):
332: (4)                cxn = Typed(expected_type=ConnectionSite, allow_none=True)
333: (4)                def __init__(self,
334: (17)                           cxn=None,
335: (16)                          ):
336: (8)                    self.cxn = cxn
337: (0)            class AdjustHandleList(Serialisable):
338: (4)                pass
339: (0)            class GeomGuide(Serialisable):
340: (4)                name = String()
341: (4)                fmla = String()
342: (4)                def __init__(self,
343: (17)                           name=None,
344: (17)                           fmla=None,
345: (16)                          ):
346: (8)                    self.name = name
347: (8)                    self.fmla = fmla
348: (0)            class GeomGuideList(Serialisable):
349: (4)                gd = Typed(expected_type=GeomGuide, allow_none=True)
350: (4)                def __init__(self,
351: (17)                           gd=None,
352: (16)                          ):
353: (8)                    self.gd = gd
354: (0)            class CustomGeometry2D(Serialisable):
355: (4)                avLst = Typed(expected_type=GeomGuideList, allow_none=True)
356: (4)                gdLst = Typed(expected_type=GeomGuideList, allow_none=True)
357: (4)                ahLst = Typed(expected_type=AdjustHandleList, allow_none=True)
358: (4)                cxnLst = Typed(expected_type=ConnectionSiteList, allow_none=True)
359: (4)                pathLst = Typed(expected_type=Path2DList, )
360: (4)                def __init__(self,
361: (17)                           avLst=None,
362: (17)                           gdLst=None,
363: (17)                           ahLst=None,
364: (17)                           cxnLst=None,
365: (17)                           rect=None,
366: (17)                           pathLst=None,
367: (16)                          ):
368: (8)                    self.avLst = avLst
369: (8)                    self.gdLst = gdLst
370: (8)                    self.ahLst = ahLst
371: (8)                    self.cxnLst = cxnLst
372: (8)                    self.rect = None
373: (8)                    self.pathLst = pathLst
374: (0)            class PresetGeometry2D(Serialisable):
375: (4)                namespace = DRAWING_NS
376: (4)                prst = Set(values=(
377: (8)                    ['line', 'lineInv', 'triangle', 'rtTriangle', 'rect',
378: (9)                     'diamond', 'parallelogram', 'trapezoid', 'nonIsoscelesTrapezoid',
379: (9)                     'pentagon', 'hexagon', 'heptagon', 'octagon', 'decagon', 'dodecagon',
380: (9)                     'star4', 'star5', 'star6', 'star7', 'star8', 'star10', 'star12',
381: (9)                     'star16', 'star24', 'star32', 'roundRect', 'round1Rect',
382: (9)                     'round2SameRect', 'round2DiagRect', 'snipRoundRect', 'snip1Rect',
383: (9)                     'snip2SameRect', 'snip2DiagRect', 'plaque', 'ellipse', 'teardrop',
384: (9)                     'homePlate', 'chevron', 'pieWedge', 'pie', 'blockArc', 'donut',
385: (9)                     'noSmoking', 'rightArrow', 'leftArrow', 'upArrow', 'downArrow',
386: (9)                     'stripedRightArrow', 'notchedRightArrow', 'bentUpArrow',
387: (9)                     'leftRightArrow', 'upDownArrow', 'leftUpArrow', 'leftRightUpArrow',
388: (9)                     'quadArrow', 'leftArrowCallout', 'rightArrowCallout',
'upArrowCallout',
389: (9)                     'downArrowCallout', 'leftRightArrowCallout', 'upDownArrowCallout',
390: (9)                     'quadArrowCallout', 'bentArrow', 'uturnArrow', 'circularArrow',
391: (9)                     'leftCircularArrow', 'leftRightCircularArrow', 'curvedRightArrow',
392: (9)                     'curvedLeftArrow', 'curvedUpArrow', 'curvedDownArrow', 'swooshArrow',
393: (9)                     'cube', 'can', 'lightningBolt', 'heart', 'sun', 'moon', 'smileyFace',
394: (9)                     'irregularSeal1', 'irregularSeal2', 'foldedCorner', 'bevel', 'frame',
395: (9)                     'halfFrame', 'corner', 'diagStripe', 'chord', 'arc', 'leftBracket',
396: (9)                     'rightBracket', 'leftBrace', 'rightBrace', 'bracketPair',
'bracePair',
397: (9)                     'straightConnector1', 'bentConnector2', 'bentConnector3',
398: (9)                     'bentConnector4', 'bentConnector5', 'curvedConnector2',
399: (9)                     'curvedConnector3', 'curvedConnector4', 'curvedConnector5',
'callout1',
400: (9)                     'callout2', 'callout3', 'accentCallout1', 'accentCallout2',
401: (9)                     'accentCallout3', 'borderCallout1', 'borderCallout2',
'borderCallout3',
402: (9)                     'accentBorderCallout1', 'accentBorderCallout2',
'accentBorderCallout3',
403: (9)                     'wedgeRectCallout', 'wedgeRoundRectCallout', 'wedgeEllipseCallout',
404: (9)                     'cloudCallout', 'cloud', 'ribbon', 'ribbon2', 'ellipseRibbon',
405: (9)                     'ellipseRibbon2', 'leftRightRibbon', 'verticalScroll',
406: (9)                     'horizontalScroll', 'wave', 'doubleWave', 'plus', 'flowChartProcess',
407: (9)                     'flowChartDecision', 'flowChartInputOutput',
408: (9)                     'flowChartPredefinedProcess', 'flowChartInternalStorage',
409: (9)                     'flowChartDocument', 'flowChartMultidocument', 'flowChartTerminator',
410: (9)                     'flowChartPreparation', 'flowChartManualInput',
411: (9)                     'flowChartManualOperation', 'flowChartConnector',
'flowChartPunchedCard',
412: (9)                     'flowChartPunchedTape', 'flowChartSummingJunction', 'flowChartOr',
413: (9)                     'flowChartCollate', 'flowChartSort', 'flowChartExtract',
414: (9)                     'flowChartMerge', 'flowChartOfflineStorage',
'flowChartOnlineStorage',
415: (9)                     'flowChartMagneticTape', 'flowChartMagneticDisk',
416: (9)                     'flowChartMagneticDrum', 'flowChartDisplay', 'flowChartDelay',
417: (9)                     'flowChartAlternateProcess', 'flowChartOffpageConnector',
418: (9)                     'actionButtonBlank', 'actionButtonHome', 'actionButtonHelp',
419: (9)                     'actionButtonInformation', 'actionButtonForwardNext',
420: (9)                     'actionButtonBackPrevious', 'actionButtonEnd',
'actionButtonBeginning',
421: (9)                     'actionButtonReturn', 'actionButtonDocument', 'actionButtonSound',
422: (9)                     'actionButtonMovie', 'gear6', 'gear9', 'funnel', 'mathPlus',
'mathMinus',
423: (9)                     'mathMultiply', 'mathDivide', 'mathEqual', 'mathNotEqual',
'cornerTabs',
424: (9)                     'squareTabs', 'plaqueTabs', 'chartX', 'chartStar', 'chartPlus']))
425: (4)                 avLst = Typed(expected_type=GeomGuideList, allow_none=True)
426: (4)                 def __init__(self,
427: (17)                             prst=None,
428: (17)                             avLst=None,
429: (16)                            ):
430: (8)                     self.prst = prst
431: (8)                     self.avLst = avLst
432: (0)             class FontReference(Serialisable):
433: (4)                 idx = NoneSet(values=(['major', 'minor']))
434: (4)                 def __init__(self,
435: (17)                             idx=None,
436: (16)                            ):
437: (8)                     self.idx = idx
438: (0)             class StyleMatrixReference(Serialisable):
439: (4)                 idx = Integer()
440: (4)                 def __init__(self,
441: (17)                             idx=None,
442: (16)                            ):
443: (8)                     self.idx = idx
444: (0)             class ShapeStyle(Serialisable):
445: (4)                 lnRef = Typed(expected_type=StyleMatrixReference, )
446: (4)                 fillRef = Typed(expected_type=StyleMatrixReference, )
447: (4)                 effectRef = Typed(expected_type=StyleMatrixReference, )
448: (4)                 fontRef = Typed(expected_type=FontReference, )
449: (4)                 def __init__(self,
450: (17)                             lnRef=None,
451: (17)                             fillRef=None,
452: (17)                             effectRef=None,
453: (17)                             fontRef=None,
                ):
        self.lnRef = lnRef
        self.fillRef = fillRef
        self.effectRef = effectRef
        self.fontRef = fontRef
```

File 80 - relation.py:

```
from openpyxl.xml.constants import CHART_NS
from openpyxl.descriptors.serialisable import Serialisable
from openpyxl.descriptors.excel import Relation
class ChartRelation(Serialisable):
    tagname = "chart"
    namespace = CHART_NS
    id = Relation()
    def __init__(self, id):
        self.id = id
```

File 81 - __init__.py:

```
from .drawing import Drawing
```

File 82 - __init__.py:

```
from .rule import Rule
```

File 83 - namespace.py:

```
def namespaced(obj, tagname, namespace=None):
    """
    Utility to create a namespaced tag for an object
    """
    namespace = getattr(obj, "namespace", None) or namespace
    if namespace is not None:
        tagname = "{%s}%s" % (namespace, tagname)
    return tagname
```

File 84 - connector.py:

```
from openpyxl.descriptors.serialisable import Serialisable
from openpyxl.descriptors import (
    Typed,
    Bool,
    Integer,
    String,
    Alias,
)
from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
from openpyxl.chart.shapes import GraphicalProperties
from openpyxl.chart.text import RichText
from .properties import (
    NonVisualDrawingProps,
    NonVisualDrawingShapeProps,
)
from .geometry import ShapeStyle
class Connection(Serialisable):
    id = Integer()
    idx = Integer()
    def __init__(self,
21: (17)                             id=None,
22: (17)                             idx=None,
23: (16)                            ):
24: (8)                     self.id = id
25: (8)                     self.idx = idx
26: (0)             class ConnectorLocking(Serialisable):
27: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
28: (4)                 def __init__(self,
29: (17)                             extLst=None,
30: (16)                            ):
31: (8)                     self.extLst = extLst
32: (0)             class NonVisualConnectorProperties(Serialisable):
33: (4)                 cxnSpLocks = Typed(expected_type=ConnectorLocking, allow_none=True)
34: (4)                 stCxn = Typed(expected_type=Connection, allow_none=True)
35: (4)                 endCxn = Typed(expected_type=Connection, allow_none=True)
36: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
37: (4)                 def __init__(self,
38: (17)                             cxnSpLocks=None,
39: (17)                             stCxn=None,
40: (17)                             endCxn=None,
41: (17)                             extLst=None,
42: (16)                            ):
43: (8)                     self.cxnSpLocks = cxnSpLocks
44: (8)                     self.stCxn = stCxn
45: (8)                     self.endCxn = endCxn
46: (8)                     self.extLst = extLst
47: (0)             class ConnectorNonVisual(Serialisable):
48: (4)                 cNvPr = Typed(expected_type=NonVisualDrawingProps, )
49: (4)                 cNvCxnSpPr = Typed(expected_type=NonVisualConnectorProperties, )
50: (4)                 __elements__ = ("cNvPr", "cNvCxnSpPr",)
51: (4)                 def __init__(self,
52: (17)                             cNvPr=None,
53: (17)                             cNvCxnSpPr=None,
54: (16)                            ):
55: (8)                     self.cNvPr = cNvPr
56: (8)                     self.cNvCxnSpPr = cNvCxnSpPr
57: (0)             class ConnectorShape(Serialisable):
58: (4)                 tagname = "cxnSp"
59: (4)                 nvCxnSpPr = Typed(expected_type=ConnectorNonVisual)
60: (4)                 spPr = Typed(expected_type=GraphicalProperties)
61: (4)                 style = Typed(expected_type=ShapeStyle, allow_none=True)
62: (4)                 macro = String(allow_none=True)
63: (4)                 fPublished = Bool(allow_none=True)
64: (4)                 def __init__(self,
65: (17)                             nvCxnSpPr=None,
66: (17)                             spPr=None,
67: (17)                             style=None,
68: (17)                             macro=None,
69: (17)                             fPublished=None,
70: (17)                             ):
71: (8)                     self.nvCxnSpPr = nvCxnSpPr
72: (8)                     self.spPr = spPr
73: (8)                     self.style = style
74: (8)                     self.macro = macro
75: (8)                     self.fPublished = fPublished
76: (0)             class ShapeMeta(Serialisable):
77: (4)                 tagname = "nvSpPr"
78: (4)                 cNvPr = Typed(expected_type=NonVisualDrawingProps)
79: (4)                 cNvSpPr = Typed(expected_type=NonVisualDrawingShapeProps)
80: (4)                 def __init__(self, cNvPr=None, cNvSpPr=None):
81: (8)                     self.cNvPr = cNvPr
82: (8)                     self.cNvSpPr = cNvSpPr
83: (0)             class Shape(Serialisable):
84: (4)                 macro = String(allow_none=True)
85: (4)                 textlink = String(allow_none=True)
86: (4)                 fPublished = Bool(allow_none=True)
87: (4)                 fLocksText = Bool(allow_none=True)
88: (4)                 nvSpPr = Typed(expected_type=ShapeMeta, allow_none=True)
89: (4)                 meta = Alias("nvSpPr")
```

```
90: (4)                 spPr = Typed(expected_type=GraphicalProperties)
91: (4)                 graphicalProperties = Alias("spPr")
92: (4)                 style = Typed(expected_type=ShapeStyle, allow_none=True)
93: (4)                 txBody = Typed(expected_type=RichText, allow_none=True)
94: (4)                 def __init__(self,
95: (17)                             macro=None,
96: (17)                             textlink=None,
97: (17)                             fPublished=None,
98: (17)                             fLocksText=None,
99: (17)                             nvSpPr=None,
100: (17)                            spPr=None,
101: (17)                            style=None,
102: (17)                            txBody=None,
103: (16)                           ):
104: (8)                     self.macro = macro
105: (8)                     self.textlink = textlink
106: (8)                     self.fPublished = fPublished
107: (8)                     self.fLocksText = fLocksText
108: (8)                     self.nvSpPr = nvSpPr
109: (8)                     self.spPr = spPr
110: (8)                     self.style = style
111: (8)                     self.txBody = txBody

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

File 85 - properties.py:

1: (0)              from openpyxl.xml.constants import DRAWING_NS
2: (0)              from openpyxl.descriptors.serialisable import Serialisable
3: (0)              from openpyxl.descriptors import (
4: (4)                  Typed,
5: (4)                  Bool,
6: (4)                  Integer,
7: (4)                  Set,
8: (4)                  String,
9: (4)                  Alias,
10: (4)                 NoneSet,
11: (0)             )
12: (0)             from openpyxl.descriptors.excel import ExtensionList as OfficeArtExtensionList
13: (0)             from .geometry import GroupTransform2D, Scene3D
14: (0)             from .text import Hyperlink
15: (0)             class GroupShapeProperties(Serialisable):
16: (4)                 tagname = "grpSpPr"
17: (4)                 bwMode = NoneSet(values=(['clr', 'auto', 'gray', 'ltGray', 'invGray',
18: (26)                                          'grayWhite', 'blackGray', 'blackWhite', 'black',
'white', 'hidden']))
19: (4)                 xfrm = Typed(expected_type=GroupTransform2D, allow_none=True)
20: (4)                 scene3d = Typed(expected_type=Scene3D, allow_none=True)
21: (4)                 extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
22: (4)                 def __init__(self,
23: (17)                             bwMode=None,
24: (17)                             xfrm=None,
25: (17)                             scene3d=None,
26: (17)                             extLst=None,
27: (16)                            ):
28: (8)                     self.bwMode = bwMode
29: (8)                     self.xfrm = xfrm
30: (8)                     self.scene3d = scene3d
31: (8)                     self.extLst = extLst
32: (0)             class GroupLocking(Serialisable):
33: (4)                 tagname = "grpSpLocks"
34: (4)                 namespace = DRAWING_NS
35: (4)                 noGrp = Bool(allow_none=True)
36: (4)                 noUngrp = Bool(allow_none=True)
37: (4)                 noSelect = Bool(allow_none=True)
38: (4)                 noRot = Bool(allow_none=True)
39: (4)                 noChangeAspect = Bool(allow_none=True)
40: (4)                 noMove = Bool(allow_none=True)
41: (4)                 noResize = Bool(allow_none=True)
```

```
(4)         noChangeArrowheads = Bool(allow_none=True)
(4)         noEditPoints = Bool(allow_none=True)
(4)         noAdjustHandles = Bool(allow_none=True)
(4)         noChangeArrowheads = Bool(allow_none=True)
(4)         noChangeShapeType = Bool(allow_none=True)
(4)         extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)         __elements__ = ()
(4)         def __init__(self,
(17)                     noGrp=None,
(17)                     noUngrp=None,
(17)                     noSelect=None,
(17)                     noRot=None,
(17)                     noChangeAspect=None,
(17)                     noChangeArrowheads=None,
(17)                     noMove=None,
(17)                     noResize=None,
(17)                     noEditPoints=None,
(17)                     noAdjustHandles=None,
(17)                     noChangeShapeType=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.noGrp = noGrp
(8)             self.noUngrp = noUngrp
(8)             self.noSelect = noSelect
(8)             self.noRot = noRot
(8)             self.noChangeAspect = noChangeAspect
(8)             self.noChangeArrowheads = noChangeArrowheads
(8)             self.noMove = noMove
(8)             self.noResize = noResize
(8)             self.noEditPoints = noEditPoints
(8)             self.noAdjustHandles = noAdjustHandles
(8)             self.noChangeShapeType = noChangeShapeType
(0)     class NonVisualGroupDrawingShapeProps(Serialisable):
(4)         tagname = "cNvGrpSpPr"
(4)         grpSpLocks = Typed(expected_type=GroupLocking, allow_none=True)
(4)         extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)         __elements__ = ("grpSpLocks",)
(4)         def __init__(self,
(17)                     grpSpLocks=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.grpSpLocks = grpSpLocks
(0)     class NonVisualDrawingShapeProps(Serialisable):
(4)         tagname = "cNvSpPr"
(4)         spLocks = Typed(expected_type=GroupLocking, allow_none=True)
(4)         txBax = Bool(allow_none=True)
(4)         extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)         __elements__ = ("spLocks", "txBax")
(4)         def __init__(self,
(17)                     spLocks=None,
(17)                     txBox=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.spLocks = spLocks
(8)             self.txBox = txBox
(0)     class NonVisualDrawingProps(Serialisable):
(4)         tagname = "cNvPr"
(4)         id = Integer()
(4)         name = String()
(4)         descr = String(allow_none=True)
(4)         hidden = Bool(allow_none=True)
(4)         title = String(allow_none=True)
(4)         hlinkClick = Typed(expected_type=Hyperlink, allow_none=True)
(4)         hlinkHover = Typed(expected_type=Hyperlink, allow_none=True)
(4)         extLst = Typed(expected_type=OfficeArtExtensionList, allow_none=True)
(4)         __elements__ = ["hlinkClick", "hlinkHover"]
(4)         def __init__(self,
(17)                     id=None,
(17)                     name=None,
(17)                         descr=None,
(17)                         hidden=None,
(17)                         title=None,
(17)                         hlinkClick=None,
(17)                         hlinkHover=None,
(17)                         extLst=None,
(16)                        ):
(8)                  self.id = id
(8)                  self.name = name
(8)                  self.descr = descr
(8)                  self.hidden = hidden
(8)                  self.title = title
(8)                  self.hlinkClick = hlinkClick
(8)                  self.hlinkHover = hlinkHover
(8)                  self.extLst = extLst
(0)          class NonVisualGroupShape(Serialisable):
(4)              tagname = "nvGrpSpPr"
(4)              cNvPr = Typed(expected_type=NonVisualDrawingProps)
(4)              cNvGrpSpPr = Typed(expected_type=NonVisualGroupDrawingShapeProps)
(4)              __elements__ = ("cNvPr", "cNvGrpSpPr")
(4)              def __init__(self,
(17)                         cNvPr=None,
(17)                         cNvGrpSpPr=None,
(16)                        ):
(8)                  self.cNvPr = cNvPr
(8)                  self.cNvGrpSpPr = cNvGrpSpPr
```

File 86 - serialisable.py:

```
(0)          from copy import copy
(0)          from keyword import kwlist
(0)          KEYWORDS = frozenset(kwlist)
(0)          from . import Descriptor
(0)          from . import MetaSerialisable
(0)          from .sequence import (
(4)              Sequence,
(4)              NestedSequence,
(4)              MultiSequencePart,
(0)          )
(0)          from .namespace import namespaced
(0)          from openpyxl.compat import safe_string
(0)          from openpyxl.xml.functions import (
(4)              Element,
(4)              localname,
(0)          )
(0)          seq_types = (list, tuple)
(0)          class Serialisable(metaclass=MetaSerialisable):
(4)              """
(4)              Objects can serialise to XML their attributes and child objects.
(4)              The following class attributes are created by the metaclass at runtime:
(4)              __attrs__ = attributes
(4)              __nested__ = single-valued child treated as an attribute
(4)              __elements__ = child elements
(4)              """
(4)              __attrs__ = None
(4)              __nested__ = None
(4)              __elements__ = None
(4)              __namespaced__ = None
(4)              idx_base = 0
(4)              @property
(4)              def tagname(self):
(8)                  raise(NotImplementedError)
(4)              namespace = None
(4)              @classmethod
(4)              def from_tree(cls, node):
(8)                  """
(8)                  Create object from XML
```

```
(8)             """
(8)             attrib = dict(node.attrib)
(8)             for key, ns in cls.__namespaced__:
(12)                if ns in attrib:
(16)                    attrib[key] = attrib[ns]
(16)                    del attrib[ns]
(8)             for key in list(attrib):
(12)                if key.startswith('{'):
(16)                    del attrib[key]
(12)                elif key in KEYWORDS:
(16)                    attrib["_" + key] = attrib[key]
(16)                    del attrib[key]
(12)                elif "-" in key:
(16)                    n = key.replace("-", "_")
(16)                    attrib[n] = attrib[key]
(16)                    del attrib[key]
(8)             if node.text and "attr_text" in cls.__attrs__:
(12)                attrib["attr_text"] = node.text
(8)             for el in node:
(12)                tag = localname(el)
(12)                if tag in KEYWORDS:
(16)                    tag = "_" + tag
(12)                desc = getattr(cls, tag, None)
(12)                if desc is None or isinstance(desc, property):
(16)                    continue
(12)                if hasattr(desc, 'from_tree'):
(16)                    obj = desc.from_tree(el)
(12)                else:
(16)                    if hasattr(desc.expected_type, "from_tree"):
(20)                        obj = desc.expected_type.from_tree(el)
(16)                    else:
(20)                        obj = el.text
(12)                if isinstance(desc, NestedSequence):
(16)                    attrib[tag] = obj
(12)                elif isinstance(desc, Sequence):
(16)                    attrib.setdefault(tag, [])
(16)                    attrib[tag].append(obj)
(12)                elif isinstance(desc, MultiSequencePart):
(16)                    attrib.setdefault(desc.store, [])
(16)                    attrib[desc.store].append(obj)
(12)                else:
(16)                    attrib[tag] = obj
(8)             return cls(**attrib)
(4)         def to_tree(self, tagname=None, idx=None, namespace=None):
(8)             if tagname is None:
(12)                tagname = self.tagname
(8)             if tagname.startswith("_"):
(12)                tagname = tagname[1:]
(8)             tagname = namespaced(self, tagname, namespace)
(8)             namespace = getattr(self, "namespace", namespace)
(8)             attrs = dict(self)
(8)             for key, ns in self.__namespaced__:
(12)                if key in attrs:
(16)                    attrs[ns] = attrs[key]
(16)                    del attrs[key]
(8)             el = Element(tagname, attrs)
(8)             if "attr_text" in self.__attrs__:
(12)                el.text = safe_string(getattr(self, "attr_text"))
(8)             for child_tag in self.__elements__:
(12)                desc = getattr(self.__class__, child_tag, None)
(12)                obj = getattr(self, child_tag)
(12)                if hasattr(desc, "namespace") and hasattr(obj, 'namespace'):
(16)                    obj.namespace = desc.namespace
(12)                if isinstance(obj, seq_types):
(16)                    if isinstance(desc, NestedSequence):
(20)                        if not obj:
(24)                            continue
(20)                        nodes = [desc.to_tree(child_tag, obj, namespace)]
(16)                    elif isinstance(desc, Sequence):
```

```
                    desc.idx_base = self.idx_base
                    nodes = (desc.to_tree(child_tag, obj, namespace))
                else: # property
                    nodes = (v.to_tree(child_tag, namespace) for v in obj)
                for node in nodes:
                    el.append(node)
            else:
                if child_tag in self.__nested__:
                    node = desc.to_tree(child_tag, obj, namespace)
                elif obj is None:
                    continue
                else:
                    node = obj.to_tree(child_tag)
                if node is not None:
                    el.append(node)
        return el
    def __iter__(self):
        for attr in self.__attrs__:
            value = getattr(self, attr)
            if attr.startswith("_"):
                attr = attr[1:]
            elif attr != "attr_text" and "_" in attr:
                desc = getattr(self.__class__, attr)
                if getattr(desc, "hyphenated", False):
                    attr = attr.replace("_", "-")
            if attr != "attr_text" and value is not None:
                yield attr, safe_string(value)
    def __eq__(self, other):
        if not self.__class__ == other.__class__:
            return False
        elif not dict(self) == dict(other):
            return False
        for el in self.__elements__:
            if getattr(self, el) != getattr(other, el):
                return False
        return True
    def __ne__(self, other):
        return not self == other
    def __repr__(self):
        s = u"<{0}.{1} object>\nParameters:".format(
            self.__module__,
            self.__class__.__name__
        )
        args = []
        for k in self.__attrs__ + self.__elements__:
            v = getattr(self, k)
            if isinstance(v, Descriptor):
                v = None
            args.append(u"{0}={1}".format(k, repr(v)))
        args = u", ".join(args)
        return u"\n".join([s, args])
    def __hash__(self):
        fields = []
        for attr in self.__attrs__ + self.__elements__:
            val = getattr(self, attr)
            if isinstance(val, list):
                val = tuple(val)
            fields.append(val)
        return hash(tuple(fields))
    def __add__(self, other):
        if type(self) != type(other):
            raise TypeError("Cannot combine instances of different types")
        vals = {}
        for attr in self.__attrs__:
            vals[attr] = getattr(self, attr) or getattr(other, attr)
        for el in self.__elements__:
            a = getattr(self, el)
            b = getattr(other, el)
            if a and b:
```

```
177: (16)                           vals[el] = a + b
178: (12)                       else:
179: (16)                           vals[el] = a or b
180: (8)                    return self.__class__(**vals)
181: (4)                def __copy__(self):
182: (8)                    xml = self.to_tree(tagname="dummy")
183: (8)                    cp = self.__class__.from_tree(xml)
184: (8)                    for k in self.__dict__:
185: (12)                       if k not in self.__attrs__ + self.__elements__:
186: (16)                           v = copy(getattr(self, k))
187: (16)                           setattr(cp, k, v)
188: (8)                    return cp

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

File 87 - spreadsheet_drawing.py:

1: (0)               from openpyxl.descriptors.serialisable import Serialisable
2: (0)               from openpyxl.descriptors import (
3: (4)                   Typed,
4: (4)                   Bool,
5: (4)                   NoneSet,
6: (4)                   Integer,
7: (4)                   Sequence,
8: (4)                   Alias,
9: (0)               )
10: (0)              from openpyxl.descriptors.nested import (
11: (4)                  NestedText,
12: (4)                  NestedNoneSet,
13: (0)              )
14: (0)              from openpyxl.descriptors.excel import Relation
15: (0)              from openpyxl.packaging.relationship import (
16: (4)                  Relationship,
17: (4)                  RelationshipList,
18: (0)              )
19: (0)              from openpyxl.utils import coordinate_to_tuple
20: (0)              from openpyxl.utils.units import (
21: (4)                  cm_to_EMU,
22: (4)                  pixels_to_EMU,
23: (0)              )
24: (0)              from openpyxl.drawing.image import Image
25: (0)              from openpyxl.xml.constants import SHEET_DRAWING_NS
26: (0)              from openpyxl.chart._chart import ChartBase
27: (0)              from .xdr import (
28: (4)                  XDRPoint2D,
29: (4)                  XDRPositiveSize2D,
30: (0)              )
31: (0)              from .fill import Blip
32: (0)              from .connector import Shape
33: (0)              from .graphic import (
34: (4)                  GroupShape,
35: (4)                  GraphicFrame,
36: (4)                  )
37: (0)              from .geometry import PresetGeometry2D
38: (0)              from .picture import PictureFrame
39: (0)              from .relation import ChartRelation
40: (0)              class AnchorClientData(Serialisable):
41: (4)                  fLocksWithSheet = Bool(allow_none=True)
42: (4)                  fPrintsWithSheet = Bool(allow_none=True)
43: (4)                  def __init__(self,
44: (17)                              fLocksWithSheet=None,
45: (17)                              fPrintsWithSheet=None,
46: (17)                              ):
47: (8)                      self.fLocksWithSheet = fLocksWithSheet
48: (8)                      self.fPrintsWithSheet = fPrintsWithSheet
49: (0)              class AnchorMarker(Serialisable):
50: (4)                  tagname = "marker"
51: (4)                  col = NestedText(expected_type=int)
52: (4)                  colOff = NestedText(expected_type=int)
```

```
53: (4)                     row = NestedText(expected_type=int)
54: (4)                     rowOff = NestedText(expected_type=int)
55: (4)                     def __init__(self,
56: (17)                                 col=0,
57: (17)                                 colOff=0,
58: (17)                                 row=0,
59: (17)                                 rowOff=0,
60: (17)                                 ):
61: (8)                         self.col = col
62: (8)                         self.colOff = colOff
63: (8)                         self.row = row
64: (8)                         self.rowOff = rowOff
65: (0)                 class _AnchorBase(Serialisable):
66: (4)                     sp = Typed(expected_type=Shape, allow_none=True)
67: (4)                     shape = Alias("sp")
68: (4)                     grpSp = Typed(expected_type=GroupShape, allow_none=True)
69: (4)                     groupShape = Alias("grpSp")
70: (4)                     graphicFrame = Typed(expected_type=GraphicFrame, allow_none=True)
71: (4)                     cxnSp = Typed(expected_type=Shape, allow_none=True)
72: (4)                     connectionShape = Alias("cxnSp")
73: (4)                     pic = Typed(expected_type=PictureFrame, allow_none=True)
74: (4)                     contentPart = Relation()
75: (4)                     clientData = Typed(expected_type=AnchorClientData)
76: (4)                     __elements__ = ('sp', 'grpSp', 'graphicFrame',
77: (20)                                    'cxnSp', 'pic', 'contentPart', 'clientData')
78: (4)                     def __init__(self,
79: (17)                                 clientData=None,
80: (17)                                 sp=None,
81: (17)                                 grpSp=None,
82: (17)                                 graphicFrame=None,
83: (17)                                 cxnSp=None,
84: (17)                                 pic=None,
85: (17)                                 contentPart=None
86: (17)                                 ):
87: (8)                         if clientData is None:
88: (12)                            clientData = AnchorClientData()
89: (8)                         self.clientData = clientData
90: (8)                         self.sp = sp
91: (8)                         self.grpSp = grpSp
92: (8)                         self.graphicFrame = graphicFrame
93: (8)                         self.cxnSp = cxnSp
94: (8)                         self.pic = pic
95: (8)                         self.contentPart = contentPart
96: (0)                 class AbsoluteAnchor(_AnchorBase):
97: (4)                     tagname = "absoluteAnchor"
98: (4)                     pos = Typed(expected_type=XDRPoint2D)
99: (4)                     ext = Typed(expected_type=XDRPositiveSize2D)
100: (4)                    sp = _AnchorBase.sp
101: (4)                    grpSp = _AnchorBase.grpSp
102: (4)                    graphicFrame = _AnchorBase.graphicFrame
103: (4)                    cxnSp = _AnchorBase.cxnSp
104: (4)                    pic = _AnchorBase.pic
105: (4)                    contentPart = _AnchorBase.contentPart
106: (4)                    clientData = _AnchorBase.clientData
107: (4)                    __elements__ = ('pos', 'ext') + _AnchorBase.__elements__
108: (4)                    def __init__(self,
109: (17)                                pos=None,
110: (17)                                ext=None,
111: (17)                                **kw
112: (16)                               ):
113: (8)                        if pos is None:
114: (12)                           pos = XDRPoint2D(0, 0)
115: (8)                        self.pos = pos
116: (8)                        if ext is None:
117: (12)                           ext = XDRPositiveSize2D(0, 0)
118: (8)                        self.ext = ext
119: (8)                        super().__init__(**kw)
120: (0)                class OneCellAnchor(_AnchorBase):
121: (4)                    tagname = "oneCellAnchor"
```

```
(4)             _from = Typed(expected_type=AnchorMarker)
(4)             ext = Typed(expected_type=XDRPositiveSize2D)
(4)             sp = _AnchorBase.sp
(4)             grpSp = _AnchorBase.grpSp
(4)             graphicFrame = _AnchorBase.graphicFrame
(4)             cxnSp = _AnchorBase.cxnSp
(4)             pic = _AnchorBase.pic
(4)             contentPart = _AnchorBase.contentPart
(4)             clientData = _AnchorBase.clientData
(4)             __elements__ = ('_from', 'ext') + _AnchorBase.__elements__
(4)             def __init__(self,
(17)                         _from=None,
(17)                         ext=None,
(17)                         **kw
(16)                        ):
(8)                 if _from is None:
(12)                    _from = AnchorMarker()
(8)                 self._from = _from
(8)                 if ext is None:
(12)                    ext = XDRPositiveSize2D(0, 0)
(8)                 self.ext = ext
(8)                 super().__init__(**kw)
(0)         class TwoCellAnchor(_AnchorBase):
(4)             tagname = "twoCellAnchor"
(4)             editAs = NoneSet(values=(['twoCell', 'oneCell', 'absolute']))
(4)             _from = Typed(expected_type=AnchorMarker)
(4)             to = Typed(expected_type=AnchorMarker)
(4)             sp = _AnchorBase.sp
(4)             grpSp = _AnchorBase.grpSp
(4)             graphicFrame = _AnchorBase.graphicFrame
(4)             cxnSp = _AnchorBase.cxnSp
(4)             pic = _AnchorBase.pic
(4)             contentPart = _AnchorBase.contentPart
(4)             clientData = _AnchorBase.clientData
(4)             __elements__ = ('_from', 'to') + _AnchorBase.__elements__
(4)             def __init__(self,
(17)                         editAs=None,
(17)                         _from=None,
(17)                         to=None,
(17)                         **kw
(17)                         ):
(8)                 self.editAs = editAs
(8)                 if _from is None:
(12)                    _from = AnchorMarker()
(8)                 self._from = _from
(8)                 if to is None:
(12)                    to = AnchorMarker()
(8)                 self.to = to
(8)                 super().__init__(**kw)
(0)         def _check_anchor(obj):
(4)             """
(4)             Check whether an object has an existing Anchor object
(4)             If not create a OneCellAnchor using the provided coordinate
(4)             """
(4)             anchor = obj.anchor
(4)             if not isinstance(anchor, _AnchorBase):
(8)                 row, col = coordinate_to_tuple(anchor.upper())
(8)                 anchor = OneCellAnchor()
(8)                 anchor._from.row = row -1
(8)                 anchor._from.col = col -1
(8)                 if isinstance(obj, ChartBase):
(12)                    anchor.ext.width = cm_to_EMU(obj.width)
(12)                    anchor.ext.height = cm_to_EMU(obj.height)
(8)                 elif isinstance(obj, Image):
(12)                    anchor.ext.width = pixels_to_EMU(obj.width)
(12)                    anchor.ext.height = pixels_to_EMU(obj.height)
(4)             return anchor
(0)         class SpreadsheetDrawing(Serialisable):
(4)             tagname = "wsDr"
```

```
(4)                 mime_type = "application/vnd.openxmlformats-officedocument.drawing+xml"
(4)                 _rel_type =
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/drawing"
(4)                 _path = PartName="/xl/drawings/drawing{0}.xml"
(4)                 _id = None
(4)                 twoCellAnchor = Sequence(expected_type=TwoCellAnchor, allow_none=True)
(4)                 oneCellAnchor = Sequence(expected_type=OneCellAnchor, allow_none=True)
(4)                 absoluteAnchor = Sequence(expected_type=AbsoluteAnchor, allow_none=True)
(4)                 __elements__ = ("twoCellAnchor", "oneCellAnchor", "absoluteAnchor")
(4)                 def __init__(self,
(17)                             twoCellAnchor=(),
(17)                             oneCellAnchor=(),
(17)                             absoluteAnchor=(),
(17)                             ):
(8)                     self.twoCellAnchor = twoCellAnchor
(8)                     self.oneCellAnchor = oneCellAnchor
(8)                     self.absoluteAnchor = absoluteAnchor
(8)                     self.charts = []
(8)                     self.images = []
(8)                     self._rels = []
(4)                 def __hash__(self):
(8)                     """
(8)                     Just need to check for identity
(8)                     """
(8)                     return id(self)
(4)                 def __bool__(self):
(8)                     return bool(self.charts) or bool(self.images)
(4)                 def _write(self):
(8)                     """
(8)                     create required structure and the serialise
(8)                     """
(8)                     anchors = []
(8)                     for idx, obj in enumerate(self.charts + self.images, 1):
(12)                        anchor = _check_anchor(obj)
(12)                        if isinstance(obj, ChartBase):
(16)                            rel = Relationship(type="chart", Target=obj.path)
(16)                            anchor.graphicFrame = self._chart_frame(idx)
(12)                        elif isinstance(obj, Image):
(16)                            rel = Relationship(type="image", Target=obj.path)
(16)                            child = anchor.pic or anchor.groupShape and
anchor.groupShape.pic
(16)                            if not child:
(20)                                anchor.pic = self._picture_frame(idx)
(16)                            else:
(20)                                child.blipFill.blip.embed = "rId{0}".format(idx)
(12)                        anchors.append(anchor)
(12)                        self._rels.append(rel)
(8)                     for a in anchors:
(12)                        if isinstance(a, OneCellAnchor):
(16)                            self.oneCellAnchor.append(a)
(12)                        elif isinstance(a, TwoCellAnchor):
(16)                            self.twoCellAnchor.append(a)
(12)                        else:
(16)                            self.absoluteAnchor.append(a)
(8)                     tree = self.to_tree()
(8)                     tree.set('xmlns', SHEET_DRAWING_NS)
(8)                     return tree
(4)                 def _chart_frame(self, idx):
(8)                     chart_rel = ChartRelation(f"rId{idx}")
(8)                     frame = GraphicFrame()
(8)                     nv = frame.nvGraphicFramePr.cNvPr
(8)                     nv.id = idx
(8)                     nv.name = "Chart {0}".format(idx)
(8)                     frame.graphic.graphicData.chart = chart_rel
(8)                     return frame
(4)                 def _picture_frame(self, idx):
(8)                     pic = PictureFrame()
(8)                     pic.nvPicPr.cNvPr.descr = "Picture"
(8)                     pic.nvPicPr.cNvPr.id = idx
```

```
258: (8)                    pic.nvPicPr.cNvPr.name = "Image {0}".format(idx)
259: (8)                    pic.blipFill.blip = Blip()
260: (8)                    pic.blipFill.blip.embed = "rId{0}".format(idx)
261: (8)                    pic.blipFill.blip.cstate = "print"
262: (8)                    pic.spPr.prstGeom = PresetGeometry2D(prst="rect")
263: (8)                    pic.spPr.ln = None
264: (8)                    return pic
265: (4)                def _write_rels(self):
266: (8)                    rels = RelationshipList()
267: (8)                    for r in self._rels:
268: (12)                       rels.append(r)
269: (8)                    return rels.to_tree()
270: (4)                @property
271: (4)                def path(self):
272: (8)                    return self._path.format(self._id)
273: (4)                @property
274: (4)                def _chart_rels(self):
275: (8)                    """
276: (8)                    Get relationship information for each chart and bind anchor to it
277: (8)                    """
278: (8)                    rels = []
279: (8)                    anchors = self.absoluteAnchor + self.oneCellAnchor +
self.twoCellAnchor
280: (8)                    for anchor in anchors:
281: (12)                       if anchor.graphicFrame is not None:
282: (16)                           graphic = anchor.graphicFrame.graphic
283: (16)                           rel = graphic.graphicData.chart
284: (16)                           if rel is not None:
285: (20)                               rel.anchor = anchor
286: (20)                               rel.anchor.graphicFrame = None
287: (20)                               rels.append(rel)
288: (8)                    return rels
289: (4)                @property
290: (4)                def _blip_rels(self):
291: (8)                    """
292: (8)                    Get relationship information for each blip and bind anchor to it
293: (8)                    Images that are not part of the XLSX package will be ignored.
294: (8)                    """
295: (8)                    rels = []
296: (8)                    anchors = self.absoluteAnchor + self.oneCellAnchor +
self.twoCellAnchor
297: (8)                    for anchor in anchors:
298: (12)                       child = anchor.pic or anchor.groupShape and anchor.groupShape.pic
299: (12)                       if child and child.blipFill:
300: (16)                           rel = child.blipFill.blip
301: (16)                           if rel is not None and rel.embed:
302: (20)                               rel.anchor = anchor
303: (20)                               rels.append(rel)
304: (8)                    return rels

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

File 88 - formatting.py:

1: (0)              from collections import OrderedDict
2: (0)              from openpyxl.descriptors import (
3: (4)                  Bool,
4: (4)                  Sequence,
5: (4)                  Alias,
6: (4)                  Convertible,
7: (0)              )
8: (0)              from openpyxl.descriptors.serialisable import Serialisable
9: (0)              from .rule import Rule
10: (0)             from openpyxl.worksheet.cell_range import MultiCellRange
11: (0)             class ConditionalFormatting(Serialisable):
12: (4)                 tagname = "conditionalFormatting"
13: (4)                 sqref = Convertible(expected_type=MultiCellRange)
14: (4)                 cells = Alias("sqref")
15: (4)                 pivot = Bool(allow_none=True)
```

```
    cfRule = Sequence(expected_type=Rule)
    rules = Alias("cfRule")
    def __init__(self, sqref=(), pivot=None, cfRule=(), extLst=None):
        self.sqref = sqref
        self.pivot = pivot
        self.cfRule = cfRule
    def __eq__(self, other):
        if not isinstance(other, self.__class__):
            return False
        return self.sqref == other.sqref
    def __hash__(self):
        return hash(self.sqref)
    def __repr__(self):
        return "<{cls} {cells}>".format(cls=self.__class__.__name__, cells=self.sqref)
    def __contains__(self, coord):
        """
        Check whether a certain cell is affected by the formatting
        """
        return coord in self.sqref
class ConditionalFormattingList:
    """Conditional formatting rules."""
    def __init__(self):
        self._cf_rules = OrderedDict()
        self.max_priority = 0
    def add(self, range_string, cfRule):
        """Add a rule such as ColorScaleRule, FormulaRule or CellIsRule
         The priority will be added automatically.
        """
        cf = range_string
        if isinstance(range_string, str):
            cf = ConditionalFormatting(range_string)
        if not isinstance(cfRule, Rule):
            raise ValueError("Only instances of openpyxl.formatting.rule.Rule may be added")
        rule = cfRule
        self.max_priority += 1
        if not rule.priority:
            rule.priority = self.max_priority
        self._cf_rules.setdefault(cf, []).append(rule)
    def __bool__(self):
        return bool(self._cf_rules)
    def __len__(self):
        return len(self._cf_rules)
    def __iter__(self):
        for cf, rules in self._cf_rules.items():
            cf.rules = rules
            yield cf
    def __getitem__(self, key):
        """
        Get the rules for a cell range
        """
        if isinstance(key, str):
            key = ConditionalFormatting(sqref=key)
        return self._cf_rules[key]
    def __delitem__(self, key):
        key = ConditionalFormatting(sqref=key)
        del self._cf_rules[key]
    def __setitem__(self, key, rule):
        """
        Add a rule for a cell range
        """
        self.add(key, rule)
```

File 89 - rule.py:

```
from openpyxl.descriptors.serialisable import Serialisable
```

```
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  String,
5: (4)                  Sequence,
6: (4)                  Bool,
7: (4)                  NoneSet,
8: (4)                  Set,
9: (4)                  Integer,
10: (4)                 Float,
11: (0)             )
12: (0)             from openpyxl.descriptors.excel import ExtensionList
13: (0)             from openpyxl.styles.colors import Color, ColorDescriptor
14: (0)             from openpyxl.styles.differential import DifferentialStyle
15: (0)             from openpyxl.utils.cell import COORD_RE
16: (0)             class ValueDescriptor(Float):
17: (4)                 """
18: (4)                 Expected type depends upon type attribute of parent :-(
19: (4)                 Most values should be numeric BUT they can also be cell references
20: (4)                 """
21: (4)                 def __set__(self, instance, value):
22: (8)                     ref = None
23: (8)                     if value is not None and isinstance(value, str):
24: (12)                        ref = COORD_RE.match(value)
25: (8)                     if instance.type == "formula" or ref:
26: (12)                        self.expected_type = str
27: (8)                     else:
28: (12)                        self.expected_type = float
29: (8)                     super().__set__(instance, value)
30: (0)             class FormatObject(Serialisable):
31: (4)                 tagname = "cfvo"
32: (4)                 type = Set(values=(['num', 'percent', 'max', 'min', 'formula',
'percentile']))
33: (4)                 val = ValueDescriptor(allow_none=True)
34: (4)                 gte = Bool(allow_none=True)
35: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
36: (4)                 __elements__ = ()
37: (4)                 def __init__(self,
38: (17)                             type,
39: (17)                             val=None,
40: (17)                             gte=None,
41: (17)                             extLst=None,
42: (16)                            ):
43: (8)                     self.type = type
44: (8)                     self.val = val
45: (8)                     self.gte = gte
46: (0)             class RuleType(Serialisable):
47: (4)                 cfvo = Sequence(expected_type=FormatObject)
48: (0)             class IconSet(RuleType):
49: (4)                 tagname = "iconSet"
50: (4)                 iconSet = NoneSet(values=(['3Arrows', '3ArrowsGray', '3Flags',
51: (27)                                       '3TrafficLights1', '3TrafficLights2', '3Signs',
'3Symbols', '3Symbols2',
52: (27)                                       '4Arrows', '4ArrowsGray', '4RedToBlack', '4Rating',
'4TrafficLights',
53: (27)                                       '5Arrows', '5ArrowsGray', '5Rating', '5Quarters']))
54: (4)                 showValue = Bool(allow_none=True)
55: (4)                 percent = Bool(allow_none=True)
56: (4)                 reverse = Bool(allow_none=True)
57: (4)                 __elements__ = ("cfvo",)
58: (4)                 def __init__(self,
59: (17)                             iconSet=None,
60: (17)                             showValue=None,
61: (17)                             percent=None,
62: (17)                             reverse=None,
63: (17)                             cfvo=None,
64: (16)                            ):
65: (8)                     self.iconSet = iconSet
66: (8)                     self.showValue = showValue
67: (8)                     self.percent = percent
```

```
68: (8)                     self.reverse = reverse
69: (8)                     self.cfvo = cfvo
70: (0)             class DataBar(RuleType):
71: (4)                 tagname = "dataBar"
72: (4)                 minLength = Integer(allow_none=True)
73: (4)                 maxLength = Integer(allow_none=True)
74: (4)                 showValue = Bool(allow_none=True)
75: (4)                 color = ColorDescriptor()
76: (4)                 __elements__ = ('cfvo', 'color')
77: (4)                 def __init__(self,
78: (17)                             minLength=None,
79: (17)                             maxLength=None,
80: (17)                             showValue=None,
81: (17)                             cfvo=None,
82: (17)                             color=None,
83: (16)                            ):
84: (8)                     self.minLength = minLength
85: (8)                     self.maxLength = maxLength
86: (8)                     self.showValue = showValue
87: (8)                     self.cfvo = cfvo
88: (8)                     self.color = color
89: (0)             class ColorScale(RuleType):
90: (4)                 tagname = "colorScale"
91: (4)                 color = Sequence(expected_type=Color)
92: (4)                 __elements__ = ('cfvo', 'color')
93: (4)                 def __init__(self,
94: (17)                             cfvo=None,
95: (17)                             color=None,
96: (16)                            ):
97: (8)                     self.cfvo = cfvo
98: (8)                     self.color = color
99: (0)             class Rule(Serialisable):
100: (4)                tagname = "cfRule"
101: (4)                type = Set(values=(['expression', 'cellIs', 'colorScale', 'dataBar',
102: (24)                                   'iconSet', 'top10', 'uniqueValues', 'duplicateValues',
'containsText',
103: (24)                                   'notContainsText', 'beginsWith', 'endsWith',
'containsBlanks',
104: (24)                                   'notContainsBlanks', 'containsErrors',
'notContainsErrors', 'timePeriod',
105: (24)                                   'aboveAverage']))
106: (4)                dxfId = Integer(allow_none=True)
107: (4)                priority = Integer()
108: (4)                stopIfTrue = Bool(allow_none=True)
109: (4)                aboveAverage = Bool(allow_none=True)
110: (4)                percent = Bool(allow_none=True)
111: (4)                bottom = Bool(allow_none=True)
112: (4)                operator = NoneSet(values=(['lessThan', 'lessThanOrEqual', 'equal',
113: (28)                                       'notEqual', 'greaterThanOrEqual', 'greaterThan',
'between', 'notBetween',
114: (28)                                       'containsText', 'notContains', 'beginsWith',
'endsWith']))
115: (4)                text = String(allow_none=True)
116: (4)                timePeriod = NoneSet(values=(['today', 'yesterday', 'tomorrow',
'last7Days',
117: (30)                                        'thisMonth', 'lastMonth', 'nextMonth',
'thisWeek', 'lastWeek',
118: (30)                                        'nextWeek']))
119: (4)                rank = Integer(allow_none=True)
120: (4)                stdDev = Integer(allow_none=True)
121: (4)                equalAverage = Bool(allow_none=True)
122: (4)                formula = Sequence(expected_type=str)
123: (4)                colorScale = Typed(expected_type=ColorScale, allow_none=True)
124: (4)                dataBar = Typed(expected_type=DataBar, allow_none=True)
125: (4)                iconSet = Typed(expected_type=IconSet, allow_none=True)
126: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
127: (4)                dxf = Typed(expected_type=DifferentialStyle, allow_none=True)
128: (4)                __elements__ = ('colorScale', 'dataBar', 'iconSet', 'formula')
129: (4)                __attrs__ = ('type', 'rank', 'priority', 'equalAverage', 'operator',
130: (17)                           'aboveAverage', 'dxfId', 'stdDev', 'stopIfTrue',
'timePeriod', 'text',
131: (17)                           'percent', 'bottom')
132: (4)                    def __init__(self,
133: (17)                           type,
134: (17)                           dxfId=None,
135: (17)                           priority=0,
136: (17)                           stopIfTrue=None,
137: (17)                           aboveAverage=None,
138: (17)                           percent=None,
139: (17)                           bottom=None,
140: (17)                           operator=None,
141: (17)                           text=None,
142: (17)                           timePeriod=None,
143: (17)                           rank=None,
144: (17)                           stdDev=None,
145: (17)                           equalAverage=None,
146: (17)                           formula=(),
147: (17)                           colorScale=None,
148: (17)                           dataBar=None,
149: (17)                           iconSet=None,
150: (17)                           extLst=None,
151: (17)                           dxf=None,
152: (16)                          ):
153: (8)                        self.type = type
154: (8)                        self.dxfId = dxfId
155: (8)                        self.priority = priority
156: (8)                        self.stopIfTrue = stopIfTrue
157: (8)                        self.aboveAverage = aboveAverage
158: (8)                        self.percent = percent
159: (8)                        self.bottom = bottom
160: (8)                        self.operator = operator
161: (8)                        self.text = text
162: (8)                        self.timePeriod = timePeriod
163: (8)                        self.rank = rank
164: (8)                        self.stdDev = stdDev
165: (8)                        self.equalAverage = equalAverage
166: (8)                        self.formula = formula
167: (8)                        self.colorScale = colorScale
168: (8)                        self.dataBar = dataBar
169: (8)                        self.iconSet = iconSet
170: (8)                        self.dxf = dxf
171: (0)                def ColorScaleRule(start_type=None,
172: (17)                           start_value=None,
173: (17)                           start_color=None,
174: (17)                           mid_type=None,
175: (17)                           mid_value=None,
176: (17)                           mid_color=None,
177: (17)                           end_type=None,
178: (17)                           end_value=None,
179: (17)                           end_color=None):
180: (4)                    """Backwards compatibility"""
181: (4)                    formats = []
182: (4)                    if start_type is not None:
183: (8)                        formats.append(FormatObject(type=start_type, val=start_value))
184: (4)                    if mid_type is not None:
185: (8)                        formats.append(FormatObject(type=mid_type, val=mid_value))
186: (4)                    if end_type is not None:
187: (8)                        formats.append(FormatObject(type=end_type, val=end_value))
188: (4)                    colors = []
189: (4)                    for v in (start_color, mid_color, end_color):
190: (8)                        if v is not None:
191: (12)                           if not isinstance(v, Color):
192: (16)                               v = Color(v)
193: (12)                           colors.append(v)
194: (4)                    cs = ColorScale(cfvo=formats, color=colors)
195: (4)                    rule = Rule(type="colorScale", colorScale=cs)
196: (4)                    return rule
197: (0)                def FormulaRule(formula=None, stopIfTrue=None, font=None, border=None,
198: (16)                       fill=None):
199: (4)                """
200: (4)                Conditional formatting with custom differential style
201: (4)                """
202: (4)                rule = Rule(type="expression", formula=formula, stopIfTrue=stopIfTrue)
203: (4)                rule.dxf =  DifferentialStyle(font=font, border=border, fill=fill)
204: (4)                return rule
205: (0)            def CellIsRule(operator=None, formula=None, stopIfTrue=None, font=None,
border=None, fill=None):
206: (4)                """
207: (4)                Conditional formatting rule based on cell contents.
208: (4)                """
209: (4)                expand = {">": "greaterThan", ">=": "greaterThanOrEqual", "<": "lessThan",
"<=": "lessThanOrEqual",
210: (14)                     "=": "equal", "==": "equal", "!=": "notEqual"}
211: (4)                operator = expand.get(operator, operator)
212: (4)                rule = Rule(type='cellIs', operator=operator, formula=formula,
stopIfTrue=stopIfTrue)
213: (4)                rule.dxf = DifferentialStyle(font=font, border=border, fill=fill)
214: (4)                return rule
215: (0)            def IconSetRule(icon_style=None, type=None, values=None, showValue=None,
percent=None, reverse=None):
216: (4)                """
217: (4)                Convenience function for creating icon set rules
218: (4)                """
219: (4)                cfvo = []
220: (4)                for val in values:
221: (8)                    cfvo.append(FormatObject(type, val))
222: (4)                icon_set = IconSet(iconSet=icon_style, cfvo=cfvo, showValue=showValue,
223: (23)                                   percent=percent, reverse=reverse)
224: (4)                rule = Rule(type='iconSet', iconSet=icon_set)
225: (4)                return rule
226: (0)            def DataBarRule(start_type=None, start_value=None, end_type=None,
227: (16)                           end_value=None, color=None, showValue=None, minLength=None,
maxLength=None):
228: (4)                start = FormatObject(start_type, start_value)
229: (4)                end = FormatObject(end_type, end_value)
230: (4)                data_bar = DataBar(cfvo=[start, end], color=color, showValue=showValue,
231: (23)                                   minLength=minLength, maxLength=maxLength)
232: (4)                rule = Rule(type='dataBar', dataBar=data_bar)
233: (4)                return rule

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

File 90 - core.py:

1: (0)              import datetime
2: (0)              from openpyxl.descriptors import (
3: (4)                  DateTime,
4: (4)                  Alias,
5: (0)              )
6: (0)              from openpyxl.descriptors.serialisable import Serialisable
7: (0)              from openpyxl.descriptors.nested import NestedText
8: (0)              from openpyxl.xml.functions import (
9: (4)                  Element,
10: (4)                 QName,
11: (0)             )
12: (0)             from openpyxl.xml.constants import (
13: (4)                 COREPROPS_NS,
14: (4)                 DCORE_NS,
15: (4)                 XSI_NS,
16: (4)                 DCTERMS_NS,
17: (0)             )
18: (0)             class NestedDateTime(DateTime, NestedText):
19: (4)                 expected_type = datetime.datetime
20: (4)                 def to_tree(self, tagname=None, value=None, namespace=None):
21: (8)                     namespace = getattr(self, "namespace", namespace)
22: (8)                     if namespace is not None:
23: (12)                        tagname = "{%s}%s" % (namespace, tagname)
```

```
24: (8)                     el = Element(tagname)
25: (8)                     if value is not None:
26: (12)                        value = value.replace(tzinfo=None)
27: (12)                        el.text = value.isoformat(timespec="seconds") + 'Z'
28: (12)                        return el
29: (0)             class QualifiedDateTime(NestedDateTime):
30: (4)                 """In certain situations Excel will complain if the additional type
31: (4)                 attribute isn't set"""
32: (4)                 def to_tree(self, tagname=None, value=None, namespace=None):
33: (8)                     el = super().to_tree(tagname, value, namespace)
34: (8)                     el.set("{%s}type" % XSI_NS, QName(DCTERMS_NS, "W3CDTF"))
35: (8)                     return el
36: (0)             class DocumentProperties(Serialisable):
37: (4)                 """High-level properties of the document.
38: (4)                 Defined in ECMA-376 Par2 Annex D
39: (4)                 """
40: (4)                 tagname = "coreProperties"
41: (4)                 namespace = COREPROPS_NS
42: (4)                 category = NestedText(expected_type=str, allow_none=True)
43: (4)                 contentStatus = NestedText(expected_type=str, allow_none=True)
44: (4)                 keywords = NestedText(expected_type=str, allow_none=True)
45: (4)                 lastModifiedBy = NestedText(expected_type=str, allow_none=True)
46: (4)                 lastPrinted = NestedDateTime(allow_none=True)
47: (4)                 revision = NestedText(expected_type=str, allow_none=True)
48: (4)                 version = NestedText(expected_type=str, allow_none=True)
49: (4)                 last_modified_by = Alias("lastModifiedBy")
50: (4)                 subject = NestedText(expected_type=str, allow_none=True,
namespace=DCORE_NS)
51: (4)                 title = NestedText(expected_type=str, allow_none=True, namespace=DCORE_NS)
52: (4)                 creator = NestedText(expected_type=str, allow_none=True,
namespace=DCORE_NS)
53: (4)                 description = NestedText(expected_type=str, allow_none=True,
namespace=DCORE_NS)
54: (4)                 identifier = NestedText(expected_type=str, allow_none=True,
namespace=DCORE_NS)
55: (4)                 language = NestedText(expected_type=str, allow_none=True,
namespace=DCORE_NS)
56: (4)                 created = QualifiedDateTime(allow_none=True, namespace=DCTERMS_NS) #
assumed to be UTC
57: (4)                 modified = QualifiedDateTime(allow_none=True, namespace=DCTERMS_NS) #
assumed to be UTC
58: (4)                 __elements__ = ("creator", "title", "description", "subject","identifier",
59: (20)                                "language", "created", "modified", "lastModifiedBy",
"category",
60: (20)                                "contentStatus", "version", "revision", "keywords",
"lastPrinted",
61: (20)                                )
62: (4)                 def __init__(self,
63: (17)                             category=None,
64: (17)                             contentStatus=None,
65: (17)                             keywords=None,
66: (17)                             lastModifiedBy=None,
67: (17)                             lastPrinted=None,
68: (17)                             revision=None,
69: (17)                             version=None,
70: (17)                             created=None,
71: (17)                             creator="openpyxl",
72: (17)                             description=None,
73: (17)                             identifier=None,
74: (17)                             language=None,
75: (17)                             modified=None,
76: (17)                             subject=None,
77: (17)                             title=None,
78: (17)                             ):
79: (8)                     now =
datetime.datetime.now(tz=datetime.timezone.utc).replace(tzinfo=None)
80: (8)                     self.contentStatus = contentStatus
81: (8)                     self.lastPrinted = lastPrinted
82: (8)                     self.revision = revision
```

```
83: (8)                 self.version = version
84: (8)                 self.creator = creator
85: (8)                 self.lastModifiedBy = lastModifiedBy
86: (8)                 self.modified = modified or now
87: (8)                 self.created = created or now
88: (8)                 self.title = title
89: (8)                 self.subject = subject
90: (8)                 self.description = description
91: (8)                 self.identifier = identifier
92: (8)                 self.language = language
93: (8)                 self.keywords = keywords
94: (8)                 self.category = category

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

File 91 - cache.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Bool,
5: (4)                  Float,
6: (4)                  Set,
7: (4)                  NoneSet,
8: (4)                  String,
9: (4)                  Integer,
10: (4)                 DateTime,
11: (4)                 Sequence,
12: (0)             )
13: (0)             from openpyxl.descriptors.excel import (
14: (4)                 HexBinary,
15: (4)                 ExtensionList,
16: (4)                 Relation,
17: (0)             )
18: (0)             from openpyxl.descriptors.nested import NestedInteger
19: (0)             from openpyxl.descriptors.sequence import (
20: (4)                 NestedSequence,
21: (4)                 MultiSequence,
22: (4)                 MultiSequencePart,
23: (0)             )
24: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
25: (0)             from openpyxl.xml.functions import tostring
26: (0)             from openpyxl.packaging.relationship import (
27: (4)                 RelationshipList,
28: (4)                 Relationship,
29: (4)                 get_rels_path
30: (0)             )
31: (0)             from .table import (
32: (4)                 PivotArea,
33: (4)                 Reference,
34: (0)             )
35: (0)             from .fields import (
36: (4)                 Boolean,
37: (4)                 Error,
38: (4)                 Missing,
39: (4)                 Number,
40: (4)                 Text,
41: (4)                 TupleList,
42: (4)                 DateTimeField,
43: (0)             )
44: (0)             class MeasureDimensionMap(Serialisable):
45: (4)                 tagname = "map"
46: (4)                 measureGroup = Integer(allow_none=True)
47: (4)                 dimension = Integer(allow_none=True)
48: (4)                 def __init__(self,
49: (17)                             measureGroup=None,
50: (17)                             dimension=None,
51: (16)                            ):
52: (8)                     self.measureGroup = measureGroup
```

```
(8)             self.dimension = dimension
(0)     class MeasureGroup(Serialisable):
(4)         tagname = "measureGroup"
(4)         name = String()
(4)         caption = String()
(4)         def __init__(self,
(17)                     name=None,
(17)                     caption=None,
(16)                    ):
(8)             self.name = name
(8)             self.caption = caption
(0)     class PivotDimension(Serialisable):
(4)         tagname = "dimension"
(4)         measure = Bool()
(4)         name = String()
(4)         uniqueName = String()
(4)         caption = String()
(4)         def __init__(self,
(17)                     measure=None,
(17)                     name=None,
(17)                     uniqueName=None,
(17)                     caption=None,
(16)                    ):
(8)             self.measure = measure
(8)             self.name = name
(8)             self.uniqueName = uniqueName
(8)             self.caption = caption
(0)     class CalculatedMember(Serialisable):
(4)         tagname = "calculatedMember"
(4)         name = String()
(4)         mdx = String()
(4)         memberName = String(allow_none=True)
(4)         hierarchy = String(allow_none=True)
(4)         parent = String(allow_none=True)
(4)         solveOrder = Integer(allow_none=True)
(4)         set = Bool()
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         __elements__ = ()
(4)         def __init__(self,
(17)                     name=None,
(17)                     mdx=None,
(17)                     memberName=None,
(17)                     hierarchy=None,
(17)                     parent=None,
(17)                     solveOrder=None,
(17)                     set=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.name = name
(8)             self.mdx = mdx
(8)             self.memberName = memberName
(8)             self.hierarchy = hierarchy
(8)             self.parent = parent
(8)             self.solveOrder = solveOrder
(8)             self.set = set
(0)     class CalculatedItem(Serialisable):
(4)         tagname = "calculatedItem"
(4)         field = Integer(allow_none=True)
(4)         formula = String()
(4)         pivotArea = Typed(expected_type=PivotArea, )
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         __elements__ = ('pivotArea', 'extLst')
(4)         def __init__(self,
(17)                     field=None,
(17)                     formula=None,
(17)                     pivotArea=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.field = field
```

```
122: (8)                    self.formula = formula
123: (8)                    self.pivotArea = pivotArea
124: (8)                    self.extLst = extLst
125: (0)            class ServerFormat(Serialisable):
126: (4)                tagname = "serverFormat"
127: (4)                culture = String(allow_none=True)
128: (4)                format = String(allow_none=True)
129: (4)                def __init__(self,
130: (17)                            culture=None,
131: (17)                            format=None,
132: (16)                           ):
133: (8)                    self.culture = culture
134: (8)                    self.format = format
135: (0)            class Query(Serialisable):
136: (4)                tagname = "query"
137: (4)                mdx = String()
138: (4)                tpls = Typed(expected_type=TupleList, allow_none=True)
139: (4)                __elements__ = ('tpls',)
140: (4)                def __init__(self,
141: (17)                            mdx=None,
142: (17)                            tpls=None,
143: (16)                           ):
144: (8)                    self.mdx = mdx
145: (8)                    self.tpls = tpls
146: (0)            class OLAPSet(Serialisable):
147: (4)                tagname = "set"
148: (4)                count = Integer()
149: (4)                maxRank = Integer()
150: (4)                setDefinition = String()
151: (4)                sortType = NoneSet(values=(['ascending', 'descending', 'ascendingAlpha',
152: (32)                                           'descendingAlpha', 'ascendingNatural',
'descendingNatural']))
153: (4)                queryFailed = Bool()
154: (4)                tpls = Typed(expected_type=TupleList, allow_none=True)
155: (4)                sortByTuple = Typed(expected_type=TupleList, allow_none=True)
156: (4)                __elements__ = ('tpls', 'sortByTuple')
157: (4)                def __init__(self,
158: (17)                            count=None,
159: (17)                            maxRank=None,
160: (17)                            setDefinition=None,
161: (17)                            sortType=None,
162: (17)                            queryFailed=None,
163: (17)                            tpls=None,
164: (17)                            sortByTuple=None,
165: (16)                           ):
166: (8)                    self.count = count
167: (8)                    self.maxRank = maxRank
168: (8)                    self.setDefinition = setDefinition
169: (8)                    self.sortType = sortType
170: (8)                    self.queryFailed = queryFailed
171: (8)                    self.tpls = tpls
172: (8)                    self.sortByTuple = sortByTuple
173: (0)            class PCDSDTCEntries(Serialisable):
174: (4)                tagname = "entries"
175: (4)                count = Integer(allow_none=True)
176: (4)                m = Typed(expected_type=Missing, allow_none=True)
177: (4)                n = Typed(expected_type=Number, allow_none=True)
178: (4)                e = Typed(expected_type=Error, allow_none=True)
179: (4)                s = Typed(expected_type=Text, allow_none=True)
180: (4)                __elements__ = ('m', 'n', 'e', 's')
181: (4)                def __init__(self,
182: (17)                            count=None,
183: (17)                            m=None,
184: (17)                            n=None,
185: (17)                            e=None,
186: (17)                            s=None,
187: (16)                           ):
188: (8)                    self.count = count
189: (8)                    self.m = m
```

```
190: (8)                    self.n = n
191: (8)                    self.e = e
192: (8)                    self.s = s
193: (0)            class TupleCache(Serialisable):
194: (4)                tagname = "tupleCache"
195: (4)                entries = Typed(expected_type=PCDSDTCEntries, allow_none=True)
196: (4)                sets = NestedSequence(expected_type=OLAPSet, count=True)
197: (4)                queryCache = NestedSequence(expected_type=Query, count=True)
198: (4)                serverFormats = NestedSequence(expected_type=ServerFormat, count=True)
199: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
200: (4)                __elements__ = ('entries', 'sets', 'queryCache', 'serverFormats',
'extLst')
201: (4)                def __init__(self,
202: (17)                            entries=None,
203: (17)                            sets=(),
204: (17)                            queryCache=(),
205: (17)                            serverFormats=(),
206: (17)                            extLst=None,
207: (16)                           ):
208: (8)                    self.entries = entries
209: (8)                    self.sets = sets
210: (8)                    self.queryCache = queryCache
211: (8)                    self.serverFormats = serverFormats
212: (8)                    self.extLst = extLst
213: (0)            class OLAPKPI(Serialisable):
214: (4)                tagname = "kpi"
215: (4)                uniqueName = String()
216: (4)                caption = String(allow_none=True)
217: (4)                displayFolder = String(allow_none=True)
218: (4)                measureGroup = String(allow_none=True)
219: (4)                parent = String(allow_none=True)
220: (4)                value = String()
221: (4)                goal = String(allow_none=True)
222: (4)                status = String(allow_none=True)
223: (4)                trend = String(allow_none=True)
224: (4)                weight = String(allow_none=True)
225: (4)                time = String(allow_none=True)
226: (4)                def __init__(self,
227: (17)                            uniqueName=None,
228: (17)                            caption=None,
229: (17)                            displayFolder=None,
230: (17)                            measureGroup=None,
231: (17)                            parent=None,
232: (17)                            value=None,
233: (17)                            goal=None,
234: (17)                            status=None,
235: (17)                            trend=None,
236: (17)                            weight=None,
237: (17)                            time=None,
238: (16)                           ):
239: (8)                    self.uniqueName = uniqueName
240: (8)                    self.caption = caption
241: (8)                    self.displayFolder = displayFolder
242: (8)                    self.measureGroup = measureGroup
243: (8)                    self.parent = parent
244: (8)                    self.value = value
245: (8)                    self.goal = goal
246: (8)                    self.status = status
247: (8)                    self.trend = trend
248: (8)                    self.weight = weight
249: (8)                    self.time = time
250: (0)            class GroupMember(Serialisable):
251: (4)                tagname = "groupMember"
252: (4)                uniqueName = String()
253: (4)                group = Bool()
254: (4)                def __init__(self,
255: (17)                            uniqueName=None,
256: (17)                            group=None,
257: (16)                           ):
```

```
258: (8)                        self.uniqueName = uniqueName
259: (8)                        self.group = group
260: (0)                class LevelGroup(Serialisable):
261: (4)                    tagname = "group"
262: (4)                    name = String()
263: (4)                    uniqueName = String()
264: (4)                    caption = String()
265: (4)                    uniqueParent = String()
266: (4)                    id = Integer()
267: (4)                    groupMembers = NestedSequence(expected_type=GroupMember, count=True)
268: (4)                    __elements__ = ('groupMembers',)
269: (4)                    def __init__(self,
270: (17)                                name=None,
271: (17)                                uniqueName=None,
272: (17)                                caption=None,
273: (17)                                uniqueParent=None,
274: (17)                                id=None,
275: (17)                                groupMembers=(),
276: (16)                               ):
277: (8)                        self.name = name
278: (8)                        self.uniqueName = uniqueName
279: (8)                        self.caption = caption
280: (8)                        self.uniqueParent = uniqueParent
281: (8)                        self.id = id
282: (8)                        self.groupMembers = groupMembers
283: (0)                class GroupLevel(Serialisable):
284: (4)                    tagname = "groupLevel"
285: (4)                    uniqueName = String()
286: (4)                    caption = String()
287: (4)                    user = Bool()
288: (4)                    customRollUp = Bool()
289: (4)                    groups = NestedSequence(expected_type=LevelGroup, count=True)
290: (4)                    extLst = Typed(expected_type=ExtensionList, allow_none=True)
291: (4)                    __elements__ = ('groups', 'extLst')
292: (4)                    def __init__(self,
293: (17)                                uniqueName=None,
294: (17)                                caption=None,
295: (17)                                user=None,
296: (17)                                customRollUp=None,
297: (17)                                groups=(),
298: (17)                                extLst=None,
299: (16)                               ):
300: (8)                        self.uniqueName = uniqueName
301: (8)                        self.caption = caption
302: (8)                        self.user = user
303: (8)                        self.customRollUp = customRollUp
304: (8)                        self.groups = groups
305: (8)                        self.extLst = extLst
306: (0)                class FieldUsage(Serialisable):
307: (4)                    tagname = "fieldUsage"
308: (4)                    x = Integer()
309: (4)                    def __init__(self,
310: (17)                                x=None,
311: (16)                               ):
312: (8)                        self.x = x
313: (0)                class CacheHierarchy(Serialisable):
314: (4)                    tagname = "cacheHierarchy"
315: (4)                    uniqueName = String()
316: (4)                    caption = String(allow_none=True)
317: (4)                    measure = Bool()
318: (4)                    set = Bool()
319: (4)                    parentSet = Integer(allow_none=True)
320: (4)                    iconSet = Integer()
321: (4)                    attribute = Bool()
322: (4)                    time = Bool()
323: (4)                    keyAttribute = Bool()
324: (4)                    defaultMemberUniqueName = String(allow_none=True)
325: (4)                    allUniqueName = String(allow_none=True)
326: (4)                    allCaption = String(allow_none=True)
```

```
327: (4)                dimensionUniqueName = String(allow_none=True)
328: (4)                displayFolder = String(allow_none=True)
329: (4)                measureGroup = String(allow_none=True)
330: (4)                measures = Bool()
331: (4)                count = Integer()
332: (4)                oneField = Bool()
333: (4)                memberValueDatatype = Integer(allow_none=True)
334: (4)                unbalanced = Bool(allow_none=True)
335: (4)                unbalancedGroup = Bool(allow_none=True)
336: (4)                hidden = Bool()
337: (4)                fieldsUsage = NestedSequence(expected_type=FieldUsage, count=True)
338: (4)                groupLevels = NestedSequence(expected_type=GroupLevel, count=True)
339: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
340: (4)                __elements__ = ('fieldsUsage', 'groupLevels')
341: (4)                def __init__(self,
342: (17)                            uniqueName="",
343: (17)                            caption=None,
344: (17)                            measure=None,
345: (17)                            set=None,
346: (17)                            parentSet=None,
347: (17)                            iconSet=0,
348: (17)                            attribute=None,
349: (17)                            time=None,
350: (17)                            keyAttribute=None,
351: (17)                            defaultMemberUniqueName=None,
352: (17)                            allUniqueName=None,
353: (17)                            allCaption=None,
354: (17)                            dimensionUniqueName=None,
355: (17)                            displayFolder=None,
356: (17)                            measureGroup=None,
357: (17)                            measures=None,
358: (17)                            count=None,
359: (17)                            oneField=None,
360: (17)                            memberValueDatatype=None,
361: (17)                            unbalanced=None,
362: (17)                            unbalancedGroup=None,
363: (17)                            hidden=None,
364: (17)                            fieldsUsage=(),
365: (17)                            groupLevels=(),
366: (17)                            extLst=None,
367: (16)                           ):
368: (8)                     self.uniqueName = uniqueName
369: (8)                     self.caption = caption
370: (8)                     self.measure = measure
371: (8)                     self.set = set
372: (8)                     self.parentSet = parentSet
373: (8)                     self.iconSet = iconSet
374: (8)                     self.attribute = attribute
375: (8)                     self.time = time
376: (8)                     self.keyAttribute = keyAttribute
377: (8)                     self.defaultMemberUniqueName = defaultMemberUniqueName
378: (8)                     self.allUniqueName = allUniqueName
379: (8)                     self.allCaption = allCaption
380: (8)                     self.dimensionUniqueName = dimensionUniqueName
381: (8)                     self.displayFolder = displayFolder
382: (8)                     self.measureGroup = measureGroup
383: (8)                     self.measures = measures
384: (8)                     self.count = count
385: (8)                     self.oneField = oneField
386: (8)                     self.memberValueDatatype = memberValueDatatype
387: (8)                     self.unbalanced = unbalanced
388: (8)                     self.unbalancedGroup = unbalancedGroup
389: (8)                     self.hidden = hidden
390: (8)                     self.fieldsUsage = fieldsUsage
391: (8)                     self.groupLevels = groupLevels
392: (8)                     self.extLst = extLst
393: (0)            class GroupItems(Serialisable):
394: (4)                tagname = "groupItems"
395: (4)                m = Sequence(expected_type=Missing)
```

```
(4)        n = Sequence(expected_type=Number)
(4)        b = Sequence(expected_type=Boolean)
(4)        e = Sequence(expected_type=Error)
(4)        s = Sequence(expected_type=Text)
(4)        d = Sequence(expected_type=DateTimeField,)
(4)        __elements__ = ('m', 'n', 'b', 'e', 's', 'd')
(4)        __attrs__ = ("count", )
(4)        def __init__(self,
(17)                    count=None,
(17)                    m=(),
(17)                    n=(),
(17)                    b=(),
(17)                    e=(),
(17)                    s=(),
(17)                    d=(),
(16)                   ):
(8)            self.m = m
(8)            self.n = n
(8)            self.b = b
(8)            self.e = e
(8)            self.s = s
(8)            self.d = d
(4)        @property
(4)        def count(self):
(8)            return len(self.m + self.n + self.b + self.e + self.s + self.d)
(0)    class RangePr(Serialisable):
(4)        tagname = "rangePr"
(4)        autoStart = Bool(allow_none=True)
(4)        autoEnd = Bool(allow_none=True)
(4)        groupBy = NoneSet(values=(['range', 'seconds', 'minutes', 'hours', 'days',
(27)                              'months', 'quarters', 'years']))
(4)        startNum = Float(allow_none=True)
(4)        endNum = Float(allow_none=True)
(4)        startDate = DateTime(allow_none=True)
(4)        endDate = DateTime(allow_none=True)
(4)        groupInterval = Float(allow_none=True)
(4)        def __init__(self,
(17)                    autoStart=True,
(17)                    autoEnd=True,
(17)                    groupBy="range",
(17)                    startNum=None,
(17)                    endNum=None,
(17)                    startDate=None,
(17)                    endDate=None,
(17)                    groupInterval=1,
(16)                   ):
(8)            self.autoStart = autoStart
(8)            self.autoEnd = autoEnd
(8)            self.groupBy = groupBy
(8)            self.startNum = startNum
(8)            self.endNum = endNum
(8)            self.startDate = startDate
(8)            self.endDate = endDate
(8)            self.groupInterval = groupInterval
(0)    class FieldGroup(Serialisable):
(4)        tagname = "fieldGroup"
(4)        par = Integer(allow_none=True)
(4)        base = Integer(allow_none=True)
(4)        rangePr = Typed(expected_type=RangePr, allow_none=True)
(4)        discretePr = NestedSequence(expected_type=NestedInteger, count=True)
(4)        groupItems = Typed(expected_type=GroupItems, allow_none=True)
(4)        __elements__ = ('rangePr', 'discretePr', 'groupItems')
(4)        def __init__(self,
(17)                    par=None,
(17)                    base=None,
(17)                    rangePr=None,
(17)                    discretePr=(),
(17)                    groupItems=None,
(16)                   ):
```

```
465: (8)                    self.par = par
466: (8)                    self.base = base
467: (8)                    self.rangePr = rangePr
468: (8)                    self.discretePr = discretePr
469: (8)                    self.groupItems = groupItems
470: (0)            class SharedItems(Serialisable):
471: (4)                tagname = "sharedItems"
472: (4)                _fields = MultiSequence()
473: (4)                m = MultiSequencePart(expected_type=Missing, store="_fields")
474: (4)                n = MultiSequencePart(expected_type=Number, store="_fields")
475: (4)                b = MultiSequencePart(expected_type=Boolean, store="_fields")
476: (4)                e = MultiSequencePart(expected_type=Error, store="_fields")
477: (4)                s = MultiSequencePart(expected_type=Text,  store="_fields")
478: (4)                d = MultiSequencePart(expected_type=DateTimeField, store="_fields")
479: (4)                containsSemiMixedTypes = Bool(allow_none=True)
480: (4)                containsNonDate = Bool(allow_none=True)
481: (4)                containsDate = Bool(allow_none=True)
482: (4)                containsString = Bool(allow_none=True)
483: (4)                containsBlank = Bool(allow_none=True)
484: (4)                containsMixedTypes = Bool(allow_none=True)
485: (4)                containsNumber = Bool(allow_none=True)
486: (4)                containsInteger = Bool(allow_none=True)
487: (4)                minValue = Float(allow_none=True)
488: (4)                maxValue = Float(allow_none=True)
489: (4)                minDate = DateTime(allow_none=True)
490: (4)                maxDate = DateTime(allow_none=True)
491: (4)                longText = Bool(allow_none=True)
492: (4)                __attrs__ = ('count', 'containsBlank', 'containsDate', 'containsInteger',
493: (17)                            'containsMixedTypes', 'containsNonDate', 'containsNumber',
494: (17)                            'containsSemiMixedTypes', 'containsString', 'minValue',
'maxValue',
495: (17)                            'minDate', 'maxDate', 'longText')
496: (4)                def __init__(self,
497: (17)                            _fields=(),
498: (17)                            containsSemiMixedTypes=None,
499: (17)                            containsNonDate=None,
500: (17)                            containsDate=None,
501: (17)                            containsString=None,
502: (17)                            containsBlank=None,
503: (17)                            containsMixedTypes=None,
504: (17)                            containsNumber=None,
505: (17)                            containsInteger=None,
506: (17)                            minValue=None,
507: (17)                            maxValue=None,
508: (17)                            minDate=None,
509: (17)                            maxDate=None,
510: (17)                            count=None,
511: (17)                            longText=None,
512: (16)                           ):
513: (8)                    self._fields = _fields
514: (8)                    self.containsBlank = containsBlank
515: (8)                    self.containsDate = containsDate
516: (8)                    self.containsNonDate = containsNonDate
517: (8)                    self.containsString = containsString
518: (8)                    self.containsMixedTypes = containsMixedTypes
519: (8)                    self.containsSemiMixedTypes = containsSemiMixedTypes
520: (8)                    self.containsNumber = containsNumber
521: (8)                    self.containsInteger = containsInteger
522: (8)                    self.minValue = minValue
523: (8)                    self.maxValue = maxValue
524: (8)                    self.minDate = minDate
525: (8)                    self.maxDate = maxDate
526: (8)                    self.longText = longText
527: (4)                @property
528: (4)                def count(self):
529: (8)                    return len(self._fields)
530: (0)            class CacheField(Serialisable):
531: (4)                tagname = "cacheField"
532: (4)                sharedItems = Typed(expected_type=SharedItems, allow_none=True)
```

```
(4)                   fieldGroup = Typed(expected_type=FieldGroup, allow_none=True)
(4)                   mpMap = NestedInteger(allow_none=True, attribute="v")
(4)                   extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                   name = String()
(4)                   caption = String(allow_none=True)
(4)                   propertyName = String(allow_none=True)
(4)                   serverField = Bool(allow_none=True)
(4)                   uniqueList = Bool(allow_none=True)
(4)                   numFmtId = Integer(allow_none=True)
(4)                   formula = String(allow_none=True)
(4)                   sqlType = Integer(allow_none=True)
(4)                   hierarchy = Integer(allow_none=True)
(4)                   level = Integer(allow_none=True)
(4)                   databaseField = Bool(allow_none=True)
(4)                   mappingCount = Integer(allow_none=True)
(4)                   memberPropertyField = Bool(allow_none=True)
(4)                   __elements__ = ('sharedItems', 'fieldGroup', 'mpMap')
(4)                   def __init__(self,
(17)                               sharedItems=None,
(17)                               fieldGroup=None,
(17)                               mpMap=None,
(17)                               extLst=None,
(17)                               name=None,
(17)                               caption=None,
(17)                               propertyName=None,
(17)                               serverField=None,
(17)                               uniqueList=True,
(17)                               numFmtId=None,
(17)                               formula=None,
(17)                               sqlType=0,
(17)                               hierarchy=0,
(17)                               level=0,
(17)                               databaseField=True,
(17)                               mappingCount=None,
(17)                               memberPropertyField=None,
(16)                              ):
(8)                       self.sharedItems = sharedItems
(8)                       self.fieldGroup = fieldGroup
(8)                       self.mpMap = mpMap
(8)                       self.extLst = extLst
(8)                       self.name = name
(8)                       self.caption = caption
(8)                       self.propertyName = propertyName
(8)                       self.serverField = serverField
(8)                       self.uniqueList = uniqueList
(8)                       self.numFmtId = numFmtId
(8)                       self.formula = formula
(8)                       self.sqlType = sqlType
(8)                       self.hierarchy = hierarchy
(8)                       self.level = level
(8)                       self.databaseField = databaseField
(8)                       self.mappingCount = mappingCount
(8)                       self.memberPropertyField = memberPropertyField
(0)               class RangeSet(Serialisable):
(4)                   tagname = "rangeSet"
(4)                   i1 = Integer(allow_none=True)
(4)                   i2 = Integer(allow_none=True)
(4)                   i3 = Integer(allow_none=True)
(4)                   i4 = Integer(allow_none=True)
(4)                   ref = String()
(4)                   name = String(allow_none=True)
(4)                   sheet = String(allow_none=True)
(4)                   def __init__(self,
(17)                               i1=None,
(17)                               i2=None,
(17)                               i3=None,
(17)                               i4=None,
(17)                               ref=None,
(17)                               name=None,
602: (17)                           sheet=None,
603: (16)                          ):
604: (8)                    self.i1 = i1
605: (8)                    self.i2 = i2
606: (8)                    self.i3 = i3
607: (8)                    self.i4 = i4
608: (8)                    self.ref = ref
609: (8)                    self.name = name
610: (8)                    self.sheet = sheet
611: (0)            class PageItem(Serialisable):
612: (4)                tagname = "pageItem"
613: (4)                name = String()
614: (4)                def __init__(self,
615: (17)                            name=None,
616: (16)                           ):
617: (8)                    self.name = name
618: (0)            class Consolidation(Serialisable):
619: (4)                tagname = "consolidation"
620: (4)                autoPage = Bool(allow_none=True)
621: (4)                pages = NestedSequence(expected_type=PageItem, count=True)
622: (4)                rangeSets = NestedSequence(expected_type=RangeSet, count=True)
623: (4)                __elements__ = ('pages', 'rangeSets')
624: (4)                def __init__(self,
625: (17)                            autoPage=None,
626: (17)                            pages=(),
627: (17)                            rangeSets=(),
628: (16)                           ):
629: (8)                    self.autoPage = autoPage
630: (8)                    self.pages = pages
631: (8)                    self.rangeSets = rangeSets
632: (0)            class WorksheetSource(Serialisable):
633: (4)                tagname = "worksheetSource"
634: (4)                ref = String(allow_none=True)
635: (4)                name = String(allow_none=True)
636: (4)                sheet = String(allow_none=True)
637: (4)                def __init__(self,
638: (17)                            ref=None,
639: (17)                            name=None,
640: (17)                            sheet=None,
641: (16)                           ):
642: (8)                    self.ref = ref
643: (8)                    self.name = name
644: (8)                    self.sheet = sheet
645: (0)            class CacheSource(Serialisable):
646: (4)                tagname = "cacheSource"
647: (4)                type = Set(values=(['worksheet', 'external', 'consolidation',
'scenario']))
648: (4)                connectionId = Integer(allow_none=True)
649: (4)                worksheetSource = Typed(expected_type=WorksheetSource, allow_none=True)
650: (4)                consolidation = Typed(expected_type=Consolidation, allow_none=True)
651: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
652: (4)                __elements__ = ('worksheetSource', 'consolidation',)
653: (4)                def __init__(self,
654: (17)                            type=None,
655: (17)                            connectionId=None,
656: (17)                            worksheetSource=None,
657: (17)                            consolidation=None,
658: (17)                            extLst=None,
659: (16)                           ):
660: (8)                    self.type = type
661: (8)                    self.connectionId = connectionId
662: (8)                    self.worksheetSource = worksheetSource
663: (8)                    self.consolidation = consolidation
664: (0)            class CacheDefinition(Serialisable):
665: (4)                mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.pivotCacheDefinition+xml"
666: (4)                rel_type =
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/pivotCacheDefinition"
667: (4)                _id = 1
```

```
668: (4)                _path = "/xl/pivotCache/pivotCacheDefinition{0}.xml"
669: (4)                records = None
670: (4)                tagname = "pivotCacheDefinition"
671: (4)                invalid = Bool(allow_none=True)
672: (4)                saveData = Bool(allow_none=True)
673: (4)                refreshOnLoad = Bool(allow_none=True)
674: (4)                optimizeMemory = Bool(allow_none=True)
675: (4)                enableRefresh = Bool(allow_none=True)
676: (4)                refreshedBy = String(allow_none=True)
677: (4)                refreshedDate = Float(allow_none=True)
678: (4)                refreshedDateIso = DateTime(allow_none=True)
679: (4)                backgroundQuery = Bool(allow_none=True)
680: (4)                missingItemsLimit = Integer(allow_none=True)
681: (4)                createdVersion = Integer(allow_none=True)
682: (4)                refreshedVersion = Integer(allow_none=True)
683: (4)                minRefreshableVersion = Integer(allow_none=True)
684: (4)                recordCount = Integer(allow_none=True)
685: (4)                upgradeOnRefresh = Bool(allow_none=True)
686: (4)                supportSubquery = Bool(allow_none=True)
687: (4)                supportAdvancedDrill = Bool(allow_none=True)
688: (4)                cacheSource = Typed(expected_type=CacheSource)
689: (4)                cacheFields = NestedSequence(expected_type=CacheField, count=True)
690: (4)                cacheHierarchies = NestedSequence(expected_type=CacheHierarchy,
allow_none=True)
691: (4)                kpis = NestedSequence(expected_type=OLAPKPI, count=True)
692: (4)                tupleCache = Typed(expected_type=TupleCache, allow_none=True)
693: (4)                calculatedItems = NestedSequence(expected_type=CalculatedItem, count=True)
694: (4)                calculatedMembers = NestedSequence(expected_type=CalculatedMember,
count=True)
695: (4)                dimensions = NestedSequence(expected_type=PivotDimension, allow_none=True)
696: (4)                measureGroups = NestedSequence(expected_type=MeasureGroup, count=True)
697: (4)                maps = NestedSequence(expected_type=MeasureDimensionMap, count=True)
698: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
699: (4)                id = Relation()
700: (4)                __elements__ = ('cacheSource', 'cacheFields', 'cacheHierarchies', 'kpis',
701: (20)                                'tupleCache', 'calculatedItems', 'calculatedMembers',
'dimensions',
702: (20)                                'measureGroups', 'maps',)
703: (4)                def __init__(self,
704: (17)                             invalid=None,
705: (17)                             saveData=None,
706: (17)                             refreshOnLoad=None,
707: (17)                             optimizeMemory=None,
708: (17)                             enableRefresh=None,
709: (17)                             refreshedBy=None,
710: (17)                             refreshedDate=None,
711: (17)                             refreshedDateIso=None,
712: (17)                             backgroundQuery=None,
713: (17)                             missingItemsLimit=None,
714: (17)                             createdVersion=None,
715: (17)                             refreshedVersion=None,
716: (17)                             minRefreshableVersion=None,
717: (17)                             recordCount=None,
718: (17)                             upgradeOnRefresh=None,
719: (17)                             tupleCache=None,
720: (17)                             supportSubquery=None,
721: (17)                             supportAdvancedDrill=None,
722: (17)                             cacheSource=None,
723: (17)                             cacheFields=(),
724: (17)                             cacheHierarchies=(),
725: (17)                             kpis=(),
726: (17)                             calculatedItems=(),
727: (17)                             calculatedMembers=(),
728: (17)                             dimensions=(),
729: (17)                             measureGroups=(),
730: (17)                             maps=(),
731: (17)                             extLst=None,
732: (17)                             id = None,
733: (16)                            ):
```

```
(8)                         self.invalid = invalid
(8)                         self.saveData = saveData
(8)                         self.refreshOnLoad = refreshOnLoad
(8)                         self.optimizeMemory = optimizeMemory
(8)                         self.enableRefresh = enableRefresh
(8)                         self.refreshedBy = refreshedBy
(8)                         self.refreshedDate = refreshedDate
(8)                         self.refreshedDateIso = refreshedDateIso
(8)                         self.backgroundQuery = backgroundQuery
(8)                         self.missingItemsLimit = missingItemsLimit
(8)                         self.createdVersion = createdVersion
(8)                         self.refreshedVersion = refreshedVersion
(8)                         self.minRefreshableVersion = minRefreshableVersion
(8)                         self.recordCount = recordCount
(8)                         self.upgradeOnRefresh = upgradeOnRefresh
(8)                         self.supportSubquery = supportSubquery
(8)                         self.supportAdvancedDrill = supportAdvancedDrill
(8)                         self.cacheSource = cacheSource
(8)                         self.cacheFields = cacheFields
(8)                         self.cacheHierarchies = cacheHierarchies
(8)                         self.kpis = kpis
(8)                         self.tupleCache = tupleCache
(8)                         self.calculatedItems = calculatedItems
(8)                         self.calculatedMembers = calculatedMembers
(8)                         self.dimensions = dimensions
(8)                         self.measureGroups = measureGroups
(8)                         self.maps = maps
(8)                         self.id = id
(4)                     def to_tree(self):
(8)                         node = super().to_tree()
(8)                         node.set("xmlns", SHEET_MAIN_NS)
(8)                         return node
(4)                     @property
(4)                     def path(self):
(8)                         return self._path.format(self._id)
(4)                     def _write(self, archive, manifest):
(8)                         """
(8)                         Add to zipfile and update manifest
(8)                         """
(8)                         self._write_rels(archive, manifest)
(8)                         xml = tostring(self.to_tree())
(8)                         archive.writestr(self.path[1:], xml)
(8)                         manifest.append(self)
(4)                     def _write_rels(self, archive, manifest):
(8)                         """
(8)                         Write the relevant child objects and add links
(8)                         """
(8)                         if self.records is None:
(12)                            return
(8)                         rels = RelationshipList()
(8)                         r = Relationship(Type=self.records.rel_type, Target=self.records.path)
(8)                         rels.append(r)
(8)                         self.id = r.id
(8)                         self.records._id = self._id
(8)                         self.records._write(archive, manifest)
(8)                         path = get_rels_path(self.path)
(8)                         xml = tostring(rels.to_tree())
(8)                         archive.writestr(path[1:], xml)
```

File 92 - table.py:

```
(0)                 from collections import defaultdict
(0)                 from openpyxl.descriptors.serialisable import Serialisable
(0)                 from openpyxl.descriptors import (
(4)                     Typed,
(4)                     Integer,
(4)                     NoneSet,
7: (4)                  Set,
8: (4)                  Bool,
9: (4)                  String,
10: (4)                 Bool,
11: (4)                 Sequence,
12: (0)             )
13: (0)             from openpyxl.descriptors.excel import ExtensionList, Relation
14: (0)             from openpyxl.descriptors.sequence import NestedSequence
15: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
16: (0)             from openpyxl.xml.functions import tostring
17: (0)             from openpyxl.packaging.relationship import (
18: (4)                 RelationshipList,
19: (4)                 Relationship,
20: (4)                 get_rels_path
21: (0)             )
22: (0)             from .fields import Index
23: (0)             from openpyxl.worksheet.filters import (
24: (4)                 AutoFilter,
25: (0)             )
26: (0)             class HierarchyUsage(Serialisable):
27: (4)                 tagname = "hierarchyUsage"
28: (4)                 hierarchyUsage = Integer()
29: (4)                 def __init__(self,
30: (17)                             hierarchyUsage=None,
31: (16)                            ):
32: (8)                     self.hierarchyUsage = hierarchyUsage
33: (0)             class ColHierarchiesUsage(Serialisable):
34: (4)                 tagname = "colHierarchiesUsage"
35: (4)                 colHierarchyUsage = Sequence(expected_type=HierarchyUsage, )
36: (4)                 __elements__ = ('colHierarchyUsage',)
37: (4)                 __attrs__ = ('count', )
38: (4)                 def __init__(self,
39: (17)                             count=None,
40: (17)                             colHierarchyUsage=(),
41: (16)                            ):
42: (8)                     self.colHierarchyUsage = colHierarchyUsage
43: (4)                 @property
44: (4)                 def count(self):
45: (8)                     return len(self.colHierarchyUsage)
46: (0)             class RowHierarchiesUsage(Serialisable):
47: (4)                 tagname = "rowHierarchiesUsage"
48: (4)                 rowHierarchyUsage = Sequence(expected_type=HierarchyUsage, )
49: (4)                 __elements__ = ('rowHierarchyUsage',)
50: (4)                 __attrs__ = ('count', )
51: (4)                 def __init__(self,
52: (17)                             count=None,
53: (17)                             rowHierarchyUsage=(),
54: (16)                            ):
55: (8)                     self.rowHierarchyUsage = rowHierarchyUsage
56: (4)                 @property
57: (4)                 def count(self):
58: (8)                     return len(self.rowHierarchyUsage)
59: (0)             class PivotFilter(Serialisable):
60: (4)                 tagname = "filter"
61: (4)                 fld = Integer()
62: (4)                 mpFld = Integer(allow_none=True)
63: (4)                 type = Set(values=(['unknown', 'count', 'percent', 'sum', 'captionEqual',
64: (24)                                    'captionNotEqual', 'captionBeginsWith',
'captionNotBeginsWith',
65: (24)                                    'captionEndsWith', 'captionNotEndsWith',
'captionContains',
66: (24)                                    'captionNotContains', 'captionGreaterThan',
'captionGreaterThanOrEqual',
67: (24)                                    'captionLessThan', 'captionLessThanOrEqual',
'captionBetween',
68: (24)                                    'captionNotBetween', 'valueEqual', 'valueNotEqual',
'valueGreaterThan',
69: (24)                                    'valueGreaterThanOrEqual', 'valueLessThan',
'valueLessThanOrEqual',
70: (24)                                    'valueBetween', 'valueNotBetween', 'dateEqual',
'dateNotEqual',
71: (24)                                    'dateOlderThan', 'dateOlderThanOrEqual',
'dateNewerThan',
72: (24)                                    'dateNewerThanOrEqual', 'dateBetween',
'dateNotBetween', 'tomorrow',
73: (24)                                    'today', 'yesterday', 'nextWeek', 'thisWeek',
'lastWeek', 'nextMonth',
74: (24)                                    'thisMonth', 'lastMonth', 'nextQuarter',
'thisQuarter', 'lastQuarter',
75: (24)                                    'nextYear', 'thisYear', 'lastYear', 'yearToDate',
'Q1', 'Q2', 'Q3', 'Q4',
76: (24)                                    'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9',
'M10', 'M11',
77: (24)                                    'M12']))
78: (4)                 evalOrder = Integer(allow_none=True)
79: (4)                 id = Integer()
80: (4)                 iMeasureHier = Integer(allow_none=True)
81: (4)                 iMeasureFld = Integer(allow_none=True)
82: (4)                 name = String(allow_none=True)
83: (4)                 description = String(allow_none=True)
84: (4)                 stringValue1 = String(allow_none=True)
85: (4)                 stringValue2 = String(allow_none=True)
86: (4)                 autoFilter = Typed(expected_type=AutoFilter, )
87: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
88: (4)                 __elements__ = ('autoFilter',)
89: (4)                 def __init__(self,
90: (17)                             fld=None,
91: (17)                             mpFld=None,
92: (17)                             type=None,
93: (17)                             evalOrder=None,
94: (17)                             id=None,
95: (17)                             iMeasureHier=None,
96: (17)                             iMeasureFld=None,
97: (17)                             name=None,
98: (17)                             description=None,
99: (17)                             stringValue1=None,
100: (17)                            stringValue2=None,
101: (17)                            autoFilter=None,
102: (17)                            extLst=None,
103: (16)                           ):
104: (8)                    self.fld = fld
105: (8)                    self.mpFld = mpFld
106: (8)                    self.type = type
107: (8)                    self.evalOrder = evalOrder
108: (8)                    self.id = id
109: (8)                    self.iMeasureHier = iMeasureHier
110: (8)                    self.iMeasureFld = iMeasureFld
111: (8)                    self.name = name
112: (8)                    self.description = description
113: (8)                    self.stringValue1 = stringValue1
114: (8)                    self.stringValue2 = stringValue2
115: (8)                    self.autoFilter = autoFilter
116: (0)             class PivotFilters(Serialisable):
117: (4)                 count = Integer()
118: (4)                 filter = Typed(expected_type=PivotFilter, allow_none=True)
119: (4)                 __elements__ = ('filter',)
120: (4)                 def __init__(self,
121: (17)                            count=None,
122: (17)                            filter=None,
123: (16)                           ):
124: (8)                    self.filter = filter
125: (0)             class PivotTableStyle(Serialisable):
126: (4)                 tagname = "pivotTableStyleInfo"
127: (4)                 name = String(allow_none=True)
128: (4)                 showRowHeaders = Bool()
129: (4)                 showColHeaders = Bool()
130: (4)                 showRowStripes = Bool()
131: (4)                 showColStripes = Bool()
```

```
(4)                showLastColumn = Bool()
(4)                def __init__(self,
(17)                            name=None,
(17)                            showRowHeaders=None,
(17)                            showColHeaders=None,
(17)                            showRowStripes=None,
(17)                            showColStripes=None,
(17)                            showLastColumn=None,
(16)                           ):
(8)                    self.name = name
(8)                    self.showRowHeaders = showRowHeaders
(8)                    self.showColHeaders = showColHeaders
(8)                    self.showRowStripes = showRowStripes
(8)                    self.showColStripes = showColStripes
(8)                    self.showLastColumn = showLastColumn
(0)            class MemberList(Serialisable):
(4)                tagname = "members"
(4)                level = Integer(allow_none=True)
(4)                member = NestedSequence(expected_type=String, attribute="name")
(4)                __elements__ = ('member',)
(4)                def __init__(self,
(17)                            count=None,
(17)                            level=None,
(17)                            member=(),
(16)                           ):
(8)                    self.level = level
(8)                    self.member = member
(4)                @property
(4)                def count(self):
(8)                    return len(self.member)
(0)            class MemberProperty(Serialisable):
(4)                tagname = "mps"
(4)                name = String(allow_none=True)
(4)                showCell = Bool(allow_none=True)
(4)                showTip = Bool(allow_none=True)
(4)                showAsCaption = Bool(allow_none=True)
(4)                nameLen = Integer(allow_none=True)
(4)                pPos = Integer(allow_none=True)
(4)                pLen = Integer(allow_none=True)
(4)                level = Integer(allow_none=True)
(4)                field = Integer()
(4)                def __init__(self,
(17)                            name=None,
(17)                            showCell=None,
(17)                            showTip=None,
(17)                            showAsCaption=None,
(17)                            nameLen=None,
(17)                            pPos=None,
(17)                            pLen=None,
(17)                            level=None,
(17)                            field=None,
(16)                           ):
(8)                    self.name = name
(8)                    self.showCell = showCell
(8)                    self.showTip = showTip
(8)                    self.showAsCaption = showAsCaption
(8)                    self.nameLen = nameLen
(8)                    self.pPos = pPos
(8)                    self.pLen = pLen
(8)                    self.level = level
(8)                    self.field = field
(0)            class PivotHierarchy(Serialisable):
(4)                tagname = "pivotHierarchy"
(4)                outline = Bool()
(4)                multipleItemSelectionAllowed = Bool()
(4)                subtotalTop = Bool()
(4)                showInFieldList = Bool()
(4)                dragToRow = Bool()
(4)                dragToCol = Bool()
```

```
(4)         dragToPage = Bool()
(4)         dragToData = Bool()
(4)         dragOff = Bool()
(4)         includeNewItemsInFilter = Bool()
(4)         caption = String(allow_none=True)
(4)         mps = NestedSequence(expected_type=MemberProperty, count=True)
(4)         members = Typed(expected_type=MemberList, allow_none=True)
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         __elements__ = ('mps', 'members',)
(4)         def __init__(self,
(17)                     outline=None,
(17)                     multipleItemSelectionAllowed=None,
(17)                     subtotalTop=None,
(17)                     showInFieldList=None,
(17)                     dragToRow=None,
(17)                     dragToCol=None,
(17)                     dragToPage=None,
(17)                     dragToData=None,
(17)                     dragOff=None,
(17)                     includeNewItemsInFilter=None,
(17)                     caption=None,
(17)                     mps=(),
(17)                     members=None,
(17)                     extLst=None,
(16)                    ):
(8)             self.outline = outline
(8)             self.multipleItemSelectionAllowed = multipleItemSelectionAllowed
(8)             self.subtotalTop = subtotalTop
(8)             self.showInFieldList = showInFieldList
(8)             self.dragToRow = dragToRow
(8)             self.dragToCol = dragToCol
(8)             self.dragToPage = dragToPage
(8)             self.dragToData = dragToData
(8)             self.dragOff = dragOff
(8)             self.includeNewItemsInFilter = includeNewItemsInFilter
(8)             self.caption = caption
(8)             self.mps = mps
(8)             self.members = members
(8)             self.extLst = extLst
(0)     class Reference(Serialisable):
(4)         tagname = "reference"
(4)         field = Integer(allow_none=True)
(4)         selected = Bool(allow_none=True)
(4)         byPosition = Bool(allow_none=True)
(4)         relative = Bool(allow_none=True)
(4)         defaultSubtotal = Bool(allow_none=True)
(4)         sumSubtotal = Bool(allow_none=True)
(4)         countASubtotal = Bool(allow_none=True)
(4)         avgSubtotal = Bool(allow_none=True)
(4)         maxSubtotal = Bool(allow_none=True)
(4)         minSubtotal = Bool(allow_none=True)
(4)         productSubtotal = Bool(allow_none=True)
(4)         countSubtotal = Bool(allow_none=True)
(4)         stdDevSubtotal = Bool(allow_none=True)
(4)         stdDevPSubtotal = Bool(allow_none=True)
(4)         varSubtotal = Bool(allow_none=True)
(4)         varPSubtotal = Bool(allow_none=True)
(4)         x = Sequence(expected_type=Index)
(4)         extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)         __elements__ = ('x',)
(4)         def __init__(self,
(17)                     field=None,
(17)                     count=None,
(17)                     selected=None,
(17)                     byPosition=None,
(17)                     relative=None,
(17)                     defaultSubtotal=None,
(17)                     sumSubtotal=None,
(17)                     countASubtotal=None,
(17)                   avgSubtotal=None,
(17)                   maxSubtotal=None,
(17)                   minSubtotal=None,
(17)                   productSubtotal=None,
(17)                   countSubtotal=None,
(17)                   stdDevSubtotal=None,
(17)                   stdDevPSubtotal=None,
(17)                   varSubtotal=None,
(17)                   varPSubtotal=None,
(17)                   x=(),
(17)                   extLst=None,
(16)                  ):
(8)           self.field = field
(8)           self.selected = selected
(8)           self.byPosition = byPosition
(8)           self.relative = relative
(8)           self.defaultSubtotal = defaultSubtotal
(8)           self.sumSubtotal = sumSubtotal
(8)           self.countASubtotal = countASubtotal
(8)           self.avgSubtotal = avgSubtotal
(8)           self.maxSubtotal = maxSubtotal
(8)           self.minSubtotal = minSubtotal
(8)           self.productSubtotal = productSubtotal
(8)           self.countSubtotal = countSubtotal
(8)           self.stdDevSubtotal = stdDevSubtotal
(8)           self.stdDevPSubtotal = stdDevPSubtotal
(8)           self.varSubtotal = varSubtotal
(8)           self.varPSubtotal = varPSubtotal
(8)           self.x = x
(4)       @property
(4)       def count(self):
(8)           return len(self.field)
(0)   class PivotArea(Serialisable):
(4)       tagname = "pivotArea"
(4)       references = NestedSequence(expected_type=Reference, count=True)
(4)       extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)       field = Integer(allow_none=True)
(4)       type = NoneSet(values=(['normal', 'data', 'all', 'origin', 'button',
(28)                              'topEnd', 'topRight']))
(4)       dataOnly = Bool(allow_none=True)
(4)       labelOnly = Bool(allow_none=True)
(4)       grandRow = Bool(allow_none=True)
(4)       grandCol = Bool(allow_none=True)
(4)       cacheIndex = Bool(allow_none=True)
(4)       outline = Bool(allow_none=True)
(4)       offset = String(allow_none=True)
(4)       collapsedLevelsAreSubtotals = Bool(allow_none=True)
(4)       axis = NoneSet(values=(['axisRow', 'axisCol', 'axisPage', 'axisValues']))
(4)       fieldPosition = Integer(allow_none=True)
(4)       __elements__ = ('references',)
(4)       def __init__(self,
(17)                   references=(),
(17)                   extLst=None,
(17)                   field=None,
(17)                   type="normal",
(17)                   dataOnly=True,
(17)                   labelOnly=None,
(17)                   grandRow=None,
(17)                   grandCol=None,
(17)                   cacheIndex=None,
(17)                   outline=True,
(17)                   offset=None,
(17)                   collapsedLevelsAreSubtotals=None,
(17)                   axis=None,
(17)                   fieldPosition=None,
(16)                  ):
(8)           self.references = references
(8)           self.extLst = extLst
(8)           self.field = field
```

```
339: (8)                    self.type = type
340: (8)                    self.dataOnly = dataOnly
341: (8)                    self.labelOnly = labelOnly
342: (8)                    self.grandRow = grandRow
343: (8)                    self.grandCol = grandCol
344: (8)                    self.cacheIndex = cacheIndex
345: (8)                    self.outline = outline
346: (8)                    self.offset = offset
347: (8)                    self.collapsedLevelsAreSubtotals = collapsedLevelsAreSubtotals
348: (8)                    self.axis = axis
349: (8)                    self.fieldPosition = fieldPosition
350: (0)            class ChartFormat(Serialisable):
351: (4)                tagname = "chartFormat"
352: (4)                chart = Integer()
353: (4)                format = Integer()
354: (4)                series = Bool()
355: (4)                pivotArea = Typed(expected_type=PivotArea, )
356: (4)                __elements__ = ('pivotArea',)
357: (4)                def __init__(self,
358: (17)                            chart=None,
359: (17)                            format=None,
360: (17)                            series=None,
361: (17)                            pivotArea=None,
362: (16)                           ):
363: (8)                    self.chart = chart
364: (8)                    self.format = format
365: (8)                    self.series = series
366: (8)                    self.pivotArea = pivotArea
367: (0)            class ConditionalFormat(Serialisable):
368: (4)                tagname = "conditionalFormat"
369: (4)                scope = Set(values=(['selection', 'data', 'field']))
370: (4)                type = NoneSet(values=(['all', 'row', 'column']))
371: (4)                priority = Integer()
372: (4)                pivotAreas = NestedSequence(expected_type=PivotArea)
373: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
374: (4)                __elements__ = ('pivotAreas',)
375: (4)                def __init__(self,
376: (17)                            scope="selection",
377: (17)                            type=None,
378: (17)                            priority=None,
379: (17)                            pivotAreas=(),
380: (17)                            extLst=None,
381: (16)                           ):
382: (8)                    self.scope = scope
383: (8)                    self.type = type
384: (8)                    self.priority = priority
385: (8)                    self.pivotAreas = pivotAreas
386: (8)                    self.extLst = extLst
387: (0)            class ConditionalFormatList(Serialisable):
388: (4)                tagname = "conditionalFormats"
389: (4)                conditionalFormat = Sequence(expected_type=ConditionalFormat)
390: (4)                __attrs__ = ("count",)
391: (4)                def __init__(self, conditionalFormat=(), count=None):
392: (8)                    self.conditionalFormat = conditionalFormat
393: (4)                def by_priority(self):
394: (8)                    """
395: (8)                    Return a dictionary of format objects keyed by (field id and format
property).
396: (8)                    This can be used to map the formats to field but also to dedupe to
match
397: (8)                    worksheet definitions which are grouped by cell range
398: (8)                    """
399: (8)                    fmts = {}
400: (8)                    for fmt in self.conditionalFormat:
401: (12)                       for area in fmt.pivotAreas:
402: (16)                           for ref in area.references:
403: (20)                               for field in ref.x:
404: (24)                                   key = (field.v, fmt.priority)
405: (24)                                   fmts[key] = fmt
```

```
(8)                 return fmts
(4)             def _dedupe(self):
(8)                 """
(8)                 Group formats by field index and priority.
(8)                 Sorted to match sorting and grouping for corresponding worksheet formats
(8)                 The implemtenters notes contain significant deviance from the OOXML
(8)                 specification, in particular how conditional formats in tables relate to
(8)                 those defined in corresponding worksheets and how to determine which
(8)                 format applies to which fields.
(8)                 There are some magical interdependencies:
(8)                 * Every pivot table fmt must have a worksheet cxf with the same priority.
(8)                 * In the reference part the field 4294967294 refers to a data field, the
(8)                 spec says -2
(8)                 * Data fields are referenced by the 0-index reference.x.v value
(8)                 Things are made more complicated by the fact that field items behave
(8)                 diffently if the parent is a reference or shared item: "In Office if the
(8)                 parent is the reference element, then restrictions of this value are
(8)                 defined by reference@field. If the parent is the tables element, then
(8)                 this value specifies the index into the table tag position in @url."
(8)                 Yeah, right!
(8)                 """
(8)                 fmts = self.by_priority()
(8)                 fmts = {field:fmt for (field, priority), fmt in sorted(fmts.items(), reverse=True)}
(8)                 if fmts:
(12)                    self.conditionalFormat = list(fmts.values())
(4)             @property
(4)             def count(self):
(8)                 return len(self.conditionalFormat)
(4)             def to_tree(self, tagname=None):
(8)                 self._dedupe()
(8)                 return super().to_tree(tagname)
(0)         class Format(Serialisable):
(4)             tagname = "format"
(4)             action = NoneSet(values=(['blank', 'formatting', 'drill', 'formula']))
(4)             dxfId = Integer(allow_none=True)
(4)             pivotArea = Typed(expected_type=PivotArea, )
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)             __elements__ = ('pivotArea',)
(4)             def __init__(self,
(17)                         action="formatting",
(17)                         dxfId=None,
(17)                         pivotArea=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.action = action
(8)                 self.dxfId = dxfId
(8)                 self.pivotArea = pivotArea
(8)                 self.extLst = extLst
(0)         class DataField(Serialisable):
(4)             tagname = "dataField"
(4)             name = String(allow_none=True)
(4)             fld = Integer()
(4)             subtotal = Set(values=(['average', 'count', 'countNums', 'max', 'min',
(28)                                    'product', 'stdDev', 'stdDevp', 'sum', 'var', 'varp']))
(4)             showDataAs = Set(values=(['normal', 'difference', 'percent',
(30)                                      'percentDiff', 'runTotal', 'percentOfRow', 'percentOfCol',
(30)                                      'percentOfTotal', 'index']))
(4)             baseField = Integer()
(4)             baseItem = Integer()
(4)             numFmtId = Integer(allow_none=True)
(4)             extLst = Typed(expected_type=ExtensionList, allow_none=True)
```

```
(4)                 __elements__ = ()
(4)                 def __init__(self,
(17)                             name=None,
(17)                             fld=None,
(17)                             subtotal="sum",
(17)                             showDataAs="normal",
(17)                             baseField=-1,
(17)                             baseItem=1048832,
(17)                             numFmtId=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.name = name
(8)                     self.fld = fld
(8)                     self.subtotal = subtotal
(8)                     self.showDataAs = showDataAs
(8)                     self.baseField = baseField
(8)                     self.baseItem = baseItem
(8)                     self.numFmtId = numFmtId
(8)                     self.extLst = extLst
(0)             class PageField(Serialisable):
(4)                 tagname = "pageField"
(4)                 fld = Integer()
(4)                 item = Integer(allow_none=True)
(4)                 hier = Integer(allow_none=True)
(4)                 name = String(allow_none=True)
(4)                 cap = String(allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ()
(4)                 def __init__(self,
(17)                             fld=None,
(17)                             item=None,
(17)                             hier=None,
(17)                             name=None,
(17)                             cap=None,
(17)                             extLst=None,
(16)                            ):
(8)                     self.fld = fld
(8)                     self.item = item
(8)                     self.hier = hier
(8)                     self.name = name
(8)                     self.cap = cap
(8)                     self.extLst = extLst
(0)             class RowColItem(Serialisable):
(4)                 tagname = "i"
(4)                 t = Set(values=(['data', 'default', 'sum', 'countA', 'avg', 'max', 'min',
(21)                                 'product', 'count', 'stdDev', 'stdDevP', 'var', 'varP',
'grand',
(21)                                 'blank']))
(4)                 r = Integer()
(4)                 i = Integer()
(4)                 x = Sequence(expected_type=Index, attribute="v")
(4)                 __elements__ = ('x',)
(4)                 def __init__(self,
(17)                             t="data",
(17)                             r=0,
(17)                             i=0,
(17)                             x=(),
(16)                            ):
(8)                     self.t = t
(8)                     self.r = r
(8)                     self.i = i
(8)                     self.x = x
(0)             class RowColField(Serialisable):
(4)                 tagname = "field"
(4)                 x = Integer()
(4)                 def __init__(self,
(17)                             x=None,
(16)                            ):
(8)                     self.x = x
```

```
(0)  class AutoSortScope(Serialisable):
(4)      pivotArea = Typed(expected_type=PivotArea, )
(4)      __elements__ = ('pivotArea',)
(4)      def __init__(self,
(17)                  pivotArea=None,
(16)                 ):
(8)          self.pivotArea = pivotArea
(0)  class FieldItem(Serialisable):
(4)      tagname = "item"
(4)      n = String(allow_none=True)
(4)      t = Set(values=(['data', 'default', 'sum', 'countA', 'avg', 'max', 'min',
(21)                      'product', 'count', 'stdDev', 'stdDevP', 'var', 'varP', 'grand',
(21)                      'blank']))
(4)      h = Bool(allow_none=True)
(4)      s = Bool(allow_none=True)
(4)      sd = Bool(allow_none=True)
(4)      f = Bool(allow_none=True)
(4)      m = Bool(allow_none=True)
(4)      c = Bool(allow_none=True)
(4)      x = Integer(allow_none=True)
(4)      d = Bool(allow_none=True)
(4)      e = Bool(allow_none=True)
(4)      def __init__(self,
(17)                  n=None,
(17)                  t="data",
(17)                  h=None,
(17)                  s=None,
(17)                  sd=True,
(17)                  f=None,
(17)                  m=None,
(17)                  c=None,
(17)                  x=None,
(17)                  d=None,
(17)                  e=None,
(16)                 ):
(8)          self.n = n
(8)          self.t = t
(8)          self.h = h
(8)          self.s = s
(8)          self.sd = sd
(8)          self.f = f
(8)          self.m = m
(8)          self.c = c
(8)          self.x = x
(8)          self.d = d
(8)          self.e = e
(0)  class PivotField(Serialisable):
(4)      tagname = "pivotField"
(4)      items = NestedSequence(expected_type=FieldItem, count=True)
(4)      autoSortScope = Typed(expected_type=AutoSortScope, allow_none=True)
(4)      extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)      name = String(allow_none=True)
(4)      axis = NoneSet(values=(['axisRow', 'axisCol', 'axisPage', 'axisValues']))
(4)      dataField = Bool(allow_none=True)
(4)      subtotalCaption = String(allow_none=True)
(4)      showDropDowns = Bool(allow_none=True)
(4)      hiddenLevel = Bool(allow_none=True)
(4)      uniqueMemberProperty = String(allow_none=True)
(4)      compact = Bool(allow_none=True)
(4)      allDrilled = Bool(allow_none=True)
(4)      numFmtId = Integer(allow_none=True)
(4)      outline = Bool(allow_none=True)
(4)      subtotalTop = Bool(allow_none=True)
(4)      dragToRow = Bool(allow_none=True)
(4)      dragToCol = Bool(allow_none=True)
(4)      multipleItemSelectionAllowed = Bool(allow_none=True)
(4)      dragToPage = Bool(allow_none=True)
(4)      dragToData = Bool(allow_none=True)
```

```
603: (4)                    dragOff = Bool(allow_none=True)
604: (4)                    showAll = Bool(allow_none=True)
605: (4)                    insertBlankRow = Bool(allow_none=True)
606: (4)                    serverField = Bool(allow_none=True)
607: (4)                    insertPageBreak = Bool(allow_none=True)
608: (4)                    autoShow = Bool(allow_none=True)
609: (4)                    topAutoShow = Bool(allow_none=True)
610: (4)                    hideNewItems = Bool(allow_none=True)
611: (4)                    measureFilter = Bool(allow_none=True)
612: (4)                    includeNewItemsInFilter = Bool(allow_none=True)
613: (4)                    itemPageCount = Integer(allow_none=True)
614: (4)                    sortType = Set(values=(['manual', 'ascending', 'descending']))
615: (4)                    dataSourceSort = Bool(allow_none=True)
616: (4)                    nonAutoSortDefault = Bool(allow_none=True)
617: (4)                    rankBy = Integer(allow_none=True)
618: (4)                    defaultSubtotal = Bool(allow_none=True)
619: (4)                    sumSubtotal = Bool(allow_none=True)
620: (4)                    countASubtotal = Bool(allow_none=True)
621: (4)                    avgSubtotal = Bool(allow_none=True)
622: (4)                    maxSubtotal = Bool(allow_none=True)
623: (4)                    minSubtotal = Bool(allow_none=True)
624: (4)                    productSubtotal = Bool(allow_none=True)
625: (4)                    countSubtotal = Bool(allow_none=True)
626: (4)                    stdDevSubtotal = Bool(allow_none=True)
627: (4)                    stdDevPSubtotal = Bool(allow_none=True)
628: (4)                    varSubtotal = Bool(allow_none=True)
629: (4)                    varPSubtotal = Bool(allow_none=True)
630: (4)                    showPropCell = Bool(allow_none=True)
631: (4)                    showPropTip = Bool(allow_none=True)
632: (4)                    showPropAsCaption = Bool(allow_none=True)
633: (4)                    defaultAttributeDrillState = Bool(allow_none=True)
634: (4)                    __elements__ = ('items', 'autoSortScope',)
635: (4)                    def __init__(self,
636: (17)                                items=(),
637: (17)                                autoSortScope=None,
638: (17)                                name=None,
639: (17)                                axis=None,
640: (17)                                dataField=None,
641: (17)                                subtotalCaption=None,
642: (17)                                showDropDowns=True,
643: (17)                                hiddenLevel=None,
644: (17)                                uniqueMemberProperty=None,
645: (17)                                compact=True,
646: (17)                                allDrilled=None,
647: (17)                                numFmtId=None,
648: (17)                                outline=True,
649: (17)                                subtotalTop=True,
650: (17)                                dragToRow=True,
651: (17)                                dragToCol=True,
652: (17)                                multipleItemSelectionAllowed=None,
653: (17)                                dragToPage=True,
654: (17)                                dragToData=True,
655: (17)                                dragOff=True,
656: (17)                                showAll=True,
657: (17)                                insertBlankRow=None,
658: (17)                                serverField=None,
659: (17)                                insertPageBreak=None,
660: (17)                                autoShow=None,
661: (17)                                topAutoShow=True,
662: (17)                                hideNewItems=None,
663: (17)                                measureFilter=None,
664: (17)                                includeNewItemsInFilter=None,
665: (17)                                itemPageCount=10,
666: (17)                                sortType="manual",
667: (17)                                dataSourceSort=None,
668: (17)                                nonAutoSortDefault=None,
669: (17)                                rankBy=None,
670: (17)                                defaultSubtotal=True,
671: (17)                                sumSubtotal=None,
(17)                 countASubtotal=None,
(17)                 avgSubtotal=None,
(17)                 maxSubtotal=None,
(17)                 minSubtotal=None,
(17)                 productSubtotal=None,
(17)                 countSubtotal=None,
(17)                 stdDevSubtotal=None,
(17)                 stdDevPSubtotal=None,
(17)                 varSubtotal=None,
(17)                 varPSubtotal=None,
(17)                 showPropCell=None,
(17)                 showPropTip=None,
(17)                 showPropAsCaption=None,
(17)                 defaultAttributeDrillState=None,
(17)                 extLst=None,
(16)                ):
(8)         self.items = items
(8)         self.autoSortScope = autoSortScope
(8)         self.name = name
(8)         self.axis = axis
(8)         self.dataField = dataField
(8)         self.subtotalCaption = subtotalCaption
(8)         self.showDropDowns = showDropDowns
(8)         self.hiddenLevel = hiddenLevel
(8)         self.uniqueMemberProperty = uniqueMemberProperty
(8)         self.compact = compact
(8)         self.allDrilled = allDrilled
(8)         self.numFmtId = numFmtId
(8)         self.outline = outline
(8)         self.subtotalTop = subtotalTop
(8)         self.dragToRow = dragToRow
(8)         self.dragToCol = dragToCol
(8)         self.multipleItemSelectionAllowed = multipleItemSelectionAllowed
(8)         self.dragToPage = dragToPage
(8)         self.dragToData = dragToData
(8)         self.dragOff = dragOff
(8)         self.showAll = showAll
(8)         self.insertBlankRow = insertBlankRow
(8)         self.serverField = serverField
(8)         self.insertPageBreak = insertPageBreak
(8)         self.autoShow = autoShow
(8)         self.topAutoShow = topAutoShow
(8)         self.hideNewItems = hideNewItems
(8)         self.measureFilter = measureFilter
(8)         self.includeNewItemsInFilter = includeNewItemsInFilter
(8)         self.itemPageCount = itemPageCount
(8)         self.sortType = sortType
(8)         self.dataSourceSort = dataSourceSort
(8)         self.nonAutoSortDefault = nonAutoSortDefault
(8)         self.rankBy = rankBy
(8)         self.defaultSubtotal = defaultSubtotal
(8)         self.sumSubtotal = sumSubtotal
(8)         self.countASubtotal = countASubtotal
(8)         self.avgSubtotal = avgSubtotal
(8)         self.maxSubtotal = maxSubtotal
(8)         self.minSubtotal = minSubtotal
(8)         self.productSubtotal = productSubtotal
(8)         self.countSubtotal = countSubtotal
(8)         self.stdDevSubtotal = stdDevSubtotal
(8)         self.stdDevPSubtotal = stdDevPSubtotal
(8)         self.varSubtotal = varSubtotal
(8)         self.varPSubtotal = varPSubtotal
(8)         self.showPropCell = showPropCell
(8)         self.showPropTip = showPropTip
(8)         self.showPropAsCaption = showPropAsCaption
(8)         self.defaultAttributeDrillState = defaultAttributeDrillState
(0) class Location(Serialisable):
(4)     tagname = "location"
(4)     ref = String()
```

```
(4)                 firstHeaderRow = Integer()
(4)                 firstDataRow = Integer()
(4)                 firstDataCol = Integer()
(4)                 rowPageCount = Integer(allow_none=True)
(4)                 colPageCount = Integer(allow_none=True)
(4)                 def __init__(self,
(17)                             ref=None,
(17)                             firstHeaderRow=None,
(17)                             firstDataRow=None,
(17)                             firstDataCol=None,
(17)                             rowPageCount=None,
(17)                             colPageCount=None,
(16)                            ):
(8)                     self.ref = ref
(8)                     self.firstHeaderRow = firstHeaderRow
(8)                     self.firstDataRow = firstDataRow
(8)                     self.firstDataCol = firstDataCol
(8)                     self.rowPageCount = rowPageCount
(8)                     self.colPageCount = colPageCount
(0)             class TableDefinition(Serialisable):
(4)                 mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.pivotTable+xml"
(4)                 rel_type =
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/pivotTable"
(4)                 _id = 1
(4)                 _path = "/xl/pivotTables/pivotTable{0}.xml"
(4)                 tagname = "pivotTableDefinition"
(4)                 cache = None
(4)                 name = String()
(4)                 cacheId = Integer()
(4)                 dataOnRows = Bool()
(4)                 dataPosition = Integer(allow_none=True)
(4)                 dataCaption = String()
(4)                 grandTotalCaption = String(allow_none=True)
(4)                 errorCaption = String(allow_none=True)
(4)                 showError = Bool()
(4)                 missingCaption = String(allow_none=True)
(4)                 showMissing = Bool()
(4)                 pageStyle = String(allow_none=True)
(4)                 pivotTableStyle = String(allow_none=True)
(4)                 vacatedStyle = String(allow_none=True)
(4)                 tag = String(allow_none=True)
(4)                 updatedVersion = Integer()
(4)                 minRefreshableVersion = Integer()
(4)                 asteriskTotals = Bool()
(4)                 showItems = Bool()
(4)                 editData = Bool()
(4)                 disableFieldList = Bool()
(4)                 showCalcMbrs = Bool()
(4)                 visualTotals = Bool()
(4)                 showMultipleLabel = Bool()
(4)                 showDataDropDown = Bool()
(4)                 showDrill = Bool()
(4)                 printDrill = Bool()
(4)                 showMemberPropertyTips = Bool()
(4)                 showDataTips = Bool()
(4)                 enableWizard = Bool()
(4)                 enableDrill = Bool()
(4)                 enableFieldProperties = Bool()
(4)                 preserveFormatting = Bool()
(4)                 useAutoFormatting = Bool()
(4)                 pageWrap = Integer()
(4)                 pageOverThenDown = Bool()
(4)                 subtotalHiddenItems = Bool()
(4)                 rowGrandTotals = Bool()
(4)                 colGrandTotals = Bool()
(4)                 fieldPrintTitles = Bool()
(4)                 itemPrintTitles = Bool()
(4)                 mergeItem = Bool()
```

```
808: (4)                showDropZones = Bool()
809: (4)                createdVersion = Integer()
810: (4)                indent = Integer()
811: (4)                showEmptyRow = Bool()
812: (4)                showEmptyCol = Bool()
813: (4)                showHeaders = Bool()
814: (4)                compact = Bool()
815: (4)                outline = Bool()
816: (4)                outlineData = Bool()
817: (4)                compactData = Bool()
818: (4)                published = Bool()
819: (4)                gridDropZones = Bool()
820: (4)                immersive = Bool()
821: (4)                multipleFieldFilters = Bool()
822: (4)                chartFormat = Integer()
823: (4)                rowHeaderCaption = String(allow_none=True)
824: (4)                colHeaderCaption = String(allow_none=True)
825: (4)                fieldListSortAscending = Bool()
826: (4)                mdxSubqueries = Bool()
827: (4)                customListSort = Bool(allow_none=True)
828: (4)                autoFormatId = Integer(allow_none=True)
829: (4)                applyNumberFormats = Bool()
830: (4)                applyBorderFormats = Bool()
831: (4)                applyFontFormats = Bool()
832: (4)                applyPatternFormats = Bool()
833: (4)                applyAlignmentFormats = Bool()
834: (4)                applyWidthHeightFormats = Bool()
835: (4)                location = Typed(expected_type=Location, )
836: (4)                pivotFields = NestedSequence(expected_type=PivotField, count=True)
837: (4)                rowFields = NestedSequence(expected_type=RowColField, count=True)
838: (4)                rowItems = NestedSequence(expected_type=RowColItem, count=True)
839: (4)                colFields = NestedSequence(expected_type=RowColField, count=True)
840: (4)                colItems = NestedSequence(expected_type=RowColItem, count=True)
841: (4)                pageFields = NestedSequence(expected_type=PageField, count=True)
842: (4)                dataFields = NestedSequence(expected_type=DataField, count=True)
843: (4)                formats = NestedSequence(expected_type=Format, count=True)
844: (4)                conditionalFormats = Typed(expected_type=ConditionalFormatList,
allow_none=True)
845: (4)                chartFormats = NestedSequence(expected_type=ChartFormat, count=True)
846: (4)                pivotHierarchies = NestedSequence(expected_type=PivotHierarchy,
count=True)
847: (4)                pivotTableStyleInfo = Typed(expected_type=PivotTableStyle,
allow_none=True)
848: (4)                filters = NestedSequence(expected_type=PivotFilter, count=True)
849: (4)                rowHierarchiesUsage = Typed(expected_type=RowHierarchiesUsage,
allow_none=True)
850: (4)                colHierarchiesUsage = Typed(expected_type=ColHierarchiesUsage,
allow_none=True)
851: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
852: (4)                id = Relation()
853: (4)                __elements__ = ('location', 'pivotFields', 'rowFields', 'rowItems',
854: (20)                               'colFields', 'colItems', 'pageFields', 'dataFields',
'formats',
855: (20)                               'conditionalFormats', 'chartFormats', 'pivotHierarchies',
856: (20)                               'pivotTableStyleInfo', 'filters', 'rowHierarchiesUsage',
857: (20)                               'colHierarchiesUsage',)
858: (4)                def __init__(self,
859: (17)                            name=None,
860: (17)                            cacheId=None,
861: (17)                            dataOnRows=False,
862: (17)                            dataPosition=None,
863: (17)                            dataCaption=None,
864: (17)                            grandTotalCaption=None,
865: (17)                            errorCaption=None,
866: (17)                            showError=False,
867: (17)                            missingCaption=None,
868: (17)                            showMissing=True,
869: (17)                            pageStyle=None,
870: (17)                            pivotTableStyle=None,
871: (17)                                   vacatedStyle=None,
872: (17)                                   tag=None,
873: (17)                                   updatedVersion=0,
874: (17)                                   minRefreshableVersion=0,
875: (17)                                   asteriskTotals=False,
876: (17)                                   showItems=True,
877: (17)                                   editData=False,
878: (17)                                   disableFieldList=False,
879: (17)                                   showCalcMbrs=True,
880: (17)                                   visualTotals=True,
881: (17)                                   showMultipleLabel=True,
882: (17)                                   showDataDropDown=True,
883: (17)                                   showDrill=True,
884: (17)                                   printDrill=False,
885: (17)                                   showMemberPropertyTips=True,
886: (17)                                   showDataTips=True,
887: (17)                                   enableWizard=True,
888: (17)                                   enableDrill=True,
889: (17)                                   enableFieldProperties=True,
890: (17)                                   preserveFormatting=True,
891: (17)                                   useAutoFormatting=False,
892: (17)                                   pageWrap=0,
893: (17)                                   pageOverThenDown=False,
894: (17)                                   subtotalHiddenItems=False,
895: (17)                                   rowGrandTotals=True,
896: (17)                                   colGrandTotals=True,
897: (17)                                   fieldPrintTitles=False,
898: (17)                                   itemPrintTitles=False,
899: (17)                                   mergeItem=False,
900: (17)                                   showDropZones=True,
901: (17)                                   createdVersion=0,
902: (17)                                   indent=1,
903: (17)                                   showEmptyRow=False,
904: (17)                                   showEmptyCol=False,
905: (17)                                   showHeaders=True,
906: (17)                                   compact=True,
907: (17)                                   outline=False,
908: (17)                                   outlineData=False,
909: (17)                                   compactData=True,
910: (17)                                   published=False,
911: (17)                                   gridDropZones=False,
912: (17)                                   immersive=True,
913: (17)                                   multipleFieldFilters=None,
914: (17)                                   chartFormat=0,
915: (17)                                   rowHeaderCaption=None,
916: (17)                                   colHeaderCaption=None,
917: (17)                                   fieldListSortAscending=None,
918: (17)                                   mdxSubqueries=None,
919: (17)                                   customListSort=None,
920: (17)                                   autoFormatId=None,
921: (17)                                   applyNumberFormats=False,
922: (17)                                   applyBorderFormats=False,
923: (17)                                   applyFontFormats=False,
924: (17)                                   applyPatternFormats=False,
925: (17)                                   applyAlignmentFormats=False,
926: (17)                                   applyWidthHeightFormats=False,
927: (17)                                   location=None,
928: (17)                                   pivotFields=(),
929: (17)                                   rowFields=(),
930: (17)                                   rowItems=(),
931: (17)                                   colFields=(),
932: (17)                                   colItems=(),
933: (17)                                   pageFields=(),
934: (17)                                   dataFields=(),
935: (17)                                   formats=(),
936: (17)                                   conditionalFormats=None,
937: (17)                                   chartFormats=(),
938: (17)                                   pivotHierarchies=(),
939: (17)                                   pivotTableStyleInfo=None,
940: (17)                                 filters=(),
941: (17)                                 rowHierarchiesUsage=None,
942: (17)                                 colHierarchiesUsage=None,
943: (17)                                 extLst=None,
944: (17)                                 id=None,
945: (16)                                ):
946: (8)                         self.name = name
947: (8)                         self.cacheId = cacheId
948: (8)                         self.dataOnRows = dataOnRows
949: (8)                         self.dataPosition = dataPosition
950: (8)                         self.dataCaption = dataCaption
951: (8)                         self.grandTotalCaption = grandTotalCaption
952: (8)                         self.errorCaption = errorCaption
953: (8)                         self.showError = showError
954: (8)                         self.missingCaption = missingCaption
955: (8)                         self.showMissing = showMissing
956: (8)                         self.pageStyle = pageStyle
957: (8)                         self.pivotTableStyle = pivotTableStyle
958: (8)                         self.vacatedStyle = vacatedStyle
959: (8)                         self.tag = tag
960: (8)                         self.updatedVersion = updatedVersion
961: (8)                         self.minRefreshableVersion = minRefreshableVersion
962: (8)                         self.asteriskTotals = asteriskTotals
963: (8)                         self.showItems = showItems
964: (8)                         self.editData = editData
965: (8)                         self.disableFieldList = disableFieldList
966: (8)                         self.showCalcMbrs = showCalcMbrs
967: (8)                         self.visualTotals = visualTotals
968: (8)                         self.showMultipleLabel = showMultipleLabel
969: (8)                         self.showDataDropDown = showDataDropDown
970: (8)                         self.showDrill = showDrill
971: (8)                         self.printDrill = printDrill
972: (8)                         self.showMemberPropertyTips = showMemberPropertyTips
973: (8)                         self.showDataTips = showDataTips
974: (8)                         self.enableWizard = enableWizard
975: (8)                         self.enableDrill = enableDrill
976: (8)                         self.enableFieldProperties = enableFieldProperties
977: (8)                         self.preserveFormatting = preserveFormatting
978: (8)                         self.useAutoFormatting = useAutoFormatting
979: (8)                         self.pageWrap = pageWrap
980: (8)                         self.pageOverThenDown = pageOverThenDown
981: (8)                         self.subtotalHiddenItems = subtotalHiddenItems
982: (8)                         self.rowGrandTotals = rowGrandTotals
983: (8)                         self.colGrandTotals = colGrandTotals
984: (8)                         self.fieldPrintTitles = fieldPrintTitles
985: (8)                         self.itemPrintTitles = itemPrintTitles
986: (8)                         self.mergeItem = mergeItem
987: (8)                         self.showDropZones = showDropZones
988: (8)                         self.createdVersion = createdVersion
989: (8)                         self.indent = indent
990: (8)                         self.showEmptyRow = showEmptyRow
991: (8)                         self.showEmptyCol = showEmptyCol
992: (8)                         self.showHeaders = showHeaders
993: (8)                         self.compact = compact
994: (8)                         self.outline = outline
995: (8)                         self.outlineData = outlineData
996: (8)                         self.compactData = compactData
997: (8)                         self.published = published
998: (8)                         self.gridDropZones = gridDropZones
999: (8)                         self.immersive = immersive
1000: (8)                        self.multipleFieldFilters = multipleFieldFilters
1001: (8)                        self.chartFormat = chartFormat
1002: (8)                        self.rowHeaderCaption = rowHeaderCaption
1003: (8)                        self.colHeaderCaption = colHeaderCaption
1004: (8)                        self.fieldListSortAscending = fieldListSortAscending
1005: (8)                        self.mdxSubqueries = mdxSubqueries
1006: (8)                        self.customListSort = customListSort
1007: (8)                        self.autoFormatId = autoFormatId
1008: (8)                        self.applyNumberFormats = applyNumberFormats
```

```
(8)             self.applyBorderFormats = applyBorderFormats
(8)             self.applyFontFormats = applyFontFormats
(8)             self.applyPatternFormats = applyPatternFormats
(8)             self.applyAlignmentFormats = applyAlignmentFormats
(8)             self.applyWidthHeightFormats = applyWidthHeightFormats
(8)             self.location = location
(8)             self.pivotFields = pivotFields
(8)             self.rowFields = rowFields
(8)             self.rowItems = rowItems
(8)             self.colFields = colFields
(8)             self.colItems = colItems
(8)             self.pageFields = pageFields
(8)             self.dataFields = dataFields
(8)             self.formats = formats
(8)             self.conditionalFormats = conditionalFormats
(8)             self.conditionalFormats = None
(8)             self.chartFormats = chartFormats
(8)             self.pivotHierarchies = pivotHierarchies
(8)             self.pivotTableStyleInfo = pivotTableStyleInfo
(8)             self.filters = filters
(8)             self.rowHierarchiesUsage = rowHierarchiesUsage
(8)             self.colHierarchiesUsage = colHierarchiesUsage
(8)             self.extLst = extLst
(8)             self.id = id
(4)         def to_tree(self):
(8)             tree = super().to_tree()
(8)             tree.set("xmlns", SHEET_MAIN_NS)
(8)             return tree
(4)         @property
(4)         def path(self):
(8)             return self._path.format(self._id)
(4)         def _write(self, archive, manifest):
(8)             """
(8)             Add to zipfile and update manifest
(8)             """
(8)             self._write_rels(archive, manifest)
(8)             xml = tostring(self.to_tree())
(8)             archive.writestr(self.path[1:], xml)
(8)             manifest.append(self)
(4)         def _write_rels(self, archive, manifest):
(8)             """
(8)             Write the relevant child objects and add links
(8)             """
(8)             if self.cache is None:
(12)                return
(8)             rels = RelationshipList()
(8)             r = Relationship(Type=self.cache.rel_type, Target=self.cache.path)
(8)             rels.append(r)
(8)             self.id = r.id
(8)             if self.cache.path[1:] not in archive.namelist():
(12)                self.cache._write(archive, manifest)
(8)             path = get_rels_path(self.path)
(8)             xml = tostring(rels.to_tree())
(8)             archive.writestr(path[1:], xml)
(4)         def formatted_fields(self):
(8)             """Map fields to associated conditional formats by priority"""
(8)             if not self.conditionalFormats:
(12)                return {}
(8)             fields = defaultdict(list)
(8)             for idx, prio in self.conditionalFormats.by_priority():
(12)                name = self.dataFields[idx].name
(12)                fields[name].append(prio)
(8)             return fields
(4)         @property
(4)         def summary(self):
(8)             """
(8)             Provide a simplified summary of the table
(8)             """
(8)             return f"{self.name} {dict(self.location)}"
```

File 93 - excel.py:

```
"""Read an xlsx file into Python"""
from zipfile import ZipFile, ZIP_DEFLATED
from io import BytesIO
import os.path
import warnings
from openpyxl.pivot.table import TableDefinition
try:
    from ..tests import KEEP_VBA
except ImportError:
    KEEP_VBA = False
from openpyxl.utils.exceptions import InvalidFileException
from openpyxl.xml.constants import (
    ARC_CORE,
    ARC_CUSTOM,
    ARC_CONTENT_TYPES,
    ARC_WORKBOOK,
    ARC_THEME,
    COMMENTS_NS,
    SHARED_STRINGS,
    XLTM,
    XLTX,
    XLSM,
    XLSX,
)
from openpyxl.cell import MergedCell
from openpyxl.comments.comment_sheet import CommentSheet
from .strings import read_string_table, read_rich_text
from .workbook import WorkbookParser
from openpyxl.styles.stylesheet import apply_stylesheet
from openpyxl.packaging.core import DocumentProperties
from openpyxl.packaging.custom import CustomPropertyList
from openpyxl.packaging.manifest import Manifest, Override
from openpyxl.packaging.relationship import (
    RelationshipList,
    get_dependents,
    get_rels_path,
)
from openpyxl.worksheet._read_only import ReadOnlyWorksheet
from openpyxl.worksheet._reader import WorksheetReader
from openpyxl.chartsheet import Chartsheet
from openpyxl.worksheet.table import Table
from openpyxl.drawing.spreadsheet_drawing import SpreadsheetDrawing
from openpyxl.xml.functions import fromstring
from .drawings import find_images
SUPPORTED_FORMATS = ('.xlsx', '.xlsm', '.xltx', '.xltm')
def _validate_archive(filename):
    """
    Does a first check whether filename is a string or a file-like
    object. If it is a string representing a filename, a check is done
    for supported formats by checking the given file-extension. If the
    file-extension is not in SUPPORTED_FORMATS an InvalidFileException
    will raised. Otherwise the filename (resp. file-like object) will
    forwarded to zipfile.ZipFile returning a ZipFile-Instance.
    """
    is_file_like = hasattr(filename, 'read')
    if not is_file_like:
        file_format = os.path.splitext(filename)[-1].lower()
        if file_format not in SUPPORTED_FORMATS:
            if file_format == '.xls':
                msg = ('openpyxl does not support the old .xls file format, '
                       'please use xlrd to read this file, or convert it to '
                       'the more recent .xlsx file format.')
            elif file_format == '.xlsb':
                msg = ('openpyxl does not support binary format .xlsb, '
65: (23)                                   'please convert this file to .xlsx format if you want '
66: (23)                                   'to open it with openpyxl')
67: (12)                        else:
68: (16)                            msg = ('openpyxl does not support %s file format, '
69: (23)                                   'please check you can open '
70: (23)                                   'it with Excel first. '
71: (23)                                   'Supported formats are: %s') % (file_format,
72: (55)
',' .join(SUPPORTED_FORMATS))
73: (12)                        raise InvalidFileException(msg)
74: (4)                 archive = ZipFile(filename, 'r')
75: (4)                 return archive
76: (0)             def _find_workbook_part(package):
77: (4)                 workbook_types = [XLTM, XLTX, XLSM, XLSX]
78: (4)                 for ct in workbook_types:
79: (8)                     part = package.find(ct)
80: (8)                     if part:
81: (12)                        return part
82: (4)                 defaults = {p.ContentType for p in package.Default}
83: (4)                 workbook_type = defaults & set(workbook_types)
84: (4)                 if workbook_type:
85: (8)                     return Override("/" + ARC_WORKBOOK, workbook_type.pop())
86: (4)                 raise IOError("File contains no valid workbook part")
87: (0)             class ExcelReader:
88: (4)                 """
89: (4)                 Read an Excel package and dispatch the contents to the relevant modules
90: (4)                 """
91: (4)                 def __init__(self, fn, read_only=False, keep_vba=KEEP_VBA,
92: (17)                             data_only=False, keep_links=True, rich_text=False):
93: (8)                     self.archive = _validate_archive(fn)
94: (8)                     self.valid_files = self.archive.namelist()
95: (8)                     self.read_only = read_only
96: (8)                     self.keep_vba = keep_vba
97: (8)                     self.data_only = data_only
98: (8)                     self.keep_links = keep_links
99: (8)                     self.rich_text = rich_text
100: (8)                    self.shared_strings = []
101: (4)                def read_manifest(self):
102: (8)                    src = self.archive.read(ARC_CONTENT_TYPES)
103: (8)                    root = fromstring(src)
104: (8)                    self.package = Manifest.from_tree(root)
105: (4)                def read_strings(self):
106: (8)                    ct = self.package.find(SHARED_STRINGS)
107: (8)                    reader = read_string_table
108: (8)                    if self.rich_text:
109: (12)                       reader = read_rich_text
110: (8)                    if ct is not None:
111: (12)                       strings_path = ct.PartName[1:]
112: (12)                       with self.archive.open(strings_path,) as src:
113: (16)                           self.shared_strings = reader(src)
114: (4)                def read_workbook(self):
115: (8)                    wb_part = _find_workbook_part(self.package)
116: (8)                    self.parser = WorkbookParser(self.archive, wb_part.PartName[1:],
keep_links=self.keep_links)
117: (8)                    self.parser.parse()
118: (8)                    wb = self.parser.wb
119: (8)                    wb._sheets = []
120: (8)                    wb._data_only = self.data_only
121: (8)                    wb._read_only = self.read_only
122: (8)                    wb.template = wb_part.ContentType in (XLTX, XLTM)
123: (8)                    if self.keep_vba:
124: (12)                       wb.vba_archive = ZipFile(BytesIO(), 'a', ZIP_DEFLATED)
125: (12)                       for name in self.valid_files:
126: (16)                           wb.vba_archive.writestr(name, self.archive.read(name))
127: (8)                    if self.read_only:
128: (12)                       wb._archive = self.archive
129: (8)                    self.wb = wb
130: (4)                def read_properties(self):
131: (8)                    if ARC_CORE in self.valid_files:
```

```
(12)                src = fromstring(self.archive.read(ARC_CORE))
(12)                self.wb.properties = DocumentProperties.from_tree(src)
(4)         def read_custom(self):
(8)             if ARC_CUSTOM in self.valid_files:
(12)                src = fromstring(self.archive.read(ARC_CUSTOM))
(12)                self.wb.custom_doc_props = CustomPropertyList.from_tree(src)
(4)         def read_theme(self):
(8)             if ARC_THEME in self.valid_files:
(12)                self.wb.loaded_theme = self.archive.read(ARC_THEME)
(4)         def read_chartsheet(self, sheet, rel):
(8)             sheet_path = rel.target
(8)             rels_path = get_rels_path(sheet_path)
(8)             rels = []
(8)             if rels_path in self.valid_files:
(12)                rels = get_dependents(self.archive, rels_path)
(8)             with self.archive.open(sheet_path, "r") as src:
(12)                xml = src.read()
(8)             node = fromstring(xml)
(8)             cs = Chartsheet.from_tree(node)
(8)             cs._parent = self.wb
(8)             cs.title = sheet.name
(8)             self.wb._add_sheet(cs)
(8)             drawings = rels.find(SpreadsheetDrawing._rel_type)
(8)             for rel in drawings:
(12)                charts, images = find_images(self.archive, rel.target)
(12)                for c in charts:
(16)                    cs.add_chart(c)
(4)         def read_worksheets(self):
(8)             comment_warning = """Cell '{0}':{1} is part of a merged range but has a comment which will be removed because merged cells cannot contain any data."""
(8)             for sheet, rel in self.parser.find_sheets():
(12)                if rel.target not in self.valid_files:
(16)                    continue
(12)                if "chartsheet" in rel.Type:
(16)                    self.read_chartsheet(sheet, rel)
(16)                    continue
(12)                rels_path = get_rels_path(rel.target)
(12)                rels = RelationshipList()
(12)                if rels_path in self.valid_files:
(16)                    rels = get_dependents(self.archive, rels_path)
(12)                if self.read_only:
(16)                    ws = ReadOnlyWorksheet(self.wb, sheet.name, rel.target, self.shared_strings)
(16)                    ws.sheet_state = sheet.state
(16)                    self.wb._sheets.append(ws)
(16)                    continue
(12)                else:
(16)                    fh = self.archive.open(rel.target)
(16)                    ws = self.wb.create_sheet(sheet.name)
(16)                    ws._rels = rels
(16)                    ws_parser = WorksheetReader(ws, fh, self.shared_strings, self.data_only, self.rich_text)
(16)                    ws_parser.bind_all()
(16)                    fh.close()
(12)                for r in rels.find(COMMENTS_NS):
(16)                    src = self.archive.read(r.target)
(16)                    comment_sheet = CommentSheet.from_tree(fromstring(src))
(16)                    for ref, comment in comment_sheet.comments:
(20)                        try:
(24)                            ws[ref].comment = comment
(20)                        except AttributeError:
(24)                            c = ws[ref]
(24)                            if isinstance(c, MergedCell):
(28)                                warnings.warn(comment_warning.format(ws.title, c.coordinate))
(28)                                continue
(12)                if self.wb.vba_archive and ws.legacy_drawing:
(16)                    ws.legacy_drawing = rels.get(ws.legacy_drawing).target
(12)                else:
```

```
197: (16)                           ws.legacy_drawing = None
198: (12)                       for t in ws_parser.tables:
199: (16)                           src = self.archive.read(t)
200: (16)                           xml = fromstring(src)
201: (16)                           table = Table.from_tree(xml)
202: (16)                           ws.add_table(table)
203: (12)                       drawings = rels.find(SpreadsheetDrawing._rel_type)
204: (12)                       for rel in drawings:
205: (16)                           charts, images = find_images(self.archive, rel.target)
206: (16)                           for c in charts:
207: (20)                               ws.add_chart(c, c.anchor)
208: (16)                           for im in images:
209: (20)                               ws.add_image(im, im.anchor)
210: (12)                       pivot_rel = rels.find(TableDefinition.rel_type)
211: (12)                       pivot_caches = self.parser.pivot_caches
212: (12)                       for r in pivot_rel:
213: (16)                           pivot_path = r.Target
214: (16)                           src = self.archive.read(pivot_path)
215: (16)                           tree = fromstring(src)
216: (16)                           pivot = TableDefinition.from_tree(tree)
217: (16)                           pivot.cache = pivot_caches[pivot.cacheId]
218: (16)                           ws.add_pivot(pivot)
219: (12)                       ws.sheet_state = sheet.state
220: (4)                def read(self):
221: (8)                    action = "read manifest"
222: (8)                    try:
223: (12)                       self.read_manifest()
224: (12)                       action = "read strings"
225: (12)                       self.read_strings()
226: (12)                       action = "read workbook"
227: (12)                       self.read_workbook()
228: (12)                       action = "read properties"
229: (12)                       self.read_properties()
230: (12)                       action = "read custom properties"
231: (12)                       self.read_custom()
232: (12)                       action = "read theme"
233: (12)                       self.read_theme()
234: (12)                       action = "read stylesheet"
235: (12)                       apply_stylesheet(self.archive, self.wb)
236: (12)                       action = "read worksheets"
237: (12)                       self.read_worksheets()
238: (12)                       action = "assign names"
239: (12)                       self.parser.assign_names()
240: (12)                       if not self.read_only:
241: (16)                           self.archive.close()
242: (8)                    except ValueError as e:
243: (12)                       raise ValueError(
244: (16)                           f"Unable to read workbook: could not {action} from 
{self.archive.filename}.\n"
245: (16)                           "This is most probably because the workbook source files 
contain some invalid XML.\n"
246: (16)                           "Please see the exception for more details."
247: (16)                           ) from e
248: (0)            def load_workbook(filename, read_only=False, keep_vba=KEEP_VBA,
249: (18)                             data_only=False, keep_links=True, rich_text=False):
250: (4)                """Open the given filename and return the workbook
251: (4)                :param filename: the path to open or a file-like object
252: (4)                :type filename: string or a file-like object open in binary mode c.f., 
:class:`zipfile.ZipFile`
253: (4)                :param read_only: optimised for reading, content cannot be edited
254: (4)                :type read_only: bool
255: (4)                :param keep_vba: preserve vba content (this does NOT mean you can use it)
256: (4)                :type keep_vba: bool
257: (4)                :param data_only: controls whether cells with formulae have either the 
formula (default) or the value stored the last time Excel read the sheet
258: (4)                :type data_only: bool
259: (4)                :param keep_links: whether links to external workbooks should be 
preserved. The default is True
260: (4)                :type keep_links: bool
```

```
261: (4)                :param rich_text: if set to True openpyxl will preserve any rich text
formatting in cells. The default is False
262: (4)                :type rich_text: bool
263: (4)                :rtype: :class:`openpyxl.workbook.Workbook`
264: (4)                .. note::
265: (8)                    When using lazy load, all worksheets will be
:class:`openpyxl.worksheet.iter_worksheet.IterableWorksheet`
266: (8)                    and the returned workbook will be read-only.
267: (4)                """
268: (4)                reader = ExcelReader(filename, read_only, keep_vba,
269: (25)                                     data_only, keep_links, rich_text)
270: (4)                reader.read()
271: (4)                return reader.wb

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

File 94 - custom.py:

1: (0)              """Implementation of custom properties see Â§ 22.3 in the specification"""
2: (0)              from warnings import warn
3: (0)              from openpyxl.descriptors import Strict
4: (0)              from openpyxl.descriptors.serialisable import Serialisable
5: (0)              from openpyxl.descriptors.sequence import Sequence
6: (0)              from openpyxl.descriptors import (
7: (4)                  Alias,
8: (4)                  String,
9: (4)                  Integer,
10: (4)                 Float,
11: (4)                 DateTime,
12: (4)                 Bool,
13: (0)             )
14: (0)             from openpyxl.descriptors.nested import (
15: (4)                 NestedText,
16: (0)             )
17: (0)             from openpyxl.xml.constants import (
18: (4)                 CUSTPROPS_NS,
19: (4)                 VTYPES_NS,
20: (4)                 CPROPS_FMTID,
21: (0)             )
22: (0)             from .core import NestedDateTime
23: (0)             class NestedBoolText(Bool, NestedText):
24: (4)                 """
25: (4)                 Descriptor for handling nested elements with the value stored in the text
part
26: (4)                 """
27: (4)                 pass
28: (0)             class _CustomDocumentProperty(Serialisable):
29: (4)                 """
30: (4)                 Low-level representation of a Custom Document Property.
31: (4)                 Not used directly
32: (4)                 Must always contain a child element, even if this is empty
33: (4)                 """
34: (4)                 tagname = "property"
35: (4)                 _typ = None
36: (4)                 name = String(allow_none=True)
37: (4)                 lpwstr = NestedText(expected_type=str, allow_none=True,
namespace=VTYPES_NS)
38: (4)                 i4 = NestedText(expected_type=int, allow_none=True, namespace=VTYPES_NS)
39: (4)                 r8 = NestedText(expected_type=float, allow_none=True, namespace=VTYPES_NS)
40: (4)                 filetime = NestedDateTime(allow_none=True, namespace=VTYPES_NS)
41: (4)                 bool = NestedBoolText(expected_type=bool, allow_none=True,
namespace=VTYPES_NS)
42: (4)                 linkTarget = String(expected_type=str, allow_none=True)
43: (4)                 fmtid = String()
44: (4)                 pid = Integer()
45: (4)                 def __init__(self,
46: (17)                             name=None,
47: (17)                             pid=0,
48: (17)                             fmtid=CPROPS_FMTID,
49: (17)                             linkTarget=None,
50: (17)                             **kw):
51: (8)                     self.fmtid = fmtid
52: (8)                     self.pid = pid
53: (8)                     self.name = name
54: (8)                     self._typ = None
55: (8)                     self.linkTarget = linkTarget
56: (8)                     for k, v in kw.items():
57: (12)                        setattr(self, k, v)
58: (12)                        setattr(self, "_typ", k) # ugh!
59: (8)                     for e in self.__elements__:
60: (12)                        if e not in kw:
61: (16)                            setattr(self, e, None)
62: (4)                 @property
63: (4)                 def type(self):
64: (8)                     if self._typ is not None:
65: (12)                        return self._typ
66: (8)                     for a in self.__elements__:
67: (12)                        if getattr(self, a) is not None:
68: (16)                            return a
69: (8)                     if self.linkTarget is not None:
70: (12)                        return "linkTarget"
71: (4)                 def to_tree(self, tagname=None, idx=None, namespace=None):
72: (8)                     child = getattr(self, self._typ, None)
73: (8)                     if child is None:
74: (12)                        setattr(self, self._typ, "")
75: (8)                     return super().to_tree(tagname=None, idx=None, namespace=None)
76: (0)             class _CustomDocumentPropertyList(Serialisable):
77: (4)                 """
78: (4)                 Parses and seriliases property lists but is not used directly
79: (4)                 """
80: (4)                 tagname = "Properties"
81: (4)                 property = Sequence(expected_type=_CustomDocumentProperty,
namespace=CUSTPROPS_NS)
82: (4)                 customProps = Alias("property")
83: (4)                 def __init__(self, property=()):
84: (8)                     self.property = property
85: (4)                 def __len__(self):
86: (8)                     return len(self.property)
87: (4)                 def to_tree(self, tagname=None, idx=None, namespace=None):
88: (8)                     for idx, p in enumerate(self.property, 2):
89: (12)                        p.pid = idx
90: (8)                     tree = super().to_tree(tagname, idx, namespace)
91: (8)                     tree.set("xmlns", CUSTPROPS_NS)
92: (8)                     return tree
93: (0)             class _TypedProperty(Strict):
94: (4)                 name = String()
95: (4)                 def __init__(self,
96: (17)                             name,
97: (17)                             value):
98: (8)                     self.name = name
99: (8)                     self.value = value
100: (4)                def __eq__(self, other):
101: (8)                    return self.name == other.name and self.value == other.value
102: (4)                def __repr__(self):
103: (8)                    return f"{self.__class__.__name__}, name={self.name}, value=
{self.value}"
104: (0)            class IntProperty(_TypedProperty):
105: (4)                value = Integer()
106: (0)            class FloatProperty(_TypedProperty):
107: (4)                value = Float()
108: (0)            class StringProperty(_TypedProperty):
109: (4)                value = String(allow_none=True)
110: (0)            class DateTimeProperty(_TypedProperty):
111: (4)                value = DateTime()
112: (0)            class BoolProperty(_TypedProperty):
113: (4)                value = Bool()
114: (0)            class LinkProperty(_TypedProperty):
115: (4)                value = String()
```

```
CLASS_MAPPING = {
    StringProperty: "lpwstr",
    IntProperty: "i4",
    FloatProperty: "r8",
    DateTimeProperty: "filetime",
    BoolProperty: "bool",
    LinkProperty: "linkTarget"
}
XML_MAPPING = {v:k for k,v in CLASS_MAPPING.items()}
class CustomPropertyList(Strict):
    props = Sequence(expected_type=_TypedProperty)
    def __init__(self):
        self.props = []
    @classmethod
    def from_tree(cls, tree):
        """
        Create list from OOXML element
        """
        prop_list = _CustomDocumentPropertyList.from_tree(tree)
        props = []
        for prop in prop_list.property:
            attr = prop.type
            typ = XML_MAPPING.get(attr, None)
            if not typ:
                warn(f"Unknown type for {prop.name}")
                continue
            value = getattr(prop, attr)
            link = prop.linkTarget
            if link is not None:
                typ = LinkProperty
                value = prop.linkTarget
            new_prop = typ(name=prop.name, value=value)
            props.append(new_prop)
        new_prop_list = cls()
        new_prop_list.props = props
        return new_prop_list
    def append(self, prop):
        if prop.name in self.names:
            raise ValueError(f"Property with name {prop.name} already exists")
        self.props.append(prop)
    def to_tree(self):
        props = []
        for p in self.props:
            attr = CLASS_MAPPING.get(p.__class__, None)
            if not attr:
                raise TypeError("Unknown adapter for {p}")
            np = _CustomDocumentProperty(name=p.name, **{attr:p.value})
            if isinstance(p, LinkProperty):
                np._typ = "lpwstr"
            props.append(np)
        prop_list = _CustomDocumentPropertyList(property=props)
        return prop_list.to_tree()
    def __len__(self):
        return len(self.props)
    @property
    def names(self):
        """List of property names"""
        return [p.name for p in self.props]
    def __getitem__(self, name):
        """
        Get property by name
        """
        for p in self.props:
            if p.name == name:
                return p
        raise KeyError(f"Property with name {name} not found")
    def __delitem__(self, name):
        """
        Delete a propery by name
```

```
(8)                 """
(8)                 for idx, p in enumerate(self.props):
(12)                    if p.name == name:
(16)                        self.props.pop(idx)
(16)                        return
(8)                 raise KeyError(f"Property with name {name} not found")
(4)             def __repr__(self):
(8)                 return f"{self.__class__.__name__} containing {self.props}"
(4)             def __iter__(self):
(8)                 return iter(self.props)
```

```
---------------------------------------
```

File 95 - fields.py:

```
(0)         from openpyxl.descriptors.serialisable import Serialisable
(0)         from openpyxl.descriptors import (
(4)             Typed,
(4)             DateTime,
(4)             Bool,
(4)             Float,
(4)             String,
(4)             Integer,
(4)             Sequence,
(0)         )
(0)         from openpyxl.descriptors.excel import HexBinary
(0)         class Index(Serialisable):
(4)             tagname = "x"
(4)             v = Integer(allow_none=True)
(4)             def __init__(self,
(17)                             v=0,
(16)                            ):
(8)                 self.v = v
(0)         class Tuple(Serialisable):
(4)             tagname = "tpl"
(4)             fld = Integer(allow_none=True)
(4)             hier = Integer(allow_none=True)
(4)             item = Integer()
(4)             def __init__(self,
(17)                             fld=None,
(17)                             hier=None,
(17)                             item=None,
(16)                            ):
(8)                 self.fld = fld
(8)                 self.hier = hier
(8)                 self.item = item
(0)         class TupleList(Serialisable):
(4)             tagname = "tpls"
(4)             c = Integer(allow_none=True)
(4)             tpl = Typed(expected_type=Tuple, )
(4)             __elements__ = ('tpl',)
(4)             def __init__(self,
(17)                             c=None,
(17)                             tpl=None,
(16)                            ):
(8)                 self.c = c
(8)                 self.tpl = tpl
(0)         class Missing(Serialisable):
(4)             tagname = "m"
(4)             tpls = Sequence(expected_type=TupleList)
(4)             x = Sequence(expected_type=Index)
(4)             u = Bool(allow_none=True)
(4)             f = Bool(allow_none=True)
(4)             c = String(allow_none=True)
(4)             cp = Integer(allow_none=True)
(4)             _in = Integer(allow_none=True)
(4)             bc = HexBinary(allow_none=True)
(4)             fc = HexBinary(allow_none=True)
(4)             i = Bool(allow_none=True)
```

```
(4)         un = Bool(allow_none=True)
(4)         st = Bool(allow_none=True)
(4)         b = Bool(allow_none=True)
(4)         __elements__ = ('tpls', 'x')
(4)         def __init__(self,
(17)                     tpls=(),
(17)                     x=(),
(17)                     u=None,
(17)                     f=None,
(17)                     c=None,
(17)                     cp=None,
(17)                     _in=None,
(17)                     bc=None,
(17)                     fc=None,
(17)                     i=None,
(17)                     un=None,
(17)                     st=None,
(17)                     b=None,
(16)                    ):
(8)             self.tpls = tpls
(8)             self.x = x
(8)             self.u = u
(8)             self.f = f
(8)             self.c = c
(8)             self.cp = cp
(8)             self._in = _in
(8)             self.bc = bc
(8)             self.fc = fc
(8)             self.i = i
(8)             self.un = un
(8)             self.st = st
(8)             self.b = b
(0)     class Number(Serialisable):
(4)         tagname = "n"
(4)         tpls = Sequence(expected_type=TupleList)
(4)         x = Sequence(expected_type=Index)
(4)         v = Float()
(4)         u = Bool(allow_none=True)
(4)         f = Bool(allow_none=True)
(4)         c = String(allow_none=True)
(4)         cp = Integer(allow_none=True)
(4)         _in = Integer(allow_none=True)
(4)         bc = HexBinary(allow_none=True)
(4)         fc = HexBinary(allow_none=True)
(4)         i = Bool(allow_none=True)
(4)         un = Bool(allow_none=True)
(4)         st = Bool(allow_none=True)
(4)         b = Bool(allow_none=True)
(4)         __elements__ = ('tpls', 'x')
(4)         def __init__(self,
(17)                     tpls=(),
(17)                     x=(),
(17)                     v=None,
(17)                     u=None,
(17)                     f=None,
(17)                     c=None,
(17)                     cp=None,
(17)                     _in=None,
(17)                     bc=None,
(17)                     fc=None,
(17)                     i=None,
(17)                     un=None,
(17)                     st=None,
(17)                     b=None,
(16)                    ):
(8)             self.tpls = tpls
(8)             self.x = x
(8)             self.v = v
(8)             self.u = u
```

```
(8)                 self.f = f
(8)                 self.c = c
(8)                 self.cp = cp
(8)                 self._in = _in
(8)                 self.bc = bc
(8)                 self.fc = fc
(8)                 self.i = i
(8)                 self.un = un
(8)                 self.st = st
(8)                 self.b = b
(0)         class Error(Serialisable):
(4)             tagname = "e"
(4)             tpls = Typed(expected_type=TupleList, allow_none=True)
(4)             x = Sequence(expected_type=Index)
(4)             v = String()
(4)             u = Bool(allow_none=True)
(4)             f = Bool(allow_none=True)
(4)             c = String(allow_none=True)
(4)             cp = Integer(allow_none=True)
(4)             _in = Integer(allow_none=True)
(4)             bc = HexBinary(allow_none=True)
(4)             fc = HexBinary(allow_none=True)
(4)             i = Bool(allow_none=True)
(4)             un = Bool(allow_none=True)
(4)             st = Bool(allow_none=True)
(4)             b = Bool(allow_none=True)
(4)             __elements__ = ('tpls', 'x')
(4)             def __init__(self,
(17)                         tpls=None,
(17)                         x=(),
(17)                         v=None,
(17)                         u=None,
(17)                         f=None,
(17)                         c=None,
(17)                         cp=None,
(17)                         _in=None,
(17)                         bc=None,
(17)                         fc=None,
(17)                         i=None,
(17)                         un=None,
(17)                         st=None,
(17)                         b=None,
(16)                        ):
(8)                 self.tpls = tpls
(8)                 self.x = x
(8)                 self.v = v
(8)                 self.u = u
(8)                 self.f = f
(8)                 self.c = c
(8)                 self.cp = cp
(8)                 self._in = _in
(8)                 self.bc = bc
(8)                 self.fc = fc
(8)                 self.i = i
(8)                 self.un = un
(8)                 self.st = st
(8)                 self.b = b
(0)         class Boolean(Serialisable):
(4)             tagname = "b"
(4)             x = Sequence(expected_type=Index)
(4)             v = Bool()
(4)             u = Bool(allow_none=True)
(4)             f = Bool(allow_none=True)
(4)             c = String(allow_none=True)
(4)             cp = Integer(allow_none=True)
(4)             __elements__ = ('x',)
(4)             def __init__(self,
(17)                         x=(),
(17)                         v=None,
193: (17)                       u=None,
194: (17)                       f=None,
195: (17)                       c=None,
196: (17)                       cp=None,
197: (16)                      ):
198: (8)                self.x = x
199: (8)                self.v = v
200: (8)                self.u = u
201: (8)                self.f = f
202: (8)                self.c = c
203: (8)                self.cp = cp
204: (0)        class Text(Serialisable):
205: (4)            tagname = "s"
206: (4)            tpls = Sequence(expected_type=TupleList)
207: (4)            x = Sequence(expected_type=Index)
208: (4)            v = String()
209: (4)            u = Bool(allow_none=True)
210: (4)            f = Bool(allow_none=True)
211: (4)            c = String(allow_none=True)
212: (4)            cp = Integer(allow_none=True)
213: (4)            _in = Integer(allow_none=True)
214: (4)            bc = HexBinary(allow_none=True)
215: (4)            fc = HexBinary(allow_none=True)
216: (4)            i = Bool(allow_none=True)
217: (4)            un = Bool(allow_none=True)
218: (4)            st = Bool(allow_none=True)
219: (4)            b = Bool(allow_none=True)
220: (4)            __elements__ = ('tpls', 'x')
221: (4)            def __init__(self,
222: (17)                       tpls=(),
223: (17)                       x=(),
224: (17)                       v=None,
225: (17)                       u=None,
226: (17)                       f=None,
227: (17)                       c=None,
228: (17)                       cp=None,
229: (17)                       _in=None,
230: (17)                       bc=None,
231: (17)                       fc=None,
232: (17)                       i=None,
233: (17)                       un=None,
234: (17)                       st=None,
235: (17)                       b=None,
236: (17)                       ):
237: (8)                self.tpls = tpls
238: (8)                self.x = x
239: (8)                self.v = v
240: (8)                self.u = u
241: (8)                self.f = f
242: (8)                self.c = c
243: (8)                self.cp = cp
244: (8)                self._in = _in
245: (8)                self.bc = bc
246: (8)                self.fc = fc
247: (8)                self.i = i
248: (8)                self.un = un
249: (8)                self.st = st
250: (8)                self.b = b
251: (0)        class DateTimeField(Serialisable):
252: (4)            tagname = "d"
253: (4)            x = Sequence(expected_type=Index)
254: (4)            v = DateTime()
255: (4)            u = Bool(allow_none=True)
256: (4)            f = Bool(allow_none=True)
257: (4)            c = String(allow_none=True)
258: (4)            cp = Integer(allow_none=True)
259: (4)            __elements__ = ('x',)
260: (4)            def __init__(self,
261: (17)                       x=(),
262: (17)                           v=None,
263: (17)                           u=None,
264: (17)                           f=None,
265: (17)                           c=None,
266: (17)                           cp=None,
267: (17)                           ):
268: (8)                    self.x = x
269: (8)                    self.v = v
270: (8)                    self.u = u
271: (8)                    self.f = f
272: (8)                    self.c = c
273: (8)                    self.cp = cp

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

File 96 - record.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Integer,
5: (4)                  Sequence,
6: (0)              )
7: (0)              from openpyxl.descriptors.sequence import (
8: (4)                  MultiSequence,
9: (4)                  MultiSequencePart,
10: (0)             )
11: (0)             from openpyxl.descriptors.excel import ExtensionList
12: (0)             from openpyxl.descriptors.nested import (
13: (4)                 NestedInteger,
14: (4)                 NestedBool,
15: (0)             )
16: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
17: (0)             from openpyxl.xml.functions import tostring
18: (0)             from .fields import (
19: (4)                 Boolean,
20: (4)                 Error,
21: (4)                 Missing,
22: (4)                 Number,
23: (4)                 Text,
24: (4)                 TupleList,
25: (4)                 DateTimeField,
26: (4)                 Index,
27: (0)             )
28: (0)             class Record(Serialisable):
29: (4)                 tagname = "r"
30: (4)                 _fields = MultiSequence()
31: (4)                 m = MultiSequencePart(expected_type=Missing, store="_fields")
32: (4)                 n = MultiSequencePart(expected_type=Number, store="_fields")
33: (4)                 b = MultiSequencePart(expected_type=Boolean, store="_fields")
34: (4)                 e = MultiSequencePart(expected_type=Error, store="_fields")
35: (4)                 s = MultiSequencePart(expected_type=Text,  store="_fields")
36: (4)                 d = MultiSequencePart(expected_type=DateTimeField, store="_fields")
37: (4)                 x = MultiSequencePart(expected_type=Index, store="_fields")
38: (4)                 def __init__(self,
39: (17)                             _fields=(),
40: (17)                             m=None,
41: (17)                             n=None,
42: (17)                             b=None,
43: (17)                             e=None,
44: (17)                             s=None,
45: (17)                             d=None,
46: (17)                             x=None,
47: (16)                            ):
48: (8)                     self._fields = _fields
49: (0)             class RecordList(Serialisable):
50: (4)                 mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.pivotCacheRecords+xml"
51: (4)                 rel_type =
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/pivotCacheRecords"
52: (4)                 _id = 1
53: (4)                 _path = "/xl/pivotCache/pivotCacheRecords{0}.xml"
54: (4)                 tagname ="pivotCacheRecords"
55: (4)                 r = Sequence(expected_type=Record, allow_none=True)
56: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
57: (4)                 __elements__ = ('r', )
58: (4)                 __attrs__ = ('count', )
59: (4)                 def __init__(self,
60: (17)                             count=None,
61: (17)                             r=(),
62: (17)                             extLst=None,
63: (16)                            ):
64: (8)                     self.r = r
65: (8)                     self.extLst = extLst
66: (4)                 @property
67: (4)                 def count(self):
68: (8)                     return len(self.r)
69: (4)                 def to_tree(self):
70: (8)                     tree = super().to_tree()
71: (8)                     tree.set("xmlns", SHEET_MAIN_NS)
72: (8)                     return tree
73: (4)                 @property
74: (4)                 def path(self):
75: (8)                     return self._path.format(self._id)
76: (4)                 def _write(self, archive, manifest):
77: (8)                     """
78: (8)                     Write to zipfile and update manifest
79: (8)                     """
80: (8)                     xml = tostring(self.to_tree())
81: (8)                     archive.writestr(self.path[1:], xml)
82: (8)                     manifest.append(self)
83: (4)                 def _write_rels(self, archive, manifest):
84: (8)                     pass

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

File 97 - strings.py:

1: (0)               from openpyxl.cell.text import Text
2: (0)               from openpyxl.xml.functions import iterparse
3: (0)               from openpyxl.xml.constants import SHEET_MAIN_NS
4: (0)               from openpyxl.cell.rich_text import CellRichText
5: (0)               def read_string_table(xml_source):
6: (4)                   """Read in all shared strings in the table"""
7: (4)                   strings = []
8: (4)                   STRING_TAG = '{%s}si' % SHEET_MAIN_NS
9: (4)                   for _, node in iterparse(xml_source):
10: (8)                      if node.tag == STRING_TAG:
11: (12)                         text = Text.from_tree(node).content
12: (12)                         text = text.replace('x005F_', '')
13: (12)                         node.clear()
14: (12)                         strings.append(text)
15: (4)                   return strings
16: (0)               def read_rich_text(xml_source):
17: (4)                   """Read in all shared strings in the table"""
18: (4)                   strings = []
19: (4)                   STRING_TAG = '{%s}si' % SHEET_MAIN_NS
20: (4)                   for _, node in iterparse(xml_source):
21: (8)                       if node.tag == STRING_TAG:
22: (12)                          text = CellRichText.from_tree(node)
23: (12)                          if len(text) == 0:
24: (16)                              text = ''
25: (12)                          elif len(text) == 1 and isinstance(text[0], str):
26: (16)                              text = text[0]
27: (12)                          node.clear()
28: (12)                          strings.append(text)
29: (4)                   return strings
```

```
----------------------------------------

File 98 - __init__.py:

1: (0)              from .tokenizer import Tokenizer

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

File 99 - extended.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (0)              )
5: (0)              from openpyxl.descriptors.nested import (
6: (4)                  NestedText,
7: (0)              )
8: (0)              from openpyxl.xml.constants import XPROPS_NS
9: (0)              from openpyxl import __version__
10: (0)             class DigSigBlob(Serialisable):
11: (4)                 __elements__ = __attrs__ = ()
12: (0)             class VectorLpstr(Serialisable):
13: (4)                 __elements__ = __attrs__ = ()
14: (0)             class VectorVariant(Serialisable):
15: (4)                 __elements__ = __attrs__ = ()
16: (0)             class ExtendedProperties(Serialisable):
17: (4)                 """
18: (4)                 See 22.2
19: (4)                 Most of this is irrelevant but Excel is very picky about the version
number
20: (4)                 It uses XX.YYYY (Version.Build) and expects everyone else to
21: (4)                 We provide Major.Minor and the full version in the application name
22: (4)                 """
23: (4)                 tagname = "Properties"
24: (4)                 Template = NestedText(expected_type=str, allow_none=True)
25: (4)                 Manager = NestedText(expected_type=str, allow_none=True)
26: (4)                 Company = NestedText(expected_type=str, allow_none=True)
27: (4)                 Pages = NestedText(expected_type=int, allow_none=True)
28: (4)                 Words = NestedText(expected_type=int,allow_none=True)
29: (4)                 Characters = NestedText(expected_type=int, allow_none=True)
30: (4)                 PresentationFormat = NestedText(expected_type=str, allow_none=True)
31: (4)                 Lines = NestedText(expected_type=int, allow_none=True)
32: (4)                 Paragraphs = NestedText(expected_type=int, allow_none=True)
33: (4)                 Slides = NestedText(expected_type=int, allow_none=True)
34: (4)                 Notes = NestedText(expected_type=int, allow_none=True)
35: (4)                 TotalTime = NestedText(expected_type=int, allow_none=True)
36: (4)                 HiddenSlides = NestedText(expected_type=int, allow_none=True)
37: (4)                 MMClips = NestedText(expected_type=int, allow_none=True)
38: (4)                 ScaleCrop = NestedText(expected_type=bool, allow_none=True)
39: (4)                 HeadingPairs = Typed(expected_type=VectorVariant, allow_none=True)
40: (4)                 TitlesOfParts = Typed(expected_type=VectorLpstr, allow_none=True)
41: (4)                 LinksUpToDate = NestedText(expected_type=bool, allow_none=True)
42: (4)                 CharactersWithSpaces = NestedText(expected_type=int, allow_none=True)
43: (4)                 SharedDoc = NestedText(expected_type=bool, allow_none=True)
44: (4)                 HyperlinkBase = NestedText(expected_type=str, allow_none=True)
45: (4)                 HLinks = Typed(expected_type=VectorVariant, allow_none=True)
46: (4)                 HyperlinksChanged = NestedText(expected_type=bool, allow_none=True)
47: (4)                 DigSig = Typed(expected_type=DigSigBlob, allow_none=True)
48: (4)                 Application = NestedText(expected_type=str, allow_none=True)
49: (4)                 AppVersion = NestedText(expected_type=str, allow_none=True)
50: (4)                 DocSecurity = NestedText(expected_type=int, allow_none=True)
51: (4)                 __elements__ = ('Application', 'AppVersion', 'DocSecurity', 'ScaleCrop',
52: (20)                                'LinksUpToDate', 'SharedDoc', 'HyperlinksChanged')
53: (4)                 def __init__(self,
54: (17)                             Template=None,
55: (17)                             Manager=None,
56: (17)                             Company=None,
57: (17)                             Pages=None,
58: (17)                             Words=None,
59: (17)                            Characters=None,
60: (17)                            PresentationFormat=None,
61: (17)                            Lines=None,
62: (17)                            Paragraphs=None,
63: (17)                            Slides=None,
64: (17)                            Notes=None,
65: (17)                            TotalTime=None,
66: (17)                            HiddenSlides=None,
67: (17)                            MMClips=None,
68: (17)                            ScaleCrop=None,
69: (17)                            HeadingPairs=None,
70: (17)                            TitlesOfParts=None,
71: (17)                            LinksUpToDate=None,
72: (17)                            CharactersWithSpaces=None,
73: (17)                            SharedDoc=None,
74: (17)                            HyperlinkBase=None,
75: (17)                            HLinks=None,
76: (17)                            HyperlinksChanged=None,
77: (17)                            DigSig=None,
78: (17)                            Application=None,
79: (17)                            AppVersion=None,
80: (17)                            DocSecurity=None,
81: (16)                           ):
82: (8)                     self.Template = Template
83: (8)                     self.Manager = Manager
84: (8)                     self.Company = Company
85: (8)                     self.Pages = Pages
86: (8)                     self.Words = Words
87: (8)                     self.Characters = Characters
88: (8)                     self.PresentationFormat = PresentationFormat
89: (8)                     self.Lines = Lines
90: (8)                     self.Paragraphs = Paragraphs
91: (8)                     self.Slides = Slides
92: (8)                     self.Notes = Notes
93: (8)                     self.TotalTime = TotalTime
94: (8)                     self.HiddenSlides = HiddenSlides
95: (8)                     self.MMClips = MMClips
96: (8)                     self.ScaleCrop = ScaleCrop
97: (8)                     self.HeadingPairs = None
98: (8)                     self.TitlesOfParts = None
99: (8)                     self.LinksUpToDate = LinksUpToDate
100: (8)                    self.CharactersWithSpaces = CharactersWithSpaces
101: (8)                    self.SharedDoc = SharedDoc
102: (8)                    self.HyperlinkBase = HyperlinkBase
103: (8)                    self.HLinks = None
104: (8)                    self.HyperlinksChanged = HyperlinksChanged
105: (8)                    self.DigSig = None
106: (8)                    self.Application = f"Microsoft Excel Compatible / Openpyxl
{__version__}"
107: (8)                    self.AppVersion = ".".join(__version__.split(".")[:-1])
108: (8)                    self.DocSecurity = DocSecurity
109: (4)                def to_tree(self):
110: (8)                    tree = super().to_tree()
111: (8)                    tree.set("xmlns", XPROPS_NS)
112: (8)                    return tree

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

File 100 - manifest.py:

1: (0)              """
2: (0)              File manifest
3: (0)              """
4: (0)              from mimetypes import MimeTypes
5: (0)              import os.path
6: (0)              from openpyxl.descriptors.serialisable import Serialisable
7: (0)              from openpyxl.descriptors import String, Sequence
8: (0)              from openpyxl.xml.functions import fromstring
9: (0)              from openpyxl.xml.constants import (
10: (4)                 ARC_CONTENT_TYPES,
11: (4)                 ARC_THEME,
12: (4)                 ARC_STYLE,
13: (4)                 THEME_TYPE,
14: (4)                 STYLES_TYPE,
15: (4)                 CONTYPES_NS,
16: (4)                 ACTIVEX,
17: (4)                 CTRL,
18: (4)                 VBA,
19: (0)             )
20: (0)             from openpyxl.xml.functions import tostring
21: (0)             mimetypes = MimeTypes()
22: (0)             mimetypes.add_type('application/xml', ".xml")
23: (0)             mimetypes.add_type('application/vnd.openxmlformats-package.relationships+xml',
".rels")
24: (0)             mimetypes.add_type("application/vnd.ms-office.vbaProject", ".bin")
25: (0)             mimetypes.add_type("application/vnd.openxmlformats-officedocument.vmlDrawing",
".vml")
26: (0)             mimetypes.add_type("image/x-emf", ".emf")
27: (0)             class FileExtension(Serialisable):
28: (4)                 tagname = "Default"
29: (4)                 Extension = String()
30: (4)                 ContentType = String()
31: (4)                 def __init__(self, Extension, ContentType):
32: (8)                     self.Extension = Extension
33: (8)                     self.ContentType = ContentType
34: (0)             class Override(Serialisable):
35: (4)                 tagname = "Override"
36: (4)                 PartName = String()
37: (4)                 ContentType = String()
38: (4)                 def __init__(self, PartName, ContentType):
39: (8)                     self.PartName = PartName
40: (8)                     self.ContentType = ContentType
41: (0)             DEFAULT_TYPES = [
42: (4)                 FileExtension("rels", "application/vnd.openxmlformats-
package.relationships+xml"),
43: (4)                 FileExtension("xml", "application/xml"),
44: (0)             ]
45: (0)             DEFAULT_OVERRIDE = [
46: (4)                 Override("/" + ARC_STYLE, STYLES_TYPE), # Styles
47: (4)                 Override("/" + ARC_THEME, THEME_TYPE), # Theme
48: (4)                 Override("/docProps/core.xml", "application/vnd.openxmlformats-
package.core-properties+xml"),
49: (4)                 Override("/docProps/app.xml", "application/vnd.openxmlformats-
officedocument.extended-properties+xml")
50: (0)             ]
51: (0)             class Manifest(Serialisable):
52: (4)                 tagname = "Types"
53: (4)                 Default = Sequence(expected_type=FileExtension, unique=True)
54: (4)                 Override = Sequence(expected_type=Override, unique=True)
55: (4)                 path = "[Content_Types].xml"
56: (4)                 __elements__ = ("Default", "Override")
57: (4)                 def __init__(self,
58: (17)                             Default=(),
59: (17)                             Override=(),
60: (17)                             ):
61: (8)                     if not Default:
62: (12)                        Default = DEFAULT_TYPES
63: (8)                     self.Default = Default
64: (8)                     if not Override:
65: (12)                        Override = DEFAULT_OVERRIDE
66: (8)                     self.Override = Override
67: (4)                 @property
68: (4)                 def filenames(self):
69: (8)                     return [part.PartName for part in self.Override]
70: (4)                 @property
71: (4)                 def extensions(self):
72: (8)                     """
73: (8)                     Map content types to file extensions
```

```
(8)                         Skip parts without extensions
(8)                         """
(8)                         exts = {os.path.splitext(part.PartName)[-1] for part in self.Override}
(8)                         return [(ext[1:], mimetypes.types_map[True][ext]) for ext in sorted(exts) if ext]
(4)                     def to_tree(self):
(8)                         """
(8)                         Custom serialisation method to allow setting a default namespace
(8)                         """
(8)                         defaults = [t.Extension for t in self.Default]
(8)                         for ext, mime in self.extensions:
(12)                            if ext not in defaults:
(16)                                mime = FileExtension(ext, mime)
(16)                                self.Default.append(mime)
(8)                         tree = super().to_tree()
(8)                         tree.set("xmlns", CONTYPES_NS)
(8)                         return tree
(4)                     def __contains__(self, content_type):
(8)                         """
(8)                         Check whether a particular content type is contained
(8)                         """
(8)                         for t in self.Override:
(12)                            if t.ContentType == content_type:
(16)                                return True
(4)                     def find(self, content_type):
(8)                         """
(8)                         Find specific content-type
(8)                         """
(8)                         try:
(12)                            return next(self.findall(content_type))
(8)                         except StopIteration:
(12)                            return
(4)                     def findall(self, content_type):
(8)                         """
(8)                         Find all elements of a specific content-type
(8)                         """
(8)                         for t in self.Override:
(12)                            if t.ContentType == content_type:
(16)                                yield t
(4)                     def append(self, obj):
(8)                         """
(8)                         Add content object to the package manifest
(8)                         """
(8)                         ct = Override(PartName=obj.path, ContentType=obj.mime_type)
(8)                         self.Override.append(ct)
(4)                     def _write(self, archive, workbook):
(8)                         """
(8)                         Write manifest to the archive
(8)                         """
(8)                         self.append(workbook)
(8)                         self._write_vba(workbook)
(8)                         self._register_mimetypes(filenames=archive.namelist())
(8)                         archive.writestr(self.path, tostring(self.to_tree()))
(4)                     def _register_mimetypes(self, filenames):
(8)                         """
(8)                         Make sure that the mime type for all file extensions is registered
(8)                         """
(8)                         for fn in filenames:
(12)                            ext = os.path.splitext(fn)[-1]
(12)                            if not ext:
(16)                                continue
(12)                            mime = mimetypes.types_map[True][ext]
(12)                            fe = FileExtension(ext[1:], mime)
(12)                            self.Default.append(fe)
(4)                     def _write_vba(self, workbook):
(8)                         """
(8)                         Add content types from cached workbook when keeping VBA
(8)                         """
(8)                         if workbook.vba_archive:
```

```
142: (12)                       node = fromstring(workbook.vba_archive.read(ARC_CONTENT_TYPES))
143: (12)                       mf = Manifest.from_tree(node)
144: (12)                       filenames = self.filenames
145: (12)                       for override in mf.Override:
146: (16)                           if override.PartName not in (ACTIVEX, CTRL, VBA):
147: (20)                               continue
148: (16)                           if override.PartName not in filenames:
149: (20)                               self.Override.append(override)

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

File 101 - workbook.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Alias,
4: (4)                  Typed,
5: (4)                  String,
6: (4)                  Integer,
7: (4)                  Bool,
8: (4)                  NoneSet,
9: (0)              )
10: (0)             from openpyxl.descriptors.excel import ExtensionList, Relation
11: (0)             from openpyxl.descriptors.sequence import NestedSequence
12: (0)             from openpyxl.descriptors.nested import NestedString
13: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
14: (0)             from openpyxl.workbook.defined_name import DefinedNameList
15: (0)             from openpyxl.workbook.external_reference import ExternalReference
16: (0)             from openpyxl.workbook.function_group import FunctionGroupList
17: (0)             from openpyxl.workbook.properties import WorkbookProperties, CalcProperties,
FileVersion
18: (0)             from openpyxl.workbook.protection import WorkbookProtection, FileSharing
19: (0)             from openpyxl.workbook.smart_tags import SmartTagList, SmartTagProperties
20: (0)             from openpyxl.workbook.views import CustomWorkbookView, BookView
21: (0)             from openpyxl.workbook.web import WebPublishing, WebPublishObjectList
22: (0)             class FileRecoveryProperties(Serialisable):
23: (4)                 tagname = "fileRecoveryPr"
24: (4)                 autoRecover = Bool(allow_none=True)
25: (4)                 crashSave = Bool(allow_none=True)
26: (4)                 dataExtractLoad = Bool(allow_none=True)
27: (4)                 repairLoad = Bool(allow_none=True)
28: (4)                 def __init__(self,
29: (17)                             autoRecover=None,
30: (17)                             crashSave=None,
31: (17)                             dataExtractLoad=None,
32: (17)                             repairLoad=None,
33: (16)                            ):
34: (8)                     self.autoRecover = autoRecover
35: (8)                     self.crashSave = crashSave
36: (8)                     self.dataExtractLoad = dataExtractLoad
37: (8)                     self.repairLoad = repairLoad
38: (0)             class ChildSheet(Serialisable):
39: (4)                 """
40: (4)                 Represents a reference to a worksheet or chartsheet in workbook.xml
41: (4)                 It contains the title, order and state but only an indirect reference to
42: (4)                 the objects themselves.
43: (4)                 """
44: (4)                 tagname = "sheet"
45: (4)                 name = String()
46: (4)                 sheetId = Integer()
47: (4)                 state = NoneSet(values=(['visible', 'hidden', 'veryHidden']))
48: (4)                 id = Relation()
49: (4)                 def __init__(self,
50: (17)                             name=None,
51: (17)                             sheetId=None,
52: (17)                             state="visible",
53: (17)                             id=None,
54: (16)                            ):
55: (8)                     self.name = name
```

```
56: (8)                         self.sheetId = sheetId
57: (8)                         self.state = state
58: (8)                         self.id = id
59: (0)                 class PivotCache(Serialisable):
60: (4)                     tagname = "pivotCache"
61: (4)                     cacheId = Integer()
62: (4)                     id = Relation()
63: (4)                     def __init__(self,
64: (17)                                 cacheId=None,
65: (17)                                 id=None
66: (16)                                ):
67: (8)                         self.cacheId = cacheId
68: (8)                         self.id = id
69: (0)                 class WorkbookPackage(Serialisable):
70: (4)                     """
71: (4)                     Represent the workbook file in the archive
72: (4)                     """
73: (4)                     tagname = "workbook"
74: (4)                     conformance = NoneSet(values=['strict', 'transitional'])
75: (4)                     fileVersion = Typed(expected_type=FileVersion, allow_none=True)
76: (4)                     fileSharing = Typed(expected_type=FileSharing, allow_none=True)
77: (4)                     workbookPr = Typed(expected_type=WorkbookProperties, allow_none=True)
78: (4)                     properties = Alias("workbookPr")
79: (4)                     workbookProtection = Typed(expected_type=WorkbookProtection,
allow_none=True)
80: (4)                     bookViews = NestedSequence(expected_type=BookView)
81: (4)                     sheets = NestedSequence(expected_type=ChildSheet)
82: (4)                     functionGroups = Typed(expected_type=FunctionGroupList, allow_none=True)
83: (4)                     externalReferences = NestedSequence(expected_type=ExternalReference)
84: (4)                     definedNames = Typed(expected_type=DefinedNameList, allow_none=True)
85: (4)                     calcPr = Typed(expected_type=CalcProperties, allow_none=True)
86: (4)                     oleSize = NestedString(allow_none=True, attribute="ref")
87: (4)                     customWorkbookViews = NestedSequence(expected_type=CustomWorkbookView)
88: (4)                     pivotCaches = NestedSequence(expected_type=PivotCache, allow_none=True)
89: (4)                     smartTagPr = Typed(expected_type=SmartTagProperties, allow_none=True)
90: (4)                     smartTagTypes = Typed(expected_type=SmartTagList, allow_none=True)
91: (4)                     webPublishing = Typed(expected_type=WebPublishing, allow_none=True)
92: (4)                     fileRecoveryPr = Typed(expected_type=FileRecoveryProperties,
allow_none=True)
93: (4)                     webPublishObjects = Typed(expected_type=WebPublishObjectList,
allow_none=True)
94: (4)                     extLst = Typed(expected_type=ExtensionList, allow_none=True)
95: (4)                     Ignorable =
NestedString(namespace="http://schemas.openxmlformats.org/markup-compatibility/2006",
allow_none=True)
96: (4)                     __elements__ = ('fileVersion', 'fileSharing', 'workbookPr',
97: (20)                                    'workbookProtection', 'bookViews', 'sheets',
'functionGroups',
98: (20)                                    'externalReferences', 'definedNames', 'calcPr', 'oleSize',
99: (20)                                    'customWorkbookViews', 'pivotCaches', 'smartTagPr',
'smartTagTypes',
100: (20)                                   'webPublishing', 'fileRecoveryPr', 'webPublishObjects')
101: (4)                    def __init__(self,
102: (17)                                conformance=None,
103: (17)                                fileVersion=None,
104: (17)                                fileSharing=None,
105: (17)                                workbookPr=None,
106: (17)                                workbookProtection=None,
107: (17)                                bookViews=(),
108: (17)                                sheets=(),
109: (17)                                functionGroups=None,
110: (17)                                externalReferences=(),
111: (17)                                definedNames=None,
112: (17)                                calcPr=None,
113: (17)                                oleSize=None,
114: (17)                                customWorkbookViews=(),
115: (17)                                pivotCaches=(),
116: (17)                                smartTagPr=None,
117: (17)                                smartTagTypes=None,
118: (17)                            webPublishing=None,
119: (17)                            fileRecoveryPr=None,
120: (17)                            webPublishObjects=None,
121: (17)                            extLst=None,
122: (17)                            Ignorable=None,
123: (16)                           ):
124: (8)                    self.conformance = conformance
125: (8)                    self.fileVersion = fileVersion
126: (8)                    self.fileSharing = fileSharing
127: (8)                    if workbookPr is None:
128: (12)                       workbookPr = WorkbookProperties()
129: (8)                    self.workbookPr = workbookPr
130: (8)                    self.workbookProtection = workbookProtection
131: (8)                    self.bookViews = bookViews
132: (8)                    self.sheets = sheets
133: (8)                    self.functionGroups = functionGroups
134: (8)                    self.externalReferences = externalReferences
135: (8)                    self.definedNames = definedNames
136: (8)                    self.calcPr = calcPr
137: (8)                    self.oleSize = oleSize
138: (8)                    self.customWorkbookViews = customWorkbookViews
139: (8)                    self.pivotCaches = pivotCaches
140: (8)                    self.smartTagPr = smartTagPr
141: (8)                    self.smartTagTypes = smartTagTypes
142: (8)                    self.webPublishing = webPublishing
143: (8)                    self.fileRecoveryPr = fileRecoveryPr
144: (8)                    self.webPublishObjects = webPublishObjects
145: (4)                def to_tree(self):
146: (8)                    tree = super().to_tree()
147: (8)                    tree.set("xmlns", SHEET_MAIN_NS)
148: (8)                    return tree
149: (4)                @property
150: (4)                def active(self):
151: (8)                    for view in self.bookViews:
152: (12)                       if view.activeTab is not None:
153: (16)                           return view.activeTab
154: (8)                    return 0
```

File 102 - __init__.py:

```
1: (0)             """
2: (0)             Stuff related to Office OpenXML packaging: relationships, archive, content
types.
3: (0)             """
```

File 103 - __init__.py:

```
1: (0)
```

File 104 - drawings.py:

```
1: (0)             from io import BytesIO
2: (0)             from warnings import warn
3: (0)             from openpyxl.xml.functions import fromstring
4: (0)             from openpyxl.xml.constants import IMAGE_NS
5: (0)             from openpyxl.packaging.relationship import (
6: (4)                 get_rel,
7: (4)                 get_rels_path,
8: (4)                 get_dependents,
9: (0)             )
10: (0)            from openpyxl.drawing.spreadsheet_drawing import SpreadsheetDrawing
11: (0)            from openpyxl.drawing.image import Image, PILImage
12: (0)            from openpyxl.chart.chartspace import ChartSpace
```

```
13: (0)             from openpyxl.chart.reader import read_chart
14: (0)             def find_images(archive, path):
15: (4)                 """
16: (4)                 Given the path to a drawing file extract charts and images
17: (4)                 Ignore errors due to unsupported parts of DrawingML
18: (4)                 """
19: (4)                 src = archive.read(path)
20: (4)                 tree = fromstring(src)
21: (4)                 try:
22: (8)                     drawing = SpreadsheetDrawing.from_tree(tree)
23: (4)                 except TypeError:
24: (8)                     warn("DrawingML support is incomplete and limited to charts and images
only. Shapes and drawings will be lost.")
25: (8)                     return [], []
26: (4)                 rels_path = get_rels_path(path)
27: (4)                 deps = []
28: (4)                 if rels_path in archive.namelist():
29: (8)                     deps = get_dependents(archive, rels_path)
30: (4)                 charts = []
31: (4)                 for rel in drawing._chart_rels:
32: (8)                     try:
33: (12)                        cs = get_rel(archive, deps, rel.id, ChartSpace)
34: (8)                     except TypeError as e:
35: (12)                        warn(f"Unable to read chart {rel.id} from {path} {e}")
36: (12)                        continue
37: (8)                     chart = read_chart(cs)
38: (8)                     chart.anchor = rel.anchor
39: (8)                     charts.append(chart)
40: (4)                 images = []
41: (4)                 if not PILImage: # Pillow not installed, drop images
42: (8)                     return charts, images
43: (4)                 for rel in drawing._blip_rels:
44: (8)                     dep = deps.get(rel.embed)
45: (8)                     if dep.Type == IMAGE_NS:
46: (12)                        try:
47: (16)                            image = Image(BytesIO(archive.read(dep.target)))
48: (12)                        except OSError:
49: (16)                            msg = "The image {0} will be removed because it cannot be
read".format(dep.target)
50: (16)                            warn(msg)
51: (16)                            continue
52: (12)                        if image.format.upper() == "WMF": # cannot save
53: (16)                            msg = "{0} image format is not supported so the image is being
dropped".format(image.format)
54: (16)                            warn(msg)
55: (16)                            continue
56: (12)                        image.anchor = rel.anchor
57: (12)                        images.append(image)
58: (4)                 return charts, images

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

File 105 - __init__.py:

1: (0)

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

File 106 - tokenizer.py:

1: (0)              """
2: (0)              This module contains a tokenizer for Excel formulae.
3: (0)              The tokenizer is based on the Javascript tokenizer found at
4: (0)              http://ewbi.blogs.com/develops/2004/12/excel_formula_p.html written by Eric
5: (0)              Bachtal
6: (0)              """
7: (0)              import re
8: (0)              class TokenizerError(Exception):
9: (4)                  """Base class for all Tokenizer errors."""
```

```python
class Tokenizer:
    """
    A tokenizer for Excel worksheet formulae.
    Converts a str string representing an Excel formula (in A1 notation)
    into a sequence of `Token` objects.
    `formula`: The str string to tokenize
    Tokenizer defines a method `._parse()` to parse the formula into tokens,
    which can then be accessed through the `.items` attribute.
    """
    SN_RE = re.compile("^[1-9](\\.[0-9]+)?[Ee]$")  # Scientific notation
    WSPACE_RE = re.compile(r"[ \n]+")
    STRING_REGEXES = {
        '"': re.compile('"(?:[^"]*"")*[^"]*"(?!")'),
        "'": re.compile("'(?:[^']*'')*[^']*'(?!')"),
    }
    ERROR_CODES = ("#NULL!", "#DIV/0!", "#VALUE!", "#REF!", "#NAME?",
                   "#NUM!", "#N/A", "#GETTING_DATA")
    TOKEN_ENDERS = ',;}) +-*/^&=><%'  # Each of these characters, marks the
    def __init__(self, formula):
        self.formula = formula
        self.items = []
        self.token_stack = []  # Used to keep track of arrays, functions, and
        self.offset = 0  # How many chars have we read
        self.token = []  # Used to build up token values char by char
        self._parse()
    def _parse(self):
        """Populate self.items with the tokens from the formula."""
        if self.offset:
            return  # Already parsed!
        if not self.formula:
            return
        elif self.formula[0] == '=':
            self.offset += 1
        else:
            self.items.append(Token(self.formula, Token.LITERAL))
            return
        consumers = (
            ('"\'', self._parse_string),
            ('[', self._parse_brackets),
            ('#', self._parse_error),
            (' ', self._parse_whitespace),
            ('\n', self._parse_whitespace),
            ('+-*/^&=><%', self._parse_operator),
            ('{(', self._parse_opener),
            (')}', self._parse_closer),
            (';,', self._parse_separator),
        )
        dispatcher = {}  # maps chars to the specific parsing function
        for chars, consumer in consumers:
            dispatcher.update(dict.fromkeys(chars, consumer))
        while self.offset < len(self.formula):
            if self.check_scientific_notation():  # May consume one character
                continue
            curr_char = self.formula[self.offset]
            if curr_char in self.TOKEN_ENDERS:
                self.save_token()
            if curr_char in dispatcher:
                self.offset += dispatcher[curr_char]()
            else:
                self.token.append(curr_char)
                self.offset += 1
        self.save_token()
    def _parse_string(self):
        """
        Parse a "-delimited string or '-delimited link.
        The offset must be pointing to either a single quote ("'") or double
        quote ('"') character. The strings are parsed according to Excel
        rules where to escape the delimiter you just double it up. E.g.,
        "abc""def" in Excel is parsed as 'abc"def' in Python.
        Returns the number of characters matched. (Does not update
        self.offset)
        """
        self.assert_empty_token(can_follow=':')
        delim = self.formula[self.offset]
        assert delim in ('"', "'")
        regex = self.STRING_REGEXES[delim]
        match = regex.match(self.formula[self.offset:])
        if match is None:
            subtype = "string" if delim == '"' else 'link'
            raise TokenizerError(f"Reached end of formula while parsing {subtype} in {self.formula}")
        match = match.group(0)
        if delim == '"':
            self.items.append(Token.make_operand(match))
        else:
            self.token.append(match)
        return len(match)
    def _parse_brackets(self):
        """
        Consume all the text between square brackets [].
        Returns the number of characters matched. (Does not update
        self.offset)
        """
        assert self.formula[self.offset] == '['
        lefts = [(t.start(), 1) for t in
                 re.finditer(r"\[", self.formula[self.offset:])]
        rights = [(t.start(), -1) for t in
                  re.finditer(r"\]", self.formula[self.offset:])]
        open_count = 0
        for idx, open_close in sorted(lefts + rights):
            open_count += open_close
            if open_count == 0:
                outer_right = idx + 1
                self.token.append(
                    self.formula[self.offset:self.offset + outer_right])
                return outer_right
        raise TokenizerError(f"Encountered unmatched '[' in {self.formula}")
    def _parse_error(self):
        """
        Consume the text following a '#' as an error.
        Looks for a match in self.ERROR_CODES and returns the number of
        characters matched. (Does not update self.offset)
        """
        self.assert_empty_token(can_follow='!')
        assert self.formula[self.offset] == '#'
        subformula = self.formula[self.offset:]
        for err in self.ERROR_CODES:
            if subformula.startswith(err):
                self.items.append(Token.make_operand(''.join(self.token) + err))
                del self.token[:]
                return len(err)
        raise TokenizerError(f"Invalid error code at position {self.offset} in '{self.formula}'")
    def _parse_whitespace(self):
        """
        Consume a string of consecutive spaces.
        Returns the number of spaces found. (Does not update self.offset).
        """
        assert self.formula[self.offset] in (' ', '\n')
        self.items.append(Token(self.formula[self.offset], Token.WSPACE))
        return self.WSPACE_RE.match(self.formula[self.offset:]).end()
    def _parse_operator(self):
        """
        Consume the characters constituting an operator.
        Returns the number of characters consumed. (Does not update
        self.offset)
        """
```

```
(8)         if self.formula[self.offset:self.offset + 2] in ('>=', '<=', '<>'):
(12)            self.items.append(Token(
(16)                self.formula[self.offset:self.offset + 2],
(16)                Token.OP_IN
(12)            ))
(12)            return 2
(8)         curr_char = self.formula[self.offset]  # guaranteed to be 1 char
(8)         assert curr_char in '%*/^&=><+-'
(8)         if curr_char == '%':
(12)            token = Token('%', Token.OP_POST)
(8)         elif curr_char in "*/^&=><":
(12)            token = Token(curr_char, Token.OP_IN)
(8)         elif not self.items:
(12)            token = Token(curr_char, Token.OP_PRE)
(8)         else:
(12)            prev = next((i for i in reversed(self.items)
(25)                         if i.type != Token.WSPACE), None)
(12)            is_infix = prev and (
(16)                prev.subtype == Token.CLOSE
(16)                or prev.type == Token.OP_POST
(16)                or prev.type == Token.OPERAND
(12)            )
(12)            if is_infix:
(16)                token = Token(curr_char, Token.OP_IN)
(12)            else:
(16)                token = Token(curr_char, Token.OP_PRE)
(8)         self.items.append(token)
(8)         return 1
(4)     def _parse_opener(self):
(8)         """
(8)         Consumes a ( or { character.
(8)         Returns the number of characters consumed. (Does not update
(8)         self.offset)
(8)         """
(8)         assert self.formula[self.offset] in ('(', '{')
(8)         if self.formula[self.offset] == '{':
(12)            self.assert_empty_token()
(12)            token = Token.make_subexp("{")
(8)         elif self.token:
(12)            token_value = "".join(self.token) + '('
(12)            del self.token[:]
(12)            token = Token.make_subexp(token_value)
(8)         else:
(12)            token = Token.make_subexp("(")
(8)         self.items.append(token)
(8)         self.token_stack.append(token)
(8)         return 1
(4)     def _parse_closer(self):
(8)         """
(8)         Consumes a } or ) character.
(8)         Returns the number of characters consumed. (Does not update
(8)         self.offset)
(8)         """
(8)         assert self.formula[self.offset] in (')', '}')
(8)         token = self.token_stack.pop().get_closer()
(8)         if token.value != self.formula[self.offset]:
(12)            raise TokenizerError(
(16)                "Mismatched ( and { pair in '%s'" % self.formula)
(8)         self.items.append(token)
(8)         return 1
(4)     def _parse_separator(self):
(8)         """
(8)         Consumes a ; or , character.
(8)         Returns the number of characters consumed. (Does not update
(8)         self.offset)
(8)         """
(8)         curr_char = self.formula[self.offset]
(8)         assert curr_char in (';', ',')
(8)         if curr_char == ';':
```

```
(12)                token = Token.make_separator(";")
(8)             else:
(12)                try:
(16)                    top_type = self.token_stack[-1].type
(12)                except IndexError:
(16)                    token = Token(",", Token.OP_IN)  # Range Union operator
(12)                else:
(16)                    if top_type == Token.PAREN:
(20)                        token = Token(",", Token.OP_IN)  # Range Union operator
(16)                    else:
(20)                        token = Token.make_separator(",")
(8)             self.items.append(token)
(8)             return 1
(4)         def check_scientific_notation(self):
(8)             """
(8)             Consumes a + or - character if part of a number in sci. notation.
(8)             Returns True if the character was consumed and self.offset was
(8)             updated, False otherwise.
(8)             """
(8)             curr_char = self.formula[self.offset]
(8)             if (curr_char in '+-'
(16)                    and len(self.token) >= 1
(16)                    and self.SN_RE.match("".join(self.token))):
(12)                self.token.append(curr_char)
(12)                self.offset += 1
(12)                return True
(8)             return False
(4)         def assert_empty_token(self, can_follow=()):
(8)             """
(8)             Ensure that there's no token currently being parsed.
(8)             Or if there is a token being parsed, it must end with a character in
(8)             can_follow.
(8)             If there are unconsumed token contents, it means we hit an unexpected
(8)             token transition. In this case, we raise a TokenizerError
(8)             """
(8)             if self.token and self.token[-1] not in can_follow:
(12)                raise TokenizerError(f"Unexpected character at position
{self.offset} in '{self.formula}'")
(4)         def save_token(self):
(8)             """If there's a token being parsed, add it to the item list."""
(8)             if self.token:
(12)                self.items.append(Token.make_operand("".join(self.token)))
(12)                del self.token[:]
(4)         def render(self):
(8)             """Convert the parsed tokens back to a string."""
(8)             if not self.items:
(12)                return ""
(8)             elif self.items[0].type == Token.LITERAL:
(12)                return self.items[0].value
(8)             return "=" + "".join(token.value for token in self.items)
(0)     class Token:
(4)         """
(4)         A token in an Excel formula.
(4)         Tokens have three attributes:
(4)         * `value`: The string value parsed that led to this token
(4)         * `type`: A string identifying the type of token
(4)         * `subtype`: A string identifying subtype of the token (optional, and
(17)                     defaults to "")
(4)         """
(4)         __slots__ = ['value', 'type', 'subtype']
(4)         LITERAL = "LITERAL"
(4)         OPERAND = "OPERAND"
(4)         FUNC = "FUNC"
(4)         ARRAY = "ARRAY"
(4)         PAREN = "PAREN"
(4)         SEP = "SEP"
(4)         OP_PRE = "OPERATOR-PREFIX"
(4)         OP_IN = "OPERATOR-INFIX"
(4)         OP_POST = "OPERATOR-POSTFIX"
```

```
(4)                WSPACE = "WHITE-SPACE"
(4)                def __init__(self, value, type_, subtype=""):
(8)                    self.value = value
(8)                    self.type = type_
(8)                    self.subtype = subtype
(4)                TEXT = 'TEXT'
(4)                NUMBER = 'NUMBER'
(4)                LOGICAL = 'LOGICAL'
(4)                ERROR = 'ERROR'
(4)                RANGE = 'RANGE'
(4)                def __repr__(self):
(8)                    return u"{0} {1} {2}:".format(self.type, self.subtype, self.value)
(4)                @classmethod
(4)                def make_operand(cls, value):
(8)                    """Create an operand token."""
(8)                    if value.startswith('"'):
(12)                       subtype = cls.TEXT
(8)                    elif value.startswith('#'):
(12)                       subtype = cls.ERROR
(8)                    elif value in ('TRUE', 'FALSE'):
(12)                       subtype = cls.LOGICAL
(8)                    else:
(12)                       try:
(16)                           float(value)
(16)                           subtype = cls.NUMBER
(12)                       except ValueError:
(16)                           subtype = cls.RANGE
(8)                    return cls(value, cls.OPERAND, subtype)
(4)                OPEN = "OPEN"
(4)                CLOSE = "CLOSE"
(4)                @classmethod
(4)                def make_subexp(cls, value, func=False):
(8)                    """
(8)                    Create a subexpression token.
(8)                    `value`: The value of the token
(8)                    `func`: If True, force the token to be of type FUNC
(8)                    """
(8)                    assert value[-1] in ('{', '}', '(', ')')
(8)                    if func:
(12)                       assert re.match('.+\\(|\\)', value)
(12)                       type_ = Token.FUNC
(8)                    elif value in '{}':
(12)                       type_ = Token.ARRAY
(8)                    elif value in '()':
(12)                       type_ = Token.PAREN
(8)                    else:
(12)                       type_ = Token.FUNC
(8)                    subtype = cls.CLOSE if value in ')}' else cls.OPEN
(8)                    return cls(value, type_, subtype)
(4)                def get_closer(self):
(8)                    """Return a closing token that matches this token's type."""
(8)                    assert self.type in (self.FUNC, self.ARRAY, self.PAREN)
(8)                    assert self.subtype == self.OPEN
(8)                    value = "}" if self.type == self.ARRAY else ")"
(8)                    return self.make_subexp(value, func=self.type == self.FUNC)
(4)                ARG = "ARG"
(4)                ROW = "ROW"
(4)                @classmethod
(4)                def make_separator(cls, value):
(8)                    """Create a separator token"""
(8)                    assert value in (',', ';')
(8)                    subtype = cls.ARG if value == ',' else cls.ROW
(8)                    return cls(value, cls.SEP, subtype)
```

File 107 - translate.py:

```
(0)                """
```

```
This module contains code to translate formulae across cells in a worksheet.
The idea is that if A1 has formula "=B1+C1", then translating it to cell A2
results in formula "=B2+C2". The algorithm relies on the formula tokenizer
to identify the parts of the formula that need to change.
"""
import re
from .tokenizer import Tokenizer, Token
from openpyxl.utils import (
    coordinate_to_tuple,
    column_index_from_string,
    get_column_letter
)
class TranslatorError(Exception):
    """
    Raised when a formula can't be translated across cells.
    This error arises when a formula's references would be translated outside
    the worksheet's bounds on the top or left. Excel represents these
    situations with a #REF! literal error. E.g., if the formula at B2 is
    '=A1', attempting to translate the formula to B1 raises TranslatorError,
    since there's no cell above A1. Similarly, translating the same formula
    from B2 to A2 raises TranslatorError, since there's no cell to the left of
    A1.
    """
class Translator:
    """
    Modifies a formula so that it can be translated from one cell to another.
    `formula`: The str string to translate. Must include the leading '='
               character.
    `origin`: The cell address (in A1 notation) where this formula was
              defined (excluding the worksheet name).
    """
    def __init__(self, formula, origin):
        self.row, self.col = coordinate_to_tuple(origin)
        self.tokenizer = Tokenizer(formula)
    def get_tokens(self):
        "Returns a list with the tokens comprising the formula."
        return self.tokenizer.items
    ROW_RANGE_RE = re.compile(r"(\$?[1-9][0-9]{0,6}):(\$?[1-9][0-9]{0,6})$")
    COL_RANGE_RE = re.compile(r"(\$?[A-Za-z]{1,3}):(\$?[A-Za-z]{1,3})$")
    CELL_REF_RE = re.compile(r"(\$?[A-Za-z]{1,3})(\$?[1-9][0-9]{0,6})$")
    @staticmethod
    def translate_row(row_str, rdelta):
        """
        Translate a range row-snippet by the given number of rows.
        """
        if row_str.startswith('$'):
            return row_str
        else:
            new_row = int(row_str) + rdelta
            if new_row <= 0:
                raise TranslatorError("Formula out of range")
            return str(new_row)
    @staticmethod
    def translate_col(col_str, cdelta):
        """
        Translate a range col-snippet by the given number of columns
        """
        if col_str.startswith('$'):
            return col_str
        else:
            try:
                return get_column_letter(
                    column_index_from_string(col_str) + cdelta)
            except ValueError:
                raise TranslatorError("Formula out of range")
    @staticmethod
    def strip_ws_name(range_str):
        "Splits out the worksheet reference, if any, from a range reference."
        if '!' in range_str:
```

```
(12)                    sheet, range_str = range_str.rsplit('!', 1)
(12)                    return sheet + "!", range_str
(8)                 return "", range_str
(4)             @classmethod
(4)             def translate_range(cls, range_str, rdelta, cdelta):
(8)                 """
(8)                 Translate an A1-style range reference to the destination cell.
(8)                 `rdelta`: the row offset to add to the range
(8)                 `cdelta`: the column offset to add to the range
(8)                 `range_str`: an A1-style reference to a range. Potentially includes
(21)                             the worksheet reference. Could also be a named range.
(8)                 """
(8)                 ws_part, range_str = cls.strip_ws_name(range_str)
(8)                 match = cls.ROW_RANGE_RE.match(range_str)  # e.g. `3:4`
(8)                 if match is not None:
(12)                    return (ws_part + cls.translate_row(match.group(1), rdelta) + ":"
(20)                            + cls.translate_row(match.group(2), rdelta))
(8)                 match = cls.COL_RANGE_RE.match(range_str)  # e.g. `A:BC`
(8)                 if match is not None:
(12)                    return (ws_part + cls.translate_col(match.group(1), cdelta) + ':'
(20)                            + cls.translate_col(match.group(2), cdelta))
(8)                 if ':' in range_str: # e.g. `A1:B5`
(12)                    return ws_part + ":".join(
(16)                        cls.translate_range(piece, rdelta, cdelta)
(16)                        for piece in range_str.split(':'))
(8)                 match = cls.CELL_REF_RE.match(range_str)
(8)                 if match is None:  # Must be a named range
(12)                    return range_str
(8)                 return (ws_part + cls.translate_col(match.group(1), cdelta)
(16)                        + cls.translate_row(match.group(2), rdelta))
(4)             def translate_formula(self, dest=None, row_delta=0, col_delta=0):
(8)                 """
(8)                 Convert the formula into A1 notation, or as row and column coordinates
(8)                 The formula is converted into A1 assuming it is assigned to the cell
(8)                 whose address is `dest` (no worksheet name).
(8)                 """
(8)                 tokens = self.get_tokens()
(8)                 if not tokens:
(12)                    return ""
(8)                 elif tokens[0].type == Token.LITERAL:
(12)                    return tokens[0].value
(8)                 out = ['=']
(8)                 if dest:
(12)                    row, col = coordinate_to_tuple(dest)
(12)                    row_delta = row - self.row
(12)                    col_delta = col - self.col
(8)                 for token in tokens:
(12)                    if (token.type == Token.OPERAND
(16)                        and token.subtype == Token.RANGE):
(16)                        out.append(self.translate_range(token.value, row_delta,
(48)                                                        col_delta))
(12)                    else:
(16)                        out.append(token.value)
(8)                 return "".join(out)
```

File 108 - interface.py:

```
(0)         from abc import abstractproperty
(0)         from openpyxl.compat.abc import ABC
(0)         class ISerialisableFile(ABC):
(4)             """
(4)             Interface for Serialisable classes that represent files in the archive
(4)             """
(4)             @abstractproperty
(4)             def id(self):
(8)                 """
(8)                 Object id making it unique
```

```
11: (8)                     """
12: (8)                     pass
13: (4)                 @abstractproperty
14: (4)                 def _path(self):
15: (8)                     """
16: (8)                     File path in the archive
17: (8)                     """
18: (8)                     pass
19: (4)                 @abstractproperty
20: (4)                 def _namespace(self):
21: (8)                     """
22: (8)                     Qualified namespace when serialised
23: (8)                     """
24: (8)                     pass
25: (4)                 @abstractproperty
26: (4)                 def _type(self):
27: (8)                     """
28: (8)                     The content type for the manifest
29: (8)                     """
30: (4)                 @abstractproperty
31: (4)                 def _rel_type(self):
32: (8)                     """
33: (8)                     The content type for relationships
34: (8)                     """
35: (4)                 @abstractproperty
36: (4)                 def _rel_id(self):
37: (8)                     """
38: (8)                     Links object with parent
39: (8)                     """
```

File 109 - relationship.py:

```
1: (0)               import posixpath
2: (0)               from warnings import warn
3: (0)               from openpyxl.descriptors import (
4: (4)                   String,
5: (4)                   Alias,
6: (4)                   Sequence,
7: (0)               )
8: (0)               from openpyxl.descriptors.serialisable import Serialisable
9: (0)               from openpyxl.descriptors.container import ElementList
10: (0)              from openpyxl.xml.constants import REL_NS, PKG_REL_NS
11: (0)              from openpyxl.xml.functions import (
12: (4)                  Element,
13: (4)                  fromstring,
14: (0)              )
15: (0)              class Relationship(Serialisable):
16: (4)                  """Represents many kinds of relationships."""
17: (4)                  tagname = "Relationship"
18: (4)                  Type = String()
19: (4)                  Target = String()
20: (4)                  target = Alias("Target")
21: (4)                  TargetMode = String(allow_none=True)
22: (4)                  Id = String(allow_none=True)
23: (4)                  id = Alias("Id")
24: (4)                  def __init__(self,
25: (17)                              Id=None,
26: (17)                              Type=None,
27: (17)                              type=None,
28: (17)                              Target=None,
29: (17)                              TargetMode=None
30: (17)                              ):
31: (8)                      """
32: (8)                      `type` can be used as a shorthand with the default relationships
namespace
33: (8)                      otherwise the `Type` must be a fully qualified URL
34: (8)                      """
```

```
(8)                 if type is not None:
(12)                    Type = "{0}/{1}".format(REL_NS, type)
(8)                 self.Type = Type
(8)                 self.Target = Target
(8)                 self.TargetMode = TargetMode
(8)                 self.Id = Id
(0)         class RelationshipList(ElementList):
(4)             tagname = "Relationships"
(4)             expected_type = Relationship
(4)             def append(self, value):
(8)                 super().append(value)
(8)                 if not value.Id:
(12)                    value.Id = f"rId{len(self)}"
(4)             def find(self, content_type):
(8)                 """
(8)                 Find relationships by content-type
(8)                 NB. these content-types namespaced objects and different to the MIME-
types
(8)                 in the package manifest :-(
(8)                 """
(8)                 for r in self:
(12)                    if r.Type == content_type:
(16)                        yield r
(4)             def get(self, key):
(8)                 for r in self:
(12)                    if r.Id == key:
(16)                        return r
(8)                 raise KeyError("Unknown relationship: {0}".format(key))
(4)             def to_dict(self):
(8)                 """Return a dictionary of relations keyed by id"""
(8)                 return {r.id:r for r in self}
(4)             def to_tree(self):
(8)                 tree = super().to_tree()
(8)                 tree.set("xmlns", PKG_REL_NS)
(8)                 return tree
(0)         def get_rels_path(path):
(4)             """
(4)             Convert relative path to absolutes that can be loaded from a zip
(4)             archive.
(4)             The path to be passed in is that of containing object (workbook,
(4)             worksheet, etc.)
(4)             """
(4)             folder, obj = posixpath.split(path)
(4)             filename = posixpath.join(folder, '_rels', '{0}.rels'.format(obj))
(4)             return filename
(0)         def get_dependents(archive, filename):
(4)             """
(4)             Normalise dependency file paths to absolute ones
(4)             Relative paths are relative to parent object
(4)             """
(4)             src = archive.read(filename)
(4)             node = fromstring(src)
(4)             try:
(8)                 rels = RelationshipList.from_tree(node)
(4)             except TypeError:
(8)                 msg = "{0} contains invalid dependency definitions".format(filename)
(8)                 warn(msg)
(8)                 rels = RelationshipList()
(4)             folder = posixpath.dirname(filename)
(4)             parent = posixpath.split(folder)[0]
(4)             for r in rels:
(8)                 if r.TargetMode == "External":
(12)                    continue
(8)                 elif r.target.startswith("/"):
(12)                    r.target = r.target[1:]
(8)                 else:
(12)                    pth = posixpath.join(parent, r.target)
(12)                    r.target = posixpath.normpath(pth)
(4)             return rels
```

```
103: (0)            def get_rel(archive, deps, id=None, cls=None):
104: (4)                """
105: (4)                Get related object based on id or rel_type
106: (4)                """
107: (4)                if not any([id, cls]):
108: (8)                    raise ValueError("Either the id or the content type are required")
109: (4)                if id is not None:
110: (8)                    rel = deps.get(id)
111: (4)                else:
112: (8)                    try:
113: (12)                       rel = next(deps.find(cls.rel_type))
114: (8)                    except StopIteration: # no known dependency
115: (12)                       return
116: (4)                path = rel.target
117: (4)                src = archive.read(path)
118: (4)                tree = fromstring(src)
119: (4)                obj = cls.from_tree(tree)
120: (4)                rels_path = get_rels_path(path)
121: (4)                try:
122: (8)                    obj.deps = get_dependents(archive, rels_path)
123: (4)                except KeyError:
124: (8)                    obj.deps = []
125: (4)                return obj

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

File 110 - fills.py:

1: (0)              from openpyxl.descriptors import (
2: (4)                  Float,
3: (4)                  Set,
4: (4)                  Alias,
5: (4)                  NoneSet,
6: (4)                  Sequence,
7: (4)                  Integer,
8: (4)                  MinMax,
9: (0)              )
10: (0)             from openpyxl.descriptors.serialisable import Serialisable
11: (0)             from openpyxl.compat import safe_string
12: (0)             from .colors import ColorDescriptor, Color
13: (0)             from openpyxl.xml.functions import Element, localname
14: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
15: (0)             FILL_NONE = 'none'
16: (0)             FILL_SOLID = 'solid'
17: (0)             FILL_PATTERN_DARKDOWN = 'darkDown'
18: (0)             FILL_PATTERN_DARKGRAY = 'darkGray'
19: (0)             FILL_PATTERN_DARKGRID = 'darkGrid'
20: (0)             FILL_PATTERN_DARKHORIZONTAL = 'darkHorizontal'
21: (0)             FILL_PATTERN_DARKTRELLIS = 'darkTrellis'
22: (0)             FILL_PATTERN_DARKUP = 'darkUp'
23: (0)             FILL_PATTERN_DARKVERTICAL = 'darkVertical'
24: (0)             FILL_PATTERN_GRAY0625 = 'gray0625'
25: (0)             FILL_PATTERN_GRAY125 = 'gray125'
26: (0)             FILL_PATTERN_LIGHTDOWN = 'lightDown'
27: (0)             FILL_PATTERN_LIGHTGRAY = 'lightGray'
28: (0)             FILL_PATTERN_LIGHTGRID = 'lightGrid'
29: (0)             FILL_PATTERN_LIGHTHORIZONTAL = 'lightHorizontal'
30: (0)             FILL_PATTERN_LIGHTTRELLIS = 'lightTrellis'
31: (0)             FILL_PATTERN_LIGHTUP = 'lightUp'
32: (0)             FILL_PATTERN_LIGHTVERTICAL = 'lightVertical'
33: (0)             FILL_PATTERN_MEDIUMGRAY = 'mediumGray'
34: (0)             fills = (FILL_SOLID, FILL_PATTERN_DARKDOWN, FILL_PATTERN_DARKGRAY,
35: (9)                      FILL_PATTERN_DARKGRID, FILL_PATTERN_DARKHORIZONTAL,
FILL_PATTERN_DARKTRELLIS,
36: (9)                      FILL_PATTERN_DARKUP, FILL_PATTERN_DARKVERTICAL,
FILL_PATTERN_GRAY0625,
37: (9)                      FILL_PATTERN_GRAY125, FILL_PATTERN_LIGHTDOWN, FILL_PATTERN_LIGHTGRAY,
38: (9)                      FILL_PATTERN_LIGHTGRID, FILL_PATTERN_LIGHTHORIZONTAL,
39: (9)                      FILL_PATTERN_LIGHTTRELLIS, FILL_PATTERN_LIGHTUP,
FILL_PATTERN_LIGHTVERTICAL,
40: (9)                      FILL_PATTERN_MEDIUMGRAY)
41: (0)             class Fill(Serialisable):
42: (4)                 """Base class"""
43: (4)                 tagname = "fill"
44: (4)                 @classmethod
45: (4)                 def from_tree(cls, el):
46: (8)                     children = [c for c in el]
47: (8)                     if not children:
48: (12)                        return
49: (8)                     child = children[0]
50: (8)                     if "patternFill" in child.tag:
51: (12)                        return PatternFill._from_tree(child)
52: (8)                     return super(Fill, GradientFill).from_tree(child)
53: (0)             class PatternFill(Fill):
54: (4)                 """Area fill patterns for use in styles.
55: (4)                 Caution: if you do not specify a fill_type, other attributes will have
56: (4)                 no effect !"""
57: (4)                 tagname = "patternFill"
58: (4)                 __elements__ = ('fgColor', 'bgColor')
59: (4)                 patternType = NoneSet(values=fills)
60: (4)                 fill_type = Alias("patternType")
61: (4)                 fgColor = ColorDescriptor()
62: (4)                 start_color = Alias("fgColor")
63: (4)                 bgColor = ColorDescriptor()
64: (4)                 end_color = Alias("bgColor")
65: (4)                 def __init__(self, patternType=None, fgColor=Color(), bgColor=Color(),
66: (17)                             fill_type=None, start_color=None, end_color=None):
67: (8)                     if fill_type is not None:
68: (12)                        patternType = fill_type
69: (8)                     self.patternType = patternType
70: (8)                     if start_color is not None:
71: (12)                        fgColor = start_color
72: (8)                     self.fgColor = fgColor
73: (8)                     if end_color is not None:
74: (12)                        bgColor = end_color
75: (8)                     self.bgColor = bgColor
76: (4)                 @classmethod
77: (4)                 def _from_tree(cls, el):
78: (8)                     attrib = dict(el.attrib)
79: (8)                     for child in el:
80: (12)                        desc = localname(child)
81: (12)                        attrib[desc] = Color.from_tree(child)
82: (8)                     return cls(**attrib)
83: (4)                 def to_tree(self, tagname=None, idx=None):
84: (8)                     parent = Element("fill")
85: (8)                     el = Element(self.tagname)
86: (8)                     if self.patternType is not None:
87: (12)                        el.set('patternType', self.patternType)
88: (8)                     for c in self.__elements__:
89: (12)                        value = getattr(self, c)
90: (12)                        if value != Color():
91: (16)                            el.append(value.to_tree(c))
92: (8)                     parent.append(el)
93: (8)                     return parent
94: (0)             DEFAULT_EMPTY_FILL = PatternFill()
95: (0)             DEFAULT_GRAY_FILL = PatternFill(patternType='gray125')
96: (0)             class Stop(Serialisable):
97: (4)                 tagname = "stop"
98: (4)                 position = MinMax(min=0, max=1)
99: (4)                 color = ColorDescriptor()
100: (4)                def __init__(self, color, position):
101: (8)                    self.position = position
102: (8)                    self.color = color
103: (0)            def _assign_position(values):
104: (4)                """
105: (4)                Automatically assign positions if a list of colours is provided.
106: (4)                It is not permitted to mix colours and stops
107: (4)                """
```

```
108: (4)                n_values = len(values)
109: (4)                n_stops = sum(isinstance(value, Stop) for value in values)
110: (4)                if n_stops == 0:
111: (8)                    interval = 1
112: (8)                    if n_values > 2:
113: (12)                       interval = 1 / (n_values - 1)
114: (8)                    values = [Stop(value, i * interval)
115: (18)                             for i, value in enumerate(values)]
116: (4)                elif n_stops < n_values:
117: (8)                    raise ValueError('Cannot interpret mix of Stops and Colors in
GradientFill')
118: (4)                pos = set()
119: (4)                for stop in values:
120: (8)                    if stop.position in pos:
121: (12)                       raise ValueError("Duplicate position {0}".format(stop.position))
122: (8)                    pos.add(stop.position)
123: (4)                return values
124: (0)            class StopList(Sequence):
125: (4)                expected_type = Stop
126: (4)                def __set__(self, obj, values):
127: (8)                    values = _assign_position(values)
128: (8)                    super().__set__(obj, values)
129: (0)            class GradientFill(Fill):
130: (4)                """Fill areas with gradient
131: (4)                Two types of gradient fill are supported:
132: (8)                    - A type='linear' gradient interpolates colours between
133: (10)                     a set of specified Stops, across the length of an area.
134: (10)                     The gradient is left-to-right by default, but this
135: (10)                     orientation can be modified with the degree
136: (10)                     attribute.  A list of Colors can be provided instead
137: (10)                     and they will be positioned with equal distance between them.
138: (8)                    - A type='path' gradient applies a linear gradient from each
139: (10)                     edge of the area. Attributes top, right, bottom, left specify
140: (10)                     the extent of fill from the respective borders. Thus top="0.2"
141: (10)                     will fill the top 20% of the cell.
142: (4)                """
143: (4)                tagname = "gradientFill"
144: (4)                type = Set(values=('linear', 'path'))
145: (4)                fill_type = Alias("type")
146: (4)                degree = Float()
147: (4)                left = Float()
148: (4)                right = Float()
149: (4)                top = Float()
150: (4)                bottom = Float()
151: (4)                stop = StopList()
152: (4)                def __init__(self, type="linear", degree=0, left=0, right=0, top=0,
153: (17)                            bottom=0, stop=()):
154: (8)                    self.degree = degree
155: (8)                    self.left = left
156: (8)                    self.right = right
157: (8)                    self.top = top
158: (8)                    self.bottom = bottom
159: (8)                    self.stop = stop
160: (8)                    self.type = type
161: (4)                def __iter__(self):
162: (8)                    for attr in self.__attrs__:
163: (12)                       value = getattr(self, attr)
164: (12)                       if value:
165: (16)                           yield attr, safe_string(value)
166: (4)                def to_tree(self, tagname=None, namespace=None, idx=None):
167: (8)                    parent = Element("fill")
168: (8)                    el = super().to_tree()
169: (8)                    parent.append(el)
170: (8)                    return parent

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

File 111 - fonts.py:
```

```python
from openpyxl.descriptors import (
    Alias,
    Sequence,
    Integer
)
from openpyxl.descriptors.serialisable import Serialisable
from openpyxl.descriptors.nested import (
    NestedValue,
    NestedBool,
    NestedNoneSet,
    NestedMinMax,
    NestedString,
    NestedInteger,
    NestedFloat,
)
from .colors import ColorDescriptor, Color, BLACK
from openpyxl.compat import safe_string
from openpyxl.xml.functions import Element, SubElement
from openpyxl.xml.constants import SHEET_MAIN_NS
def _no_value(tagname, value, namespace=None):
    if value:
        return Element(tagname, val=safe_string(value))
class Font(Serialisable):
    """Font options used in styles."""
    UNDERLINE_DOUBLE = 'double'
    UNDERLINE_DOUBLE_ACCOUNTING = 'doubleAccounting'
    UNDERLINE_SINGLE = 'single'
    UNDERLINE_SINGLE_ACCOUNTING = 'singleAccounting'
    name = NestedString(allow_none=True)
    charset = NestedInteger(allow_none=True)
    family = NestedMinMax(min=0, max=14, allow_none=True)
    sz = NestedFloat(allow_none=True)
    size = Alias("sz")
    b = NestedBool(to_tree=_no_value)
    bold = Alias("b")
    i = NestedBool(to_tree=_no_value)
    italic = Alias("i")
    strike = NestedBool(allow_none=True)
    strikethrough = Alias("strike")
    outline = NestedBool(allow_none=True)
    shadow = NestedBool(allow_none=True)
    condense = NestedBool(allow_none=True)
    extend = NestedBool(allow_none=True)
    u = NestedNoneSet(values=('single', 'double', 'singleAccounting',
                             'doubleAccounting'))
    underline = Alias("u")
    vertAlign = NestedNoneSet(values=('superscript', 'subscript', 'baseline'))
    color = ColorDescriptor(allow_none=True)
    scheme = NestedNoneSet(values=("major", "minor"))
    tagname = "font"
    __elements__ = ('name', 'charset', 'family', 'b', 'i', 'strike', 'outline',
                  'shadow', 'condense', 'color', 'extend', 'sz', 'u', 'vertAlign',
                  'scheme')
    def __init__(self, name=None, sz=None, b=None, i=None, charset=None,
                 u=None, strike=None, color=None, scheme=None, family=None, size=None,
                 bold=None, italic=None, strikethrough=None, underline=None,
                 vertAlign=None, outline=None, shadow=None, condense=None,
                 extend=None):
        self.name = name
        self.family = family
        if size is not None:
            sz = size
        self.sz = sz
        if bold is not None:
            b = bold
        self.b = b
```

```
67: (8)                         if italic is not None:
68: (12)                            i = italic
69: (8)                         self.i = i
70: (8)                         if underline is not None:
71: (12)                            u = underline
72: (8)                         self.u = u
73: (8)                         if strikethrough is not None:
74: (12)                            strike = strikethrough
75: (8)                         self.strike = strike
76: (8)                         self.color = color
77: (8)                         self.vertAlign = vertAlign
78: (8)                         self.charset = charset
79: (8)                         self.outline = outline
80: (8)                         self.shadow = shadow
81: (8)                         self.condense = condense
82: (8)                         self.extend = extend
83: (8)                         self.scheme = scheme
84: (4)                     @classmethod
85: (4)                     def from_tree(cls, node):
86: (8)                         """
87: (8)                         Set default value for underline if child element is present
88: (8)                         """
89: (8)                         underline = node.find("{%s}u" % SHEET_MAIN_NS)
90: (8)                         if underline is not None and underline.get('val') is None:
91: (12)                            underline.set("val", "single")
92: (8)                         return super().from_tree(node)
93: (0)                 DEFAULT_FONT = Font(name="Calibri", sz=11, family=2, b=False, i=False,
94: (20)                                    color=Color(theme=1), scheme="minor")

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

File 112 - proxy.py:

1: (0)                  from copy import copy
2: (0)                  from openpyxl.compat import deprecated
3: (0)                  class StyleProxy:
4: (4)                      """
5: (4)                      Proxy formatting objects so that they cannot be altered
6: (4)                      """
7: (4)                      __slots__ = ('__target')
8: (4)                      def __init__(self, target):
9: (8)                          self.__target = target
10: (4)                     def __repr__(self):
11: (8)                         return repr(self.__target)
12: (4)                     def __getattr__(self, attr):
13: (8)                         return getattr(self.__target, attr)
14: (4)                     def __setattr__(self, attr, value):
15: (8)                         if attr != "_StyleProxy__target":
16: (12)                            raise AttributeError("Style objects are immutable and cannot be
changed."
17: (33)                                                 "Reassign the style with a copy")
18: (8)                         super().__setattr__(attr, value)
19: (4)                     def __copy__(self):
20: (8)                         """
21: (8)                         Return a copy of the proxied object.
22: (8)                         """
23: (8)                         return copy(self.__target)
24: (4)                     def __add__(self, other):
25: (8)                         """
26: (8)                         Add proxied object to another instance and return the combined object
27: (8)                         """
28: (8)                         return self.__target + other
29: (4)                     @deprecated("Use copy(obj) or cell.obj = cell.obj + other")
30: (4)                     def copy(self, **kw):
31: (8)                         """Return a copy of the proxied object. Keyword args will be passed
through"""
32: (8)                         cp = copy(self.__target)
33: (8)                         for k, v in kw.items():
34: (12)                            setattr(cp, k, v)
```

```
35: (8)                 return cp
36: (4)             def __eq__(self, other):
37: (8)                 return self.__target == other
38: (4)             def __ne__(self, other):
39: (8)                 return not self == other
```

File 113 - colors.py:

```
1: (0)          import re
2: (0)          from openpyxl.compat import safe_string
3: (0)          from openpyxl.descriptors import (
4: (4)              String,
5: (4)              Bool,
6: (4)              MinMax,
7: (4)              Integer,
8: (4)              Typed,
9: (0)          )
10: (0)         from openpyxl.descriptors.sequence import NestedSequence
11: (0)         from openpyxl.descriptors.serialisable import Serialisable
12: (0)         COLOR_INDEX = (
13: (4)             '00000000', '00FFFFFF', '00FF0000', '0000FF00', '000000FF', #0-4
14: (4)             '00FFFF00', '00FF00FF', '0000FFFF', '00000000', '00FFFFFF', #5-9
15: (4)             '00FF0000', '0000FF00', '000000FF', '00FFFF00', '00FF00FF', #10-14
16: (4)             '0000FFFF', '00800000', '00008000', '00000080', '00808000', #15-19
17: (4)             '00800080', '00008080', '00C0C0C0', '00808080', '009999FF', #20-24
18: (4)             '00993366', '00FFFFCC', '00CCFFFF', '00660066', '00FF8080', #25-29
19: (4)             '000066CC', '00CCCCFF', '00000080', '00FF00FF', '00FFFF00', #30-34
20: (4)             '0000FFFF', '00800080', '00800000', '00008080', '000000FF', #35-39
21: (4)             '0000CCFF', '00CCFFFF', '00CCFFCC', '00FFFF99', '0099CCFF', #40-44
22: (4)             '00FF99CC', '00CC99FF', '00FFCC99', '003366FF', '0033CCCC', #45-49
23: (4)             '0099CC00', '00FFCC00', '00FF9900', '00FF6600', '00666699', #50-54
24: (4)             '00969696', '00003366', '00339966', '00003300', '00333300', #55-59
25: (4)             '00993300', '00993366', '00333399', '00333333',  #60-63
26: (0)         )
27: (0)         BLACK = COLOR_INDEX[0]
28: (0)         WHITE = COLOR_INDEX[1]
29: (0)         BLUE = COLOR_INDEX[4]
30: (0)         aRGB_REGEX = re.compile("^([A-Fa-f0-9]{8}|[A-Fa-f0-9]{6})$")
31: (0)         class RGB(Typed):
32: (4)             """
33: (4)             Descriptor for aRGB values
34: (4)             If not supplied alpha is 00
35: (4)             """
36: (4)             expected_type = str
37: (4)             def __set__(self, instance, value):
38: (8)                 if not self.allow_none:
39: (12)                    m = aRGB_REGEX.match(value)
40: (12)                    if m is None:
41: (16)                        raise ValueError("Colors must be aRGB hex values")
42: (12)                    if len(value) == 6:
43: (16)                        value = "00" + value
44: (8)                 super().__set__(instance, value)
45: (0)         class Color(Serialisable):
46: (4)             """Named colors for use in styles."""
47: (4)             tagname = "color"
48: (4)             rgb = RGB()
49: (4)             indexed = Integer()
50: (4)             auto = Bool()
51: (4)             theme = Integer()
52: (4)             tint = MinMax(min=-1, max=1, expected_type=float)
53: (4)             type = String()
54: (4)             def __init__(self, rgb=BLACK, indexed=None, auto=None, theme=None,
tint=0.0, index=None, type='rgb'):
55: (8)                 if index is not None:
56: (12)                    indexed = index
57: (8)                 if indexed is not None:
58: (12)                    self.type = 'indexed'
```

```
(12)                    self.indexed = indexed
(8)                 elif theme is not None:
(12)                    self.type = 'theme'
(12)                    self.theme = theme
(8)                 elif auto is not None:
(12)                    self.type = 'auto'
(12)                    self.auto = auto
(8)                 else:
(12)                    self.rgb = rgb
(12)                    self.type = 'rgb'
(8)                 self.tint = tint
(4)             @property
(4)             def value(self):
(8)                 return getattr(self, self.type)
(4)             @value.setter
(4)             def value(self, value):
(8)                 setattr(self, self.type, value)
(4)             def __iter__(self):
(8)                 attrs = [(self.type, self.value)]
(8)                 if self.tint != 0:
(12)                    attrs.append(('tint', self.tint))
(8)                 for k, v in attrs:
(12)                    yield k, safe_string(v)
(4)             @property
(4)             def index(self):
(8)                 return self.value
(4)             def __add__(self, other):
(8)                 """
(8)                 Adding colours is undefined behaviour best do nothing
(8)                 """
(8)                 if not isinstance(other, Color):
(12)                    return super().__add__(other)
(8)                 return self
(0)         class ColorDescriptor(Typed):
(4)             expected_type = Color
(4)             def __set__(self, instance, value):
(8)                 if isinstance(value, str):
(12)                    value = Color(rgb=value)
(8)                 super().__set__(instance, value)
(0)         class RgbColor(Serialisable):
(4)             tagname = "rgbColor"
(4)             rgb = RGB()
(4)             def __init__(self,
(17)                         rgb=None,
(16)                        ):
(8)                 self.rgb = rgb
(0)         class ColorList(Serialisable):
(4)             tagname = "colors"
(4)             indexedColors = NestedSequence(expected_type=RgbColor)
(4)             mruColors = NestedSequence(expected_type=Color)
(4)             __elements__ = ('indexedColors', 'mruColors')
(4)             def __init__(self,
(17)                         indexedColors=(),
(17)                         mruColors=(),
(16)                        ):
(8)                 self.indexedColors = indexedColors
(8)                 self.mruColors = mruColors
(4)             def __bool__(self):
(8)                 return bool(self.indexedColors) or bool(self.mruColors)
(4)             @property
(4)             def index(self):
(8)                 return [val.rgb for val in self.indexedColors]
```

File 114 - borders.py:

```
(0)         from openpyxl.compat import safe_string
(0)         from openpyxl.descriptors import (
3: (4)                  NoneSet,
4: (4)                  Typed,
5: (4)                  Bool,
6: (4)                  Alias,
7: (4)                  Sequence,
8: (4)                  Integer,
9: (0)               )
10: (0)              from openpyxl.descriptors.serialisable import Serialisable
11: (0)              from .colors import ColorDescriptor
12: (0)              BORDER_NONE = None
13: (0)              BORDER_DASHDOT = 'dashDot'
14: (0)              BORDER_DASHDOTDOT = 'dashDotDot'
15: (0)              BORDER_DASHED = 'dashed'
16: (0)              BORDER_DOTTED = 'dotted'
17: (0)              BORDER_DOUBLE = 'double'
18: (0)              BORDER_HAIR = 'hair'
19: (0)              BORDER_MEDIUM = 'medium'
20: (0)              BORDER_MEDIUMDASHDOT = 'mediumDashDot'
21: (0)              BORDER_MEDIUMDASHDOTDOT = 'mediumDashDotDot'
22: (0)              BORDER_MEDIUMDASHED = 'mediumDashed'
23: (0)              BORDER_SLANTDASHDOT = 'slantDashDot'
24: (0)              BORDER_THICK = 'thick'
25: (0)              BORDER_THIN = 'thin'
26: (0)              class Side(Serialisable):
27: (4)                  """Border options for use in styles.
28: (4)                  Caution: if you do not specify a border_style, other attributes will
29: (4)                  have no effect !"""
30: (4)                  color = ColorDescriptor(allow_none=True)
31: (4)                  style = NoneSet(values=('dashDot','dashDotDot', 'dashed','dotted',
32: (28)                                         'double','hair', 'medium', 'mediumDashDot',
'mediumDashDotDot',
33: (28)                                         'mediumDashed', 'slantDashDot', 'thick', 'thin')
34: (20)                                 )
35: (4)                  border_style = Alias('style')
36: (4)                  def __init__(self, style=None, color=None, border_style=None):
37: (8)                      if border_style is not None:
38: (12)                         style = border_style
39: (8)                      self.style = style
40: (8)                      self.color = color
41: (0)              class Border(Serialisable):
42: (4)                  """Border positioning for use in styles."""
43: (4)                  tagname = "border"
44: (4)                  __elements__ = ('start', 'end', 'left', 'right', 'top', 'bottom',
45: (20)                                 'diagonal', 'vertical', 'horizontal')
46: (4)                  start = Typed(expected_type=Side, allow_none=True)
47: (4)                  end = Typed(expected_type=Side, allow_none=True)
48: (4)                  left = Typed(expected_type=Side, allow_none=True)
49: (4)                  right = Typed(expected_type=Side, allow_none=True)
50: (4)                  top = Typed(expected_type=Side, allow_none=True)
51: (4)                  bottom = Typed(expected_type=Side, allow_none=True)
52: (4)                  diagonal = Typed(expected_type=Side, allow_none=True)
53: (4)                  vertical = Typed(expected_type=Side, allow_none=True)
54: (4)                  horizontal = Typed(expected_type=Side, allow_none=True)
55: (4)                  outline = Bool()
56: (4)                  diagonalUp = Bool()
57: (4)                  diagonalDown = Bool()
58: (4)                  def __init__(self, left=None, right=None, top=None,
59: (17)                              bottom=None, diagonal=None, diagonal_direction=None,
60: (17)                              vertical=None, horizontal=None, diagonalUp=False,
diagonalDown=False,
61: (17)                              outline=True, start=None, end=None):
62: (8)                      self.left = left
63: (8)                      self.right = right
64: (8)                      self.top = top
65: (8)                      self.bottom = bottom
66: (8)                      self.diagonal = diagonal
67: (8)                      self.vertical = vertical
68: (8)                      self.horizontal = horizontal
69: (8)                      self.diagonal_direction = diagonal_direction
```

```
70: (8)                     self.diagonalUp = diagonalUp
71: (8)                     self.diagonalDown = diagonalDown
72: (8)                     self.outline = outline
73: (8)                     self.start = start
74: (8)                     self.end = end
75: (4)                 def __iter__(self):
76: (8)                     for attr in self.__attrs__:
77: (12)                        value = getattr(self, attr)
78: (12)                        if value and attr != "outline":
79: (16)                            yield attr, safe_string(value)
80: (12)                        elif attr == "outline" and not value:
81: (16)                            yield attr, safe_string(value)
82: (0)             DEFAULT_BORDER = Border(left=Side(), right=Side(), top=Side(), bottom=Side(),
diagonal=Side())

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

File 115 - numbers.py:

1: (0)              import re
2: (0)              from openpyxl.descriptors import (
3: (4)                  String,
4: (4)                  Sequence,
5: (4)                  Integer,
6: (0)              )
7: (0)              from openpyxl.descriptors.serialisable import Serialisable
8: (0)              BUILTIN_FORMATS = {
9: (4)                  0: 'General',
10: (4)                 1: '0',
11: (4)                 2: '0.00',
12: (4)                 3: '#,##0',
13: (4)                 4: '#,##0.00',
14: (4)                 5: '"$"#,##0_);("$"#,##0)',
15: (4)                 6: '"$"#,##0_);[Red]("$"#,##0)',
16: (4)                 7: '"$"#,##0.00_);("$"#,##0.00)',
17: (4)                 8: '"$"#,##0.00_);[Red]("$"#,##0.00)',
18: (4)                 9: '0%',
19: (4)                 10: '0.00%',
20: (4)                 11: '0.00E+00',
21: (4)                 12: '# ?/?',
22: (4)                 13: '# ??/??',
23: (4)                 14: 'mm-dd-yy',
24: (4)                 15: 'd-mmm-yy',
25: (4)                 16: 'd-mmm',
26: (4)                 17: 'mmm-yy',
27: (4)                 18: 'h:mm AM/PM',
28: (4)                 19: 'h:mm:ss AM/PM',
29: (4)                 20: 'h:mm',
30: (4)                 21: 'h:mm:ss',
31: (4)                 22: 'm/d/yy h:mm',
32: (4)                 37: '#,##0_);(#,##0)',
33: (4)                 38: '#,##0_);[Red](#,##0)',
34: (4)                 39: '#,##0.00_);(#,##0.00)',
35: (4)                 40: '#,##0.00_);[Red](#,##0.00)',
36: (4)                 41: r'_(* #,##0_);_(* \(#,##0\);_(* "-"_);_(@_)',
37: (4)                 42: r'_("$"* #,##0_);_("$"* \(#,##0\);_("$"* "-"_);_(@_)',
38: (4)                 43: r'_(* #,##0.00_);_(* \(#,##0.00\);_(* "-"??_);_(@_)',
39: (4)                 44: r'_("$"* #,##0.00_)_("$"* \(#,##0.00\)_("$"* "-"??_)_(@_)',
40: (4)                 45: 'mm:ss',
41: (4)                 46: '[h]:mm:ss',
42: (4)                 47: 'mmss.0',
43: (4)                 48: '##0.0E+0',
44: (4)                 49: '@', }
45: (0)             BUILTIN_FORMATS_MAX_SIZE = 164
46: (0)             BUILTIN_FORMATS_REVERSE = dict(
47: (8)                     [(value, key) for key, value in BUILTIN_FORMATS.items()])
48: (0)             FORMAT_GENERAL = BUILTIN_FORMATS[0]
49: (0)             FORMAT_TEXT = BUILTIN_FORMATS[49]
50: (0)             FORMAT_NUMBER = BUILTIN_FORMATS[1]
```

```
(0)             FORMAT_NUMBER_00 = BUILTIN_FORMATS[2]
(0)             FORMAT_NUMBER_COMMA_SEPARATED1 = BUILTIN_FORMATS[4]
(0)             FORMAT_NUMBER_COMMA_SEPARATED2 = '#,##0.00_-'
(0)             FORMAT_PERCENTAGE = BUILTIN_FORMATS[9]
(0)             FORMAT_PERCENTAGE_00 = BUILTIN_FORMATS[10]
(0)             FORMAT_DATE_YYYYMMDD2 = 'yyyy-mm-dd'
(0)             FORMAT_DATE_YYMMDD = 'yy-mm-dd'
(0)             FORMAT_DATE_DDMMYY = 'dd/mm/yy'
(0)             FORMAT_DATE_DMYSLASH = 'd/m/y'
(0)             FORMAT_DATE_DMYMINUS = 'd-m-y'
(0)             FORMAT_DATE_DMMINUS = 'd-m'
(0)             FORMAT_DATE_MYMINUS = 'm-y'
(0)             FORMAT_DATE_XLSX14 = BUILTIN_FORMATS[14]
(0)             FORMAT_DATE_XLSX15 = BUILTIN_FORMATS[15]
(0)             FORMAT_DATE_XLSX16 = BUILTIN_FORMATS[16]
(0)             FORMAT_DATE_XLSX17 = BUILTIN_FORMATS[17]
(0)             FORMAT_DATE_XLSX22 = BUILTIN_FORMATS[22]
(0)             FORMAT_DATE_DATETIME = 'yyyy-mm-dd h:mm:ss'
(0)             FORMAT_DATE_TIME1 = BUILTIN_FORMATS[18]
(0)             FORMAT_DATE_TIME2 = BUILTIN_FORMATS[19]
(0)             FORMAT_DATE_TIME3 = BUILTIN_FORMATS[20]
(0)             FORMAT_DATE_TIME4 = BUILTIN_FORMATS[21]
(0)             FORMAT_DATE_TIME5 = BUILTIN_FORMATS[45]
(0)             FORMAT_DATE_TIME6 = BUILTIN_FORMATS[21]
(0)             FORMAT_DATE_TIME7 = 'i:s.S'
(0)             FORMAT_DATE_TIME8 = 'h:mm:ss@'
(0)             FORMAT_DATE_TIMEDELTA = '[hh]:mm:ss'
(0)             FORMAT_DATE_YYMMDDSLASH = 'yy/mm/dd@'
(0)             FORMAT_CURRENCY_USD_SIMPLE = '"$"#,##0.00_-'
(0)             FORMAT_CURRENCY_USD = '$#,##0_-'
(0)             FORMAT_CURRENCY_EUR_SIMPLE = '[$EUR ]#,##0.00_-'
(0)             COLORS = r"\[(BLACK|BLUE|CYAN|GREEN|MAGENTA|RED|WHITE|YELLOW)\]"
(0)             LITERAL_GROUP = r'".*?"' # anything in quotes
(0)             LOCALE_GROUP = r'\[(?!hh?\]|mm?\]|ss?\])[^\]]*\]' # anything in square brackets, except hours or minutes or seconds
(0)             STRIP_RE = re.compile(f"{LITERAL_GROUP}|{LOCALE_GROUP}")
(0)             TIMEDELTA_RE = re.compile(r'\[hh?\](:mm(:ss(\.0*)?)?)?|\[mm?\](:ss(\.0*)?)?|\[ss?\](\.0*)?', re.I)
(0)             def is_date_format(fmt):
(4)                 if fmt is None:
(8)                     return False
(4)                 fmt = fmt.split(";")[0] # only look at the first format
(4)                 fmt = STRIP_RE.sub("", fmt) # ignore some formats
(4)                 return re.search(r"(?<![_\\])[dmhysDMHYS]", fmt) is not None
(0)             def is_timedelta_format(fmt):
(4)                 if fmt is None:
(8)                     return False
(4)                 fmt = fmt.split(";")[0] # only look at the first format
(4)                 return TIMEDELTA_RE.search(fmt) is not None
(0)             def is_datetime(fmt):
(4)                 """
(4)                 Return date, time or datetime
(4)                 """
(4)                 if not is_date_format(fmt):
(8)                     return
(4)                 DATE = TIME = False
(4)                 if any((x in fmt for x in 'dy')):
(8)                     DATE = True
(4)                 if any((x in fmt for x in 'hs')):
(8)                     TIME = True
(4)                 if DATE and TIME:
(8)                     return "datetime"
(4)                 if DATE:
(8)                     return "date"
(4)                 return "time"
(0)             def is_builtin(fmt):
(4)                 return fmt in BUILTIN_FORMATS.values()
(0)             def builtin_format_code(index):
(4)                 """Return one of the standard format codes by index."""
```

```
(4)              try:
(8)                  fmt = BUILTIN_FORMATS[index]
(4)              except KeyError:
(8)                  fmt = None
(4)              return fmt
(0)          def builtin_format_id(fmt):
(4)              """Return the id of a standard style."""
(4)              return BUILTIN_FORMATS_REVERSE.get(fmt)
(0)          class NumberFormatDescriptor(String):
(4)              def __set__(self, instance, value):
(8)                  if value is None:
(12)                     value = FORMAT_GENERAL
(8)                  super().__set__(instance, value)
(0)          class NumberFormat(Serialisable):
(4)              numFmtId = Integer()
(4)              formatCode = String()
(4)              def __init__(self,
(17)                          numFmtId=None,
(17)                          formatCode=None,
(16)                         ):
(8)                  self.numFmtId = numFmtId
(8)                  self.formatCode = formatCode
(0)          class NumberFormatList(Serialisable):
(4)              count = Integer(allow_none=True)
(4)              numFmt = Sequence(expected_type=NumberFormat)
(4)              __elements__ = ('numFmt',)
(4)              __attrs__ = ("count",)
(4)              def __init__(self,
(17)                          count=None,
(17)                          numFmt=(),
(16)                         ):
(8)                  self.numFmt = numFmt
(4)              @property
(4)              def count(self):
(8)                  return len(self.numFmt)
(4)              def __getitem__(self, idx):
(8)                  return self.numFmt[idx]
```

File 116 - workbook.py:

```
(0)          from warnings import warn
(0)          from openpyxl.xml.functions import fromstring
(0)          from openpyxl.packaging.relationship import (
(4)              get_dependents,
(4)              get_rels_path,
(4)              get_rel,
(0)          )
(0)          from openpyxl.packaging.workbook import WorkbookPackage
(0)          from openpyxl.workbook import Workbook
(0)          from openpyxl.workbook.defined_name import DefinedNameList
(0)          from openpyxl.workbook.external_link.external import read_external_link
(0)          from openpyxl.pivot.cache import CacheDefinition
(0)          from openpyxl.pivot.record import RecordList
(0)          from openpyxl.worksheet.print_settings import PrintTitles, PrintArea
(0)          from openpyxl.utils.datetime import CALENDAR_MAC_1904
(0)          class WorkbookParser:
(4)              _rels = None
(4)              def __init__(self, archive, workbook_part_name, keep_links=True):
(8)                  self.archive = archive
(8)                  self.workbook_part_name = workbook_part_name
(8)                  self.defined_names = DefinedNameList()
(8)                  self.wb = Workbook()
(8)                  self.keep_links = keep_links
(8)                  self.sheets = []
(4)              @property
(4)              def rels(self):
(8)                  if self._rels is None:
```

```
(12)                        self._rels = get_dependents(self.archive, get_rels_path(self.workbook_part_name)).to_dict()
(8)                     return self._rels
(4)                 def parse(self):
(8)                     src = self.archive.read(self.workbook_part_name)
(8)                     node = fromstring(src)
(8)                     package = WorkbookPackage.from_tree(node)
(8)                     if package.properties.date1904:
(12)                        self.wb.epoch = CALENDAR_MAC_1904
(8)                     self.wb.code_name = package.properties.codeName
(8)                     self.wb.active = package.active
(8)                     self.wb.views = package.bookViews
(8)                     self.sheets = package.sheets
(8)                     self.wb.calculation = package.calcPr
(8)                     self.caches = package.pivotCaches
(8)                     if not self.keep_links:
(12)                        package.externalReferences = []
(8)                     for ext_ref in package.externalReferences:
(12)                        rel = self.rels.get(ext_ref.id)
(12)                        self.wb._external_links.append(
(16)                            read_external_link(self.archive, rel.Target)
(12)                        )
(8)                     if package.definedNames:
(12)                        self.defined_names = package.definedNames
(8)                     self.wb.security = package.workbookProtection
(4)                 def find_sheets(self):
(8)                     """
(8)                     Find all sheets in the workbook and return the link to the source file.
(8)                     Older XLSM files sometimes contain invalid sheet elements.
(8)                     Warn user when these are removed.
(8)                     """
(8)                     for sheet in self.sheets:
(12)                        if not sheet.id:
(16)                            msg = f"File contains an invalid specification for {0}. This will be removed".format(sheet.name)
(16)                            warn(msg)
(16)                            continue
(12)                        yield sheet, self.rels[sheet.id]
(4)                 def assign_names(self):
(8)                     """
(8)                     Bind defined names and other definitions to worksheets or the workbook
(8)                     """
(8)                     for idx, names in self.defined_names.by_sheet().items():
(12)                        if idx == "global":
(16)                            self.wb.defined_names = names
(16)                            continue
(12)                        try:
(16)                            sheet = self.wb._sheets[idx]
(12)                        except IndexError:
(16)                            warn(f"Defined names for sheet index {idx} cannot be located")
(16)                            continue
(12)                        for name, defn in names.items():
(16)                            reserved = defn.is_reserved
(16)                            if reserved is None:
(20)                                sheet.defined_names[name] = defn
(16)                            elif reserved == "Print_Titles":
(20)                                titles = PrintTitles.from_string(defn.value)
(20)                                sheet._print_rows = titles.rows
(20)                                sheet._print_cols = titles.cols
(16)                            elif reserved == "Print_Area":
(20)                                try:
(24)                                    sheet._print_area = PrintArea.from_string(defn.value)
(20)                                except TypeError:
(24)                                    warn(f"Print area cannot be set to Defined name: {defn.value}.")
(24)                                    continue
(4)                 @property
(4)                 def pivot_caches(self):
```

```
93: (8)                     """
94: (8)                     Get PivotCache objects
95: (8)                     """
96: (8)                     d = {}
97: (8)                     for c in self.caches:
98: (12)                        cache = get_rel(self.archive, self.rels, id=c.id,
cls=CacheDefinition)
99: (12)                        if cache.deps:
100: (16)                           records = get_rel(self.archive, cache.deps, cache.id,
RecordList)
101: (16)                           cache.records = records
102: (12)                       d[c.cacheId] = cache
103: (8)                    return d

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

File 117 - builtins.py:

1: (0)              from .named_styles import NamedStyle
2: (0)              from openpyxl.xml.functions import fromstring
3: (0)              normal = """
4: (2)                <namedStyle builtinId="0" name="Normal">
5: (4)                  <alignment/>
6: (4)                  <border>
7: (6)                    <left/>
8: (6)                    <right/>
9: (6)                    <top/>
10: (6)                   <bottom/>
11: (6)                   <diagonal/>
12: (4)                 </border>
13: (4)                 <fill>
14: (6)                   <patternFill/>
15: (4)                 </fill>
16: (4)                 <font>
17: (6)                   <name val="Calibri"/>
18: (6)                   <family val="2"/>
19: (6)                   <color theme="1"/>
20: (6)                   <sz val="12"/>
21: (6)                   <scheme val="minor"/>
22: (4)                 </font>
23: (4)                 <protection hidden="0" locked="1"/>
24: (2)               </namedStyle>
25: (0)             """
26: (0)             comma = """
27: (2)               <namedStyle builtinId="3" name="Comma">
28: (4)                 <alignment/>
29: (4)                 <number_format>_-* #,##0.00\\ _$_-;\\-* #,##0.00\\ _$_-;_-* "-"??\\
_$_-;_-@_-</number_format>
30: (4)                 <border>
31: (6)                   <left/>
32: (6)                   <right/>
33: (6)                   <top/>
34: (6)                   <bottom/>
35: (6)                   <diagonal/>
36: (4)                 </border>
37: (4)                 <fill>
38: (6)                   <patternFill/>
39: (4)                 </fill>
40: (4)                 <font>
41: (6)                   <name val="Calibri"/>
42: (6)                   <family val="2"/>
43: (6)                   <color theme="1"/>
44: (6)                   <sz val="12"/>
45: (6)                   <scheme val="minor"/>
46: (4)                 </font>
47: (4)                 <protection hidden="0" locked="1"/>
48: (2)               </namedStyle>
49: (0)             """
50: (0)             comma_0 = """
```

```
  51: (2)                 <namedStyle builtinId="6" name="Comma [0]">
  52: (4)                   <alignment/>
  53: (4)                   <number_format>_-* #,##0\\ _$_-;\\-* #,##0\\ _$_-;_-* "-"\\ _$_-;_-@_-
  </number_format>
  54: (4)                   <border>
  55: (6)                     <left/>
  56: (6)                     <right/>
  57: (6)                     <top/>
  58: (6)                     <bottom/>
  59: (6)                     <diagonal/>
  60: (4)                   </border>
  61: (4)                   <fill>
  62: (6)                     <patternFill/>
  63: (4)                   </fill>
  64: (4)                   <font>
  65: (6)                     <name val="Calibri"/>
  66: (6)                     <family val="2"/>
  67: (6)                     <color theme="1"/>
  68: (6)                     <sz val="12"/>
  69: (6)                     <scheme val="minor"/>
  70: (4)                   </font>
  71: (4)                   <protection hidden="0" locked="1"/>
  72: (2)                 </namedStyle>
  73: (0)               """
  74: (0)               currency = """
  75: (2)                 <namedStyle builtinId="4" name="Currency">
  76: (4)                   <alignment/>
  77: (4)                   <number_format>_-* #,##0.00\\ "$"_-;\\-* #,##0.00\\ "$"_-;_-* "-"??\\
  "$"_-;_-@_-</number_format>
  78: (4)                   <border>
  79: (6)                     <left/>
  80: (6)                     <right/>
  81: (6)                     <top/>
  82: (6)                     <bottom/>
  83: (6)                     <diagonal/>
  84: (4)                   </border>
  85: (4)                   <fill>
  86: (6)                     <patternFill/>
  87: (4)                   </fill>
  88: (4)                   <font>
  89: (6)                     <name val="Calibri"/>
  90: (6)                     <family val="2"/>
  91: (6)                     <color theme="1"/>
  92: (6)                     <sz val="12"/>
  93: (6)                     <scheme val="minor"/>
  94: (4)                   </font>
  95: (4)                   <protection hidden="0" locked="1"/>
  96: (2)                 </namedStyle>
  97: (0)               """
  98: (0)               currency_0 = """
  99: (2)                 <namedStyle builtinId="7" name="Currency [0]">
  100: (4)                  <alignment/>
  101: (4)                  <number_format>_-* #,##0\\ "$"_-;\\-* #,##0\\ "$"_-;_-* "-"\\ "$"_-;_-@_-
  </number_format>
  102: (4)                  <border>
  103: (6)                    <left/>
  104: (6)                    <right/>
  105: (6)                    <top/>
  106: (6)                    <bottom/>
  107: (6)                    <diagonal/>
  108: (4)                  </border>
  109: (4)                  <fill>
  110: (6)                    <patternFill/>
  111: (4)                  </fill>
  112: (4)                  <font>
  113: (6)                    <name val="Calibri"/>
  114: (6)                    <family val="2"/>
  115: (6)                    <color theme="1"/>
  116: (6)                    <sz val="12"/>
```

```
(6)          <scheme val="minor"/>
(4)        </font>
(4)        <protection hidden="0" locked="1"/>
(2)      </namedStyle>
(0)    """
(0)    percent = """
(2)      <namedStyle builtinId="5" name="Percent">
(4)        <alignment/>
(4)        <number_format>0%</number_format>
(4)        <border>
(6)          <left/>
(6)          <right/>
(6)          <top/>
(6)          <bottom/>
(6)          <diagonal/>
(4)        </border>
(4)        <fill>
(6)          <patternFill/>
(4)        </fill>
(4)        <font>
(6)          <name val="Calibri"/>
(6)          <family val="2"/>
(6)          <color theme="1"/>
(6)          <sz val="12"/>
(6)          <scheme val="minor"/>
(4)        </font>
(4)        <protection hidden="0" locked="1"/>
(2)      </namedStyle>
(0)    """
(0)    hyperlink = """
(2)      <namedStyle builtinId="8" name="Hyperlink" >
(4)        <alignment/>
(4)        <border>
(6)          <left/>
(6)          <right/>
(6)          <top/>
(6)          <bottom/>
(6)          <diagonal/>
(4)        </border>
(4)        <fill>
(6)          <patternFill/>
(4)        </fill>
(4)        <font>
(6)          <name val="Calibri"/>
(6)          <family val="2"/>
(6)          <color theme="10"/>
(6)          <sz val="12"/>
(6)          <scheme val="minor"/>
(4)        </font>
(4)        <protection hidden="0" locked="1"/>
(2)      </namedStyle>"""
(0)    followed_hyperlink = """
(2)      <namedStyle builtinId="9" name="Followed Hyperlink" >
(4)        <alignment/>
(4)        <border>
(6)          <left/>
(6)          <right/>
(6)          <top/>
(6)          <bottom/>
(6)          <diagonal/>
(4)        </border>
(4)        <fill>
(6)          <patternFill/>
(4)        </fill>
(4)        <font>
(6)          <name val="Calibri"/>
(6)          <family val="2"/>
(6)          <color theme="11"/>
(6)          <sz val="12"/>
```

```
(6)                 <scheme val="minor"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>"""
(0)           title = """
(2)             <namedStyle builtinId="15" name="Title">
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top/>
(6)                 <bottom/>
(6)                 <diagonal/>
(4)               </border>
(4)               <fill>
(6)                 <patternFill/>
(4)               </fill>
(4)               <font>
(6)                 <name val="Cambria"/>
(6)                 <family val="2"/>
(6)                 <b val="1"/>
(6)                 <color theme="3"/>
(6)                 <sz val="18"/>
(6)                 <scheme val="major"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>
(0)           """
(0)           headline_1 = """
(2)             <namedStyle builtinId="16" name="Headline 1" >
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top/>
(6)                 <bottom style="thick">
(8)                   <color theme="4"/>
(6)                 </bottom>
(6)                 <diagonal/>
(4)               </border>
(4)               <fill>
(6)                 <patternFill/>
(4)               </fill>
(4)               <font>
(6)                 <name val="Calibri"/>
(6)                 <family val="2"/>
(6)                 <b val="1"/>
(6)                 <color theme="3"/>
(6)                 <sz val="15"/>
(6)                 <scheme val="minor"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>
(0)           """
(0)           headline_2 = """
(2)             <namedStyle builtinId="17" name="Headline 2" >
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top/>
(6)                 <bottom style="thick">
(8)                   <color theme="4" tint="0.5"/>
(6)                 </bottom>
(6)                 <diagonal/>
(4)               </border>
(4)               <fill>
(6)                 <patternFill/>
(4)               </fill>
```

```
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <b val="1"/>
(6)                   <color theme="3"/>
(6)                   <sz val="13"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>
(0)             """
(0)             headline_3 = """
(3)                <namedStyle builtinId="18" name="Headline 3" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom style="medium">
(8)                     <color theme="4" tint="0.4"/>
(6)                   </bottom>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill/>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <b val="1"/>
(6)                   <color theme="3"/>
(6)                   <sz val="11"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>
(0)             """
(0)             headline_4 = """
(2)               <namedStyle builtinId="19" name="Headline 4">
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill/>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <b val="1"/>
(6)                   <color theme="3"/>
(6)                   <sz val="11"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>
(0)             """
(0)             good = """
(2)               <namedStyle builtinId="26" name="Good" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
```

```
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor rgb="FFC6EFCE"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color rgb="FF006100"/>
(6)           <sz val="12"/>
(6)           <scheme val="minor"/>
(4)         </font>
(4)         <protection hidden="0" locked="1"/>
(2)       </namedStyle>
(0)     """
(0)     bad = """
(2)       <namedStyle builtinId="27" name="Bad" >
(4)         <alignment/>
(4)         <border>
(6)           <left/>
(6)           <right/>
(6)           <top/>
(6)           <bottom/>
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor rgb="FFFFC7CE"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color rgb="FF9C0006"/>
(6)           <sz val="12"/>
(6)           <scheme val="minor"/>
(4)         </font>
(4)         <protection hidden="0" locked="1"/>
(2)       </namedStyle>
(0)     """
(0)     neutral = """
(2)       <namedStyle builtinId="28" name="Neutral" >
(4)         <alignment/>
(4)         <border>
(6)           <left/>
(6)           <right/>
(6)           <top/>
(6)           <bottom/>
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor rgb="FFFFEB9C"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color rgb="FF9C6500"/>
(6)           <sz val="12"/>
(6)           <scheme val="minor"/>
(4)         </font>
(4)         <protection hidden="0" locked="1"/>
(2)       </namedStyle>
(0)     """
(0)     input = """
(2)       <namedStyle builtinId="20" name="Input" >
```

```
(4)                <alignment/>
(4)                <border>
(6)                  <left style="thin">
(8)                    <color rgb="FF7F7F7F"/>
(6)                  </left>
(6)                  <right style="thin">
(8)                    <color rgb="FF7F7F7F"/>
(6)                  </right>
(6)                  <top style="thin">
(8)                    <color rgb="FF7F7F7F"/>
(6)                  </top>
(6)                  <bottom style="thin">
(8)                    <color rgb="FF7F7F7F"/>
(6)                  </bottom>
(6)                  <diagonal/>
(4)                </border>
(4)                <fill>
(6)                  <patternFill patternType="solid">
(8)                    <fgColor rgb="FFFFCC99"/>
(6)                  </patternFill>
(4)                </fill>
(4)                <font>
(6)                  <name val="Calibri"/>
(6)                  <family val="2"/>
(6)                  <color rgb="FF3F3F76"/>
(6)                  <sz val="12"/>
(6)                  <scheme val="minor"/>
(4)                </font>
(4)                <protection hidden="0" locked="1"/>
(2)              </namedStyle>
(0)            """
(0)            output = """
(2)              <namedStyle builtinId="21" name="Output" >
(4)                <alignment/>
(4)                <border>
(6)                  <left style="thin">
(8)                    <color rgb="FF3F3F3F"/>
(6)                  </left>
(6)                  <right style="thin">
(8)                    <color rgb="FF3F3F3F"/>
(6)                  </right>
(6)                  <top style="thin">
(8)                    <color rgb="FF3F3F3F"/>
(6)                  </top>
(6)                  <bottom style="thin">
(8)                    <color rgb="FF3F3F3F"/>
(6)                  </bottom>
(6)                  <diagonal/>
(4)                </border>
(4)                <fill>
(6)                  <patternFill patternType="solid">
(8)                    <fgColor rgb="FFF2F2F2"/>
(6)                  </patternFill>
(4)                </fill>
(4)                <font>
(6)                  <name val="Calibri"/>
(6)                  <family val="2"/>
(6)                  <b val="1"/>
(6)                  <color rgb="FF3F3F3F"/>
(6)                  <sz val="12"/>
(6)                  <scheme val="minor"/>
(4)                </font>
(4)                <protection hidden="0" locked="1"/>
(2)              </namedStyle>
(0)            """
(0)            calculation = """
(2)              <namedStyle builtinId="22" name="Calculation" >
(4)                <alignment/>
(4)                <border>
```

```
(6)          <left style="thin">
(8)            <color rgb="FF7F7F7F"/>
(6)          </left>
(6)          <right style="thin">
(8)            <color rgb="FF7F7F7F"/>
(6)          </right>
(6)          <top style="thin">
(8)            <color rgb="FF7F7F7F"/>
(6)          </top>
(6)          <bottom style="thin">
(8)            <color rgb="FF7F7F7F"/>
(6)          </bottom>
(6)          <diagonal/>
(4)        </border>
(4)        <fill>
(6)          <patternFill patternType="solid">
(8)            <fgColor rgb="FFF2F2F2"/>
(6)          </patternFill>
(4)        </fill>
(4)        <font>
(6)          <name val="Calibri"/>
(6)          <family val="2"/>
(6)          <b val="1"/>
(6)          <color rgb="FFFA7D00"/>
(6)          <sz val="12"/>
(6)          <scheme val="minor"/>
(4)        </font>
(4)        <protection hidden="0" locked="1"/>
(2)      </namedStyle>
(0)    """
(0)    linked_cell = """
(2)      <namedStyle builtinId="24" name="Linked Cell" >
(4)        <alignment/>
(4)        <border>
(6)          <left/>
(6)          <right/>
(6)          <top/>
(6)          <bottom style="double">
(8)            <color rgb="FFFF8001"/>
(6)          </bottom>
(6)          <diagonal/>
(4)        </border>
(4)        <fill>
(6)          <patternFill/>
(4)        </fill>
(4)        <font>
(6)          <name val="Calibri"/>
(6)          <family val="2"/>
(6)          <color rgb="FFFA7D00"/>
(6)          <sz val="12"/>
(6)          <scheme val="minor"/>
(4)        </font>
(4)        <protection hidden="0" locked="1"/>
(2)      </namedStyle>
(0)    """
(0)    check_cell = """
(2)      <namedStyle builtinId="23" name="Check Cell" >
(4)        <alignment/>
(4)        <border>
(6)          <left style="double">
(8)            <color rgb="FF3F3F3F"/>
(6)          </left>
(6)          <right style="double">
(8)            <color rgb="FF3F3F3F"/>
(6)          </right>
(6)          <top style="double">
(8)            <color rgb="FF3F3F3F"/>
(6)          </top>
(6)          <bottom style="double">
```

```
(8)                 <color rgb="FF3F3F3F"/>
(6)               </bottom>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
(6)               <patternFill patternType="solid">
(8)                 <fgColor rgb="FFA5A5A5"/>
(6)               </patternFill>
(4)             </fill>
(4)             <font>
(6)               <name val="Calibri"/>
(6)               <family val="2"/>
(6)               <b val="1"/>
(6)               <color theme="0"/>
(6)               <sz val="12"/>
(6)               <scheme val="minor"/>
(4)             </font>
(4)             <protection hidden="0" locked="1"/>
(2)           </namedStyle>
(0)         """
(0)         warning = """
(2)           <namedStyle builtinId="11" name="Warning Text" >
(4)             <alignment/>
(4)             <border>
(6)               <left/>
(6)               <right/>
(6)               <top/>
(6)               <bottom/>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
(6)               <patternFill/>
(4)             </fill>
(4)             <font>
(6)               <name val="Calibri"/>
(6)               <family val="2"/>
(6)               <color rgb="FFFF0000"/>
(6)               <sz val="12"/>
(6)               <scheme val="minor"/>
(4)             </font>
(4)             <protection hidden="0" locked="1"/>
(2)           </namedStyle>
(0)         """
(0)         note = """
(2)           <namedStyle builtinId="10" name="Note" >
(4)             <alignment/>
(4)             <border>
(6)               <left style="thin">
(8)                 <color rgb="FFB2B2B2"/>
(6)               </left>
(6)               <right style="thin">
(8)                 <color rgb="FFB2B2B2"/>
(6)               </right>
(6)               <top style="thin">
(8)                 <color rgb="FFB2B2B2"/>
(6)               </top>
(6)               <bottom style="thin">
(8)                 <color rgb="FFB2B2B2"/>
(6)               </bottom>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
(6)               <patternFill patternType="solid">
(8)                 <fgColor rgb="FFFFFFCC"/>
(6)               </patternFill>
(4)             </fill>
(4)             <font>
(6)               <name val="Calibri"/>
(6)               <family val="2"/>
```

```
(6)                 <color theme="1"/>
(6)                 <sz val="12"/>
(6)                 <scheme val="minor"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>
(0)           """
(0)           explanatory = """
(2)             <namedStyle builtinId="53" name="Explanatory Text" >
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top/>
(6)                 <bottom/>
(6)                 <diagonal/>
(4)               </border>
(4)               <fill>
(6)                 <patternFill/>
(4)               </fill>
(4)               <font>
(6)                 <name val="Calibri"/>
(6)                 <family val="2"/>
(6)                 <i val="1"/>
(6)                 <color rgb="FF7F7F7F"/>
(6)                 <sz val="12"/>
(6)                 <scheme val="minor"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>
(0)           """
(0)           total = """
(2)             <namedStyle builtinId="25" name="Total" >
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top style="thin">
(8)                   <color theme="4"/>
(6)                 </top>
(6)                 <bottom style="double">
(8)                   <color theme="4"/>
(6)                 </bottom>
(6)                 <diagonal/>
(4)               </border>
(4)               <fill>
(6)                 <patternFill/>
(4)               </fill>
(4)               <font>
(6)                 <name val="Calibri"/>
(6)                 <family val="2"/>
(6)                 <b val="1"/>
(6)                 <color theme="1"/>
(6)                 <sz val="12"/>
(6)                 <scheme val="minor"/>
(4)               </font>
(4)               <protection hidden="0" locked="1"/>
(2)             </namedStyle>
(0)           """
(0)           accent_1 = """
(2)             <namedStyle builtinId="29" name="Accent1" >
(4)               <alignment/>
(4)               <border>
(6)                 <left/>
(6)                 <right/>
(6)                 <top/>
(6)                 <bottom/>
(6)                 <diagonal/>
(4)               </border>
```

```
669: (4)                    <fill>
670: (6)                      <patternFill patternType="solid">
671: (8)                        <fgColor theme="4"/>
672: (6)                      </patternFill>
673: (4)                    </fill>
674: (4)                    <font>
675: (6)                      <name val="Calibri"/>
676: (6)                      <family val="2"/>
677: (6)                      <color theme="0"/>
678: (6)                      <sz val="12"/>
679: (6)                      <scheme val="minor"/>
680: (4)                    </font>
681: (4)                    <protection hidden="0" locked="1"/>
682: (2)                  </namedStyle>
683: (0)                """
684: (0)                accent_1_20 = """
685: (2)                  <namedStyle builtinId="30" name="20 % - Accent1" >
686: (4)                    <alignment/>
687: (4)                    <border>
688: (6)                      <left/>
689: (6)                      <right/>
690: (6)                      <top/>
691: (6)                      <bottom/>
692: (6)                      <diagonal/>
693: (4)                    </border>
694: (4)                    <fill>
695: (6)                      <patternFill patternType="solid">
696: (8)                        <fgColor theme="4" tint="0.7999816888943144"/>
697: (8)                        <bgColor indexed="65"/>
698: (6)                      </patternFill>
699: (4)                    </fill>
700: (4)                    <font>
701: (6)                      <name val="Calibri"/>
702: (6)                      <family val="2"/>
703: (6)                      <color theme="1"/>
704: (6)                      <sz val="12"/>
705: (6)                      <scheme val="minor"/>
706: (4)                    </font>
707: (4)                    <protection hidden="0" locked="1"/>
708: (2)                  </namedStyle>
709: (0)                """
710: (0)                accent_1_40 = """
711: (2)                  <namedStyle builtinId="31" name="40 % - Accent1" >
712: (4)                    <alignment/>
713: (4)                    <border>
714: (6)                      <left/>
715: (6)                      <right/>
716: (6)                      <top/>
717: (6)                      <bottom/>
718: (6)                      <diagonal/>
719: (4)                    </border>
720: (4)                    <fill>
721: (6)                      <patternFill patternType="solid">
722: (8)                        <fgColor theme="4" tint="0.5999938962981048"/>
723: (8)                        <bgColor indexed="65"/>
724: (6)                      </patternFill>
725: (4)                    </fill>
726: (4)                    <font>
727: (6)                      <name val="Calibri"/>
728: (6)                      <family val="2"/>
729: (6)                      <color theme="1"/>
730: (6)                      <sz val="12"/>
731: (6)                      <scheme val="minor"/>
732: (4)                    </font>
733: (4)                    <protection hidden="0" locked="1"/>
734: (2)                  </namedStyle>
735: (0)                """
736: (0)                accent_1_60 = """
737: (2)                  <namedStyle builtinId="32" name="60 % - Accent1" >
```

```
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="4" tint="0.3999755851924192"/>
(8)                     <bgColor indexed="65"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <color theme="0"/>
(6)                   <sz val="12"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>
(0)             """
(0)             accent_2 = """<namedStyle builtinId="33" name="Accent2" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="5"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <color theme="0"/>
(6)                   <sz val="12"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>"""
(0)             accent_2_20 = """
(2)               <namedStyle builtinId="34" name="20 % - Accent2" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="5" tint="0.7999816888943144"/>
(8)                     <bgColor indexed="65"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <color theme="1"/>
(6)                   <sz val="12"/>
(6)                   <scheme val="minor"/>
```

```
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>"""
(0)             accent_2_40 = """
(0)             <namedStyle builtinId="35" name="40 % - Accent2" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="5" tint="0.5999938962981048"/>
(8)                     <bgColor indexed="65"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <color theme="1"/>
(6)                   <sz val="12"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>"""
(0)             accent_2_60 = """
(0)             <namedStyle builtinId="36" name="60 % - Accent2" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="5" tint="0.3999755851924192"/>
(8)                     <bgColor indexed="65"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
(6)                   <name val="Calibri"/>
(6)                   <family val="2"/>
(6)                   <color theme="0"/>
(6)                   <sz val="12"/>
(6)                   <scheme val="minor"/>
(4)                 </font>
(4)                 <protection hidden="0" locked="1"/>
(2)               </namedStyle>"""
(0)             accent_3 = """
(0)             <namedStyle builtinId="37" name="Accent3" >
(4)                 <alignment/>
(4)                 <border>
(6)                   <left/>
(6)                   <right/>
(6)                   <top/>
(6)                   <bottom/>
(6)                   <diagonal/>
(4)                 </border>
(4)                 <fill>
(6)                   <patternFill patternType="solid">
(8)                     <fgColor theme="6"/>
(6)                   </patternFill>
(4)                 </fill>
(4)                 <font>
```

```
876: (6)                    <name val="Calibri"/>
877: (6)                    <family val="2"/>
878: (6)                    <color theme="0"/>
879: (6)                    <sz val="12"/>
880: (6)                    <scheme val="minor"/>
881: (4)                  </font>
882: (4)                  <protection hidden="0" locked="1"/>
883: (2)                </namedStyle>"""
884: (0)              accent_3_20 = """
885: (2)                <namedStyle builtinId="38" name="20 % - Accent3" >
886: (4)                  <alignment/>
887: (4)                  <border>
888: (6)                    <left/>
889: (6)                    <right/>
890: (6)                    <top/>
891: (6)                    <bottom/>
892: (6)                    <diagonal/>
893: (4)                  </border>
894: (4)                  <fill>
895: (6)                    <patternFill patternType="solid">
896: (8)                      <fgColor theme="6" tint="0.7999816888943144"/>
897: (8)                      <bgColor indexed="65"/>
898: (6)                    </patternFill>
899: (4)                  </fill>
900: (4)                  <font>
901: (6)                    <name val="Calibri"/>
902: (6)                    <family val="2"/>
903: (6)                    <color theme="1"/>
904: (6)                    <sz val="12"/>
905: (6)                    <scheme val="minor"/>
906: (4)                  </font>
907: (4)                  <protection hidden="0" locked="1"/>
908: (2)                </namedStyle>"""
909: (0)              accent_3_40 = """
910: (2)                <namedStyle builtinId="39" name="40 % - Accent3" >
911: (4)                  <alignment/>
912: (4)                  <border>
913: (6)                    <left/>
914: (6)                    <right/>
915: (6)                    <top/>
916: (6)                    <bottom/>
917: (6)                    <diagonal/>
918: (4)                  </border>
919: (4)                  <fill>
920: (6)                    <patternFill patternType="solid">
921: (8)                      <fgColor theme="6" tint="0.5999938962981048"/>
922: (8)                      <bgColor indexed="65"/>
923: (6)                    </patternFill>
924: (4)                  </fill>
925: (4)                  <font>
926: (6)                    <name val="Calibri"/>
927: (6)                    <family val="2"/>
928: (6)                    <color theme="1"/>
929: (6)                    <sz val="12"/>
930: (6)                    <scheme val="minor"/>
931: (4)                  </font>
932: (4)                  <protection hidden="0" locked="1"/>
933: (2)                </namedStyle>
934: (0)              """
935: (0)              accent_3_60 = """
936: (2)                <namedStyle builtinId="40" name="60 % - Accent3" >
937: (4)                  <alignment/>
938: (4)                  <border>
939: (6)                    <left/>
940: (6)                    <right/>
941: (6)                    <top/>
942: (6)                    <bottom/>
943: (6)                    <diagonal/>
944: (4)                  </border>
```

```
945: (4)                    <fill>
946: (6)                      <patternFill patternType="solid">
947: (8)                        <fgColor theme="6" tint="0.3999755851924192"/>
948: (8)                        <bgColor indexed="65"/>
949: (6)                      </patternFill>
950: (4)                    </fill>
951: (4)                    <font>
952: (6)                      <name val="Calibri"/>
953: (6)                      <family val="2"/>
954: (6)                      <color theme="0"/>
955: (6)                      <sz val="12"/>
956: (6)                      <scheme val="minor"/>
957: (4)                    </font>
958: (4)                    <protection hidden="0" locked="1"/>
959: (2)                  </namedStyle>
960: (0)                """
961: (0)                accent_4 = """
962: (2)                  <namedStyle builtinId="41" name="Accent4" >
963: (4)                    <alignment/>
964: (4)                    <border>
965: (6)                      <left/>
966: (6)                      <right/>
967: (6)                      <top/>
968: (6)                      <bottom/>
969: (6)                      <diagonal/>
970: (4)                    </border>
971: (4)                    <fill>
972: (6)                      <patternFill patternType="solid">
973: (8)                        <fgColor theme="7"/>
974: (6)                      </patternFill>
975: (4)                    </fill>
976: (4)                    <font>
977: (6)                      <name val="Calibri"/>
978: (6)                      <family val="2"/>
979: (6)                      <color theme="0"/>
980: (6)                      <sz val="12"/>
981: (6)                      <scheme val="minor"/>
982: (4)                    </font>
983: (4)                    <protection hidden="0" locked="1"/>
984: (2)                  </namedStyle>
985: (0)                """
986: (0)                accent_4_20 = """
987: (2)                  <namedStyle builtinId="42" name="20 % - Accent4" >
988: (4)                    <alignment/>
989: (4)                    <border>
990: (6)                      <left/>
991: (6)                      <right/>
992: (6)                      <top/>
993: (6)                      <bottom/>
994: (6)                      <diagonal/>
995: (4)                    </border>
996: (4)                    <fill>
997: (6)                      <patternFill patternType="solid">
998: (8)                        <fgColor theme="7" tint="0.7999816888943144"/>
999: (8)                        <bgColor indexed="65"/>
1000: (6)                     </patternFill>
1001: (4)                   </fill>
1002: (4)                   <font>
1003: (6)                     <name val="Calibri"/>
1004: (6)                     <family val="2"/>
1005: (6)                     <color theme="1"/>
1006: (6)                     <sz val="12"/>
1007: (6)                     <scheme val="minor"/>
1008: (4)                   </font>
1009: (4)                   <protection hidden="0" locked="1"/>
1010: (2)                 </namedStyle>
1011: (0)               """
1012: (0)               accent_4_40 = """
1013: (2)                 <namedStyle builtinId="43" name="40 % - Accent4" >
```

```
(4)         <alignment/>
(4)         <border>
(6)           <left/>
(6)           <right/>
(6)           <top/>
(6)           <bottom/>
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor theme="7" tint="0.5999938962981048"/>
(8)             <bgColor indexed="65"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color theme="1"/>
(6)           <sz val="12"/>
(6)           <scheme val="minor"/>
(4)         </font>
(4)         <protection hidden="0" locked="1"/>
(2)       </namedStyle>
(0)     """
(0)     accent_4_60 = """
(0)     <namedStyle builtinId="44" name="60 % - Accent4" >
(4)         <alignment/>
(4)         <border>
(6)           <left/>
(6)           <right/>
(6)           <top/>
(6)           <bottom/>
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor theme="7" tint="0.3999755851924192"/>
(8)             <bgColor indexed="65"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color theme="0"/>
(6)           <sz val="12"/>
(6)           <scheme val="minor"/>
(4)         </font>
(4)         <protection hidden="0" locked="1"/>
(2)       </namedStyle>
(0)     """
(0)     accent_5 = """
(2)       <namedStyle builtinId="45" name="Accent5" >
(4)         <alignment/>
(4)         <border>
(6)           <left/>
(6)           <right/>
(6)           <top/>
(6)           <bottom/>
(6)           <diagonal/>
(4)         </border>
(4)         <fill>
(6)           <patternFill patternType="solid">
(8)             <fgColor theme="8"/>
(6)           </patternFill>
(4)         </fill>
(4)         <font>
(6)           <name val="Calibri"/>
(6)           <family val="2"/>
(6)           <color theme="0"/>
```

```
(6)               <sz val="12"/>
(6)               <scheme val="minor"/>
(4)             </font>
(4)             <protection hidden="0" locked="1"/>
(2)           </namedStyle>
(0)         """
(0)         accent_5_20 = """
(2)           <namedStyle builtinId="46" name="20 % - Accent5" >
(4)             <alignment/>
(4)             <border>
(6)               <left/>
(6)               <right/>
(6)               <top/>
(6)               <bottom/>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
(6)               <patternFill patternType="solid">
(8)                 <fgColor theme="8" tint="0.7999816888943144"/>
(8)                 <bgColor indexed="65"/>
(6)               </patternFill>
(4)             </fill>
(4)             <font>
(6)               <name val="Calibri"/>
(6)               <family val="2"/>
(6)               <color theme="1"/>
(6)               <sz val="12"/>
(6)               <scheme val="minor"/>
(4)             </font>
(4)             <protection hidden="0" locked="1"/>
(2)           </namedStyle>
(0)         """
(0)         accent_5_40 = """
(2)           <namedStyle builtinId="47" name="40 % - Accent5" >
(4)             <alignment/>
(4)             <border>
(6)               <left/>
(6)               <right/>
(6)               <top/>
(6)               <bottom/>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
(6)               <patternFill patternType="solid">
(8)                 <fgColor theme="8" tint="0.5999938962981048"/>
(8)                 <bgColor indexed="65"/>
(6)               </patternFill>
(4)             </fill>
(4)             <font>
(6)               <name val="Calibri"/>
(6)               <family val="2"/>
(6)               <color theme="1"/>
(6)               <sz val="12"/>
(6)               <scheme val="minor"/>
(4)             </font>
(4)             <protection hidden="0" locked="1"/>
(2)           </namedStyle>
(0)         """
(0)         accent_5_60 = """
(2)           <namedStyle builtinId="48" name="60 % - Accent5" >
(4)             <alignment/>
(4)             <border>
(6)               <left/>
(6)               <right/>
(6)               <top/>
(6)               <bottom/>
(6)               <diagonal/>
(4)             </border>
(4)             <fill>
```

```
      <patternFill patternType="solid">
        <fgColor theme="8" tint="0.3999755851924192"/>
        <bgColor indexed="65"/>
      </patternFill>
    </fill>
    <font>
      <name val="Calibri"/>
      <family val="2"/>
      <color theme="0"/>
      <sz val="12"/>
      <scheme val="minor"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
accent_6 = """
  <namedStyle builtinId="49" name="Accent6" >
    <alignment/>
    <border>
      <left/>
      <right/>
      <top/>
      <bottom/>
      <diagonal/>
    </border>
    <fill>
      <patternFill patternType="solid">
        <fgColor theme="9"/>
      </patternFill>
    </fill>
    <font>
      <name val="Calibri"/>
      <family val="2"/>
      <color theme="0"/>
      <sz val="12"/>
      <scheme val="minor"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
accent_6_20 = """
  <namedStyle builtinId="50" name="20 % - Accent6" >
    <alignment/>
    <border>
      <left/>
      <right/>
      <top/>
      <bottom/>
      <diagonal/>
    </border>
    <fill>
      <patternFill patternType="solid">
        <fgColor theme="9" tint="0.7999816888943144"/>
        <bgColor indexed="65"/>
      </patternFill>
    </fill>
    <font>
      <name val="Calibri"/>
      <family val="2"/>
      <color theme="1"/>
      <sz val="12"/>
      <scheme val="minor"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
accent_6_40 = """
  <namedStyle builtinId="51" name="40 % - Accent6" >
    <alignment/>
    <border>
      <left/>
      <right/>
      <top/>
      <bottom/>
      <diagonal/>
    </border>
    <fill>
      <patternFill patternType="solid">
        <fgColor theme="9" tint="0.5999938962981048"/>
        <bgColor indexed="65"/>
      </patternFill>
    </fill>
    <font>
      <name val="Calibri"/>
      <family val="2"/>
      <color theme="1"/>
      <sz val="12"/>
      <scheme val="minor"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
accent_6_60 = """
  <namedStyle builtinId="52" name="60 % - Accent6" >
    <alignment/>
    <border>
      <left/>
      <right/>
      <top/>
      <bottom/>
      <diagonal/>
    </border>
    <fill>
      <patternFill patternType="solid">
        <fgColor theme="9" tint="0.3999755851924192"/>
        <bgColor indexed="65"/>
      </patternFill>
    </fill>
    <font>
      <name val="Calibri"/>
      <family val="2"/>
      <color theme="0"/>
      <sz val="12"/>
      <scheme val="minor"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
pandas_highlight = """
  <namedStyle hidden="0" name="Pandas">
    <alignment horizontal="center"/>
    <border>
      <left style="thin"><color rgb="00000000"/></left>
      <right style="thin"><color rgb="00000000"/></right>
      <top style="thin"><color rgb="00000000"/></top>
      <bottom style="thin"><color rgb="00000000"/></bottom>
      <diagonal/>
    </border>
    <fill>
      <patternFill/>
    </fill>
    <font>
      <b val="1"/>
    </font>
    <protection hidden="0" locked="1"/>
  </namedStyle>
"""
styles = dict(
1290: (4)               [
1291: (8)                   ('Normal', NamedStyle.from_tree(fromstring(normal))),
1292: (8)                   ('Comma', NamedStyle.from_tree(fromstring(comma))),
1293: (8)                   ('Currency', NamedStyle.from_tree(fromstring(currency))),
1294: (8)                   ('Percent', NamedStyle.from_tree(fromstring(percent))),
1295: (8)                   ('Comma [0]', NamedStyle.from_tree(fromstring(comma_0))),
1296: (8)                   ('Currency [0]', NamedStyle.from_tree(fromstring(currency_0))),
1297: (8)                   ('Hyperlink', NamedStyle.from_tree(fromstring(hyperlink))),
1298: (8)                   ('Followed Hyperlink',
NamedStyle.from_tree(fromstring(followed_hyperlink))),
1299: (8)                   ('Note', NamedStyle.from_tree(fromstring(note))),
1300: (8)                   ('Warning Text', NamedStyle.from_tree(fromstring(warning))),
1301: (8)                   ('Title', NamedStyle.from_tree(fromstring(title))),
1302: (8)                   ('Headline 1', NamedStyle.from_tree(fromstring(headline_1))),
1303: (8)                   ('Headline 2', NamedStyle.from_tree(fromstring(headline_2))),
1304: (8)                   ('Headline 3', NamedStyle.from_tree(fromstring(headline_3))),
1305: (8)                   ('Headline 4', NamedStyle.from_tree(fromstring(headline_4))),
1306: (8)                   ('Input', NamedStyle.from_tree(fromstring(input))),
1307: (8)                   ('Output', NamedStyle.from_tree(fromstring(output))),
1308: (8)                   ('Calculation',NamedStyle.from_tree(fromstring(calculation))),
1309: (8)                   ('Check Cell', NamedStyle.from_tree(fromstring(check_cell))),
1310: (8)                   ('Linked Cell', NamedStyle.from_tree(fromstring(linked_cell))),
1311: (8)                   ('Total', NamedStyle.from_tree(fromstring(total))),
1312: (8)                   ('Good', NamedStyle.from_tree(fromstring(good))),
1313: (8)                   ('Bad', NamedStyle.from_tree(fromstring(bad))),
1314: (8)                   ('Neutral', NamedStyle.from_tree(fromstring(neutral))),
1315: (8)                   ('Accent1', NamedStyle.from_tree(fromstring(accent_1))),
1316: (8)                   ('20 % - Accent1', NamedStyle.from_tree(fromstring(accent_1_20))),
1317: (8)                   ('40 % - Accent1', NamedStyle.from_tree(fromstring(accent_1_40))),
1318: (8)                   ('60 % - Accent1', NamedStyle.from_tree(fromstring(accent_1_60))),
1319: (8)                   ('Accent2', NamedStyle.from_tree(fromstring(accent_2))),
1320: (8)                   ('20 % - Accent2', NamedStyle.from_tree(fromstring(accent_2_20))),
1321: (8)                   ('40 % - Accent2', NamedStyle.from_tree(fromstring(accent_2_40))),
1322: (8)                   ('60 % - Accent2', NamedStyle.from_tree(fromstring(accent_2_60))),
1323: (8)                   ('Accent3', NamedStyle.from_tree(fromstring(accent_3))),
1324: (8)                   ('20 % - Accent3', NamedStyle.from_tree(fromstring(accent_3_20))),
1325: (8)                   ('40 % - Accent3', NamedStyle.from_tree(fromstring(accent_3_40))),
1326: (8)                   ('60 % - Accent3', NamedStyle.from_tree(fromstring(accent_3_60))),
1327: (8)                   ('Accent4', NamedStyle.from_tree(fromstring(accent_4))),
1328: (8)                   ('20 % - Accent4', NamedStyle.from_tree(fromstring(accent_4_20))),
1329: (8)                   ('40 % - Accent4', NamedStyle.from_tree(fromstring(accent_4_40))),
1330: (8)                   ('60 % - Accent4', NamedStyle.from_tree(fromstring(accent_4_60))),
1331: (8)                   ('Accent5', NamedStyle.from_tree(fromstring(accent_5))),
1332: (8)                   ('20 % - Accent5', NamedStyle.from_tree(fromstring(accent_5_20))),
1333: (8)                   ('40 % - Accent5', NamedStyle.from_tree(fromstring(accent_5_40))),
1334: (8)                   ('60 % - Accent5', NamedStyle.from_tree(fromstring(accent_5_60))),
1335: (8)                   ('Accent6', NamedStyle.from_tree(fromstring(accent_6))),
1336: (8)                   ('20 % - Accent6', NamedStyle.from_tree(fromstring(accent_6_20))),
1337: (8)                   ('40 % - Accent6', NamedStyle.from_tree(fromstring(accent_6_40))),
1338: (8)                   ('60 % - Accent6', NamedStyle.from_tree(fromstring(accent_6_60))),
1339: (8)                   ('Explanatory Text', NamedStyle.from_tree(fromstring(explanatory))),
1340: (8)                   ('Pandas', NamedStyle.from_tree(fromstring(pandas_highlight)))
1341: (4)               ]
1342: (0)           )
```

File 118 - __init__.py:

```
1: (0)              from .alignment import Alignment
2: (0)              from .borders import Border, Side
3: (0)              from .colors import Color
4: (0)              from .fills import PatternFill, GradientFill, Fill
5: (0)              from .fonts import Font, DEFAULT_FONT
6: (0)              from .numbers import NumberFormatDescriptor, is_date_format, is_builtin
7: (0)              from .protection import Protection
8: (0)              from .named_styles import NamedStyle
```

File 119 - alignment.py:

```
1: (0)              from openpyxl.compat import safe_string
2: (0)              from openpyxl.descriptors import Bool, MinMax, Min, Alias, NoneSet
3: (0)              from openpyxl.descriptors.serialisable import Serialisable
4: (0)              horizontal_alignments = (
5: (4)                  "general", "left", "center", "right", "fill", "justify",
"centerContinuous",
6: (4)                  "distributed", )
7: (0)              vertical_aligments = (
8: (4)                  "top", "center", "bottom", "justify", "distributed",
9: (0)              )
10: (0)             class Alignment(Serialisable):
11: (4)                 """Alignment options for use in styles."""
12: (4)                 tagname = "alignment"
13: (4)                 horizontal = NoneSet(values=horizontal_alignments)
14: (4)                 vertical = NoneSet(values=vertical_aligments)
15: (4)                 textRotation = NoneSet(values=range(181))
16: (4)                 textRotation.values.add(255)
17: (4)                 text_rotation = Alias('textRotation')
18: (4)                 wrapText = Bool(allow_none=True)
19: (4)                 wrap_text = Alias('wrapText')
20: (4)                 shrinkToFit = Bool(allow_none=True)
21: (4)                 shrink_to_fit = Alias('shrinkToFit')
22: (4)                 indent = MinMax(min=0, max=255)
23: (4)                 relativeIndent = MinMax(min=-255, max=255)
24: (4)                 justifyLastLine = Bool(allow_none=True)
25: (4)                 readingOrder = Min(min=0)
26: (4)                 def __init__(self, horizontal=None, vertical=None,
27: (17)                             textRotation=0, wrapText=None, shrinkToFit=None, indent=0,
relativeIndent=0,
28: (17)                             justifyLastLine=None, readingOrder=0, text_rotation=None,
29: (17)                             wrap_text=None, shrink_to_fit=None, mergeCell=None):
30: (8)                     self.horizontal = horizontal
31: (8)                     self.vertical = vertical
32: (8)                     self.indent = indent
33: (8)                     self.relativeIndent = relativeIndent
34: (8)                     self.justifyLastLine = justifyLastLine
35: (8)                     self.readingOrder = readingOrder
36: (8)                     if text_rotation is not None:
37: (12)                        textRotation = text_rotation
38: (8)                     if textRotation is not None:
39: (12)                        self.textRotation = int(textRotation)
40: (8)                     if wrap_text is not None:
41: (12)                        wrapText = wrap_text
42: (8)                     self.wrapText = wrapText
43: (8)                     if shrink_to_fit is not None:
44: (12)                        shrinkToFit = shrink_to_fit
45: (8)                     self.shrinkToFit = shrinkToFit
46: (4)                 def __iter__(self):
47: (8)                     for attr in self.__attrs__:
48: (12)                        value = getattr(self, attr)
49: (12)                        if value is not None and value != 0:
50: (16)                            yield attr, safe_string(value)
```

File 120 - styleable.py:

```
1: (0)              from copy import copy
2: (0)              from .numbers import (
3: (4)                  BUILTIN_FORMATS,
4: (4)                  BUILTIN_FORMATS_MAX_SIZE,
5: (4)                  BUILTIN_FORMATS_REVERSE,
6: (0)              )
7: (0)              from .proxy import StyleProxy
8: (0)              from .cell_style import StyleArray
9: (0)              from .named_styles import NamedStyle
```

```
from .builtins import styles
class StyleDescriptor:
    def __init__(self, collection, key):
        self.collection = collection
        self.key = key
    def __set__(self, instance, value):
        coll = getattr(instance.parent.parent, self.collection)
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        setattr(instance._style, self.key, coll.add(value))
    def __get__(self, instance, cls):
        coll = getattr(instance.parent.parent, self.collection)
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        idx =  getattr(instance._style, self.key)
        return StyleProxy(coll[idx])
class NumberFormatDescriptor:
    key = "numFmtId"
    collection = '_number_formats'
    def __set__(self, instance, value):
        coll = getattr(instance.parent.parent, self.collection)
        if value in BUILTIN_FORMATS_REVERSE:
            idx = BUILTIN_FORMATS_REVERSE[value]
        else:
            idx = coll.add(value) + BUILTIN_FORMATS_MAX_SIZE
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        setattr(instance._style, self.key, idx)
    def __get__(self, instance, cls):
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        idx = getattr(instance._style, self.key)
        if idx < BUILTIN_FORMATS_MAX_SIZE:
            return BUILTIN_FORMATS.get(idx, "General")
        coll = getattr(instance.parent.parent, self.collection)
        return coll[idx - BUILTIN_FORMATS_MAX_SIZE]
class NamedStyleDescriptor:
    key = "xfId"
    collection = "_named_styles"
    def __set__(self, instance, value):
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        coll = getattr(instance.parent.parent, self.collection)
        if isinstance(value, NamedStyle):
            style = value
            if style not in coll:
                instance.parent.parent.add_named_style(style)
        elif value not in coll.names:
            if value in styles: # is it builtin?
                style = styles[value]
                if style not in coll:
                    instance.parent.parent.add_named_style(style)
            else:
                raise ValueError("{0} is not a known style".format(value))
        else:
            style = coll[value]
        instance._style = copy(style.as_tuple())
    def __get__(self, instance, cls):
        if not getattr(instance, "_style"):
            instance._style = StyleArray()
        idx = getattr(instance._style, self.key)
        coll = getattr(instance.parent.parent, self.collection)
        return coll.names[idx]
class StyleArrayDescriptor:
    def __init__(self, key):
        self.key = key
    def __set__(self, instance, value):
        if instance._style is None:
            instance._style = StyleArray()
        setattr(instance._style, self.key, value)
    def __get__(self, instance, cls):
        if instance._style is None:
            return False
        return bool(getattr(instance._style, self.key))
class StyleableObject:
    """
    Base class for styleble objects implementing proxy and lookup functions
    """
    font = StyleDescriptor('_fonts', "fontId")
    fill = StyleDescriptor('_fills', "fillId")
    border = StyleDescriptor('_borders', "borderId")
    number_format = NumberFormatDescriptor()
    protection = StyleDescriptor('_protections', "protectionId")
    alignment = StyleDescriptor('_alignments', "alignmentId")
    style = NamedStyleDescriptor()
    quotePrefix = StyleArrayDescriptor('quotePrefix')
    pivotButton = StyleArrayDescriptor('pivotButton')
    __slots__ = ('parent', '_style')
    def __init__(self, sheet, style_array=None):
        self.parent = sheet
        if style_array is not None:
            style_array = StyleArray(style_array)
        self._style = style_array
    @property
    def style_id(self):
        if self._style is None:
            self._style = StyleArray()
        return self.parent.parent._cell_styles.add(self._style)
    @property
    def has_style(self):
        if self._style is None:
            return False
        return any(self._style)
```

File 121 - cell_style.py:

```
from array import array
from openpyxl.descriptors.serialisable import Serialisable
from openpyxl.descriptors import (
    Typed,
    Float,
    Bool,
    Integer,
    Sequence,
)
from openpyxl.descriptors.excel import ExtensionList
from openpyxl.utils.indexed_list import IndexedList
from .alignment import Alignment
from .protection import Protection
class ArrayDescriptor:
    def __init__(self, key):
        self.key = key
    def __get__(self, instance, cls):
        return instance[self.key]
    def __set__(self, instance, value):
        instance[self.key] = value
class StyleArray(array):
    """
    Simplified named tuple with an array
    """
    __slots__ = ()
    tagname = 'xf'
    fontId = ArrayDescriptor(0)
    fillId = ArrayDescriptor(1)
    borderId = ArrayDescriptor(2)
    numFmtId = ArrayDescriptor(3)
```

```
31: (4)                 protectionId = ArrayDescriptor(4)
32: (4)                 alignmentId = ArrayDescriptor(5)
33: (4)                 pivotButton = ArrayDescriptor(6)
34: (4)                 quotePrefix = ArrayDescriptor(7)
35: (4)                 xfId = ArrayDescriptor(8)
36: (4)                 def __new__(cls, args=[0]*9):
37: (8)                     return array.__new__(cls, 'i', args)
38: (4)                 def __hash__(self):
39: (8)                     return hash(tuple(self))
40: (4)                 def __copy__(self):
41: (8)                     return StyleArray((self))
42: (4)                 def __deepcopy__(self, memo):
43: (8)                     return StyleArray((self))
44: (0)             class CellStyle(Serialisable):
45: (4)                 tagname = "xf"
46: (4)                 numFmtId = Integer()
47: (4)                 fontId = Integer()
48: (4)                 fillId = Integer()
49: (4)                 borderId = Integer()
50: (4)                 xfId = Integer(allow_none=True)
51: (4)                 quotePrefix = Bool(allow_none=True)
52: (4)                 pivotButton = Bool(allow_none=True)
53: (4)                 applyNumberFormat = Bool(allow_none=True)
54: (4)                 applyFont = Bool(allow_none=True)
55: (4)                 applyFill = Bool(allow_none=True)
56: (4)                 applyBorder = Bool(allow_none=True)
57: (4)                 applyAlignment = Bool(allow_none=True)
58: (4)                 applyProtection = Bool(allow_none=True)
59: (4)                 alignment = Typed(expected_type=Alignment, allow_none=True)
60: (4)                 protection = Typed(expected_type=Protection, allow_none=True)
61: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
62: (4)                 __elements__ = ('alignment', 'protection')
63: (4)                 __attrs__ = ("numFmtId", "fontId", "fillId", "borderId",
64: (17)                             "applyAlignment", "applyProtection", "pivotButton",
"quotePrefix", "xfId")
65: (4)                 def __init__(self,
66: (17)                             numFmtId=0,
67: (17)                             fontId=0,
68: (17)                             fillId=0,
69: (17)                             borderId=0,
70: (17)                             xfId=None,
71: (17)                             quotePrefix=None,
72: (17)                             pivotButton=None,
73: (17)                             applyNumberFormat=None,
74: (17)                             applyFont=None,
75: (17)                             applyFill=None,
76: (17)                             applyBorder=None,
77: (17)                             applyAlignment=None,
78: (17)                             applyProtection=None,
79: (17)                             alignment=None,
80: (17)                             protection=None,
81: (17)                             extLst=None,
82: (16)                            ):
83: (8)                     self.numFmtId = numFmtId
84: (8)                     self.fontId = fontId
85: (8)                     self.fillId = fillId
86: (8)                     self.borderId = borderId
87: (8)                     self.xfId = xfId
88: (8)                     self.quotePrefix = quotePrefix
89: (8)                     self.pivotButton = pivotButton
90: (8)                     self.applyNumberFormat = applyNumberFormat
91: (8)                     self.applyFont = applyFont
92: (8)                     self.applyFill = applyFill
93: (8)                     self.applyBorder = applyBorder
94: (8)                     self.alignment = alignment
95: (8)                     self.protection = protection
96: (4)                 def to_array(self):
97: (8)                     """
98: (8)                     Convert to StyleArray
```

```
99: (8)                     """
100: (8)                    style = StyleArray()
101: (8)                    for k in ("fontId", "fillId", "borderId", "numFmtId", "pivotButton",
102: (18)                             "quotePrefix", "xfId"):
103: (12)                       v = getattr(self, k, 0)
104: (12)                       if v is not None:
105: (16)                           setattr(style, k, v)
106: (8)                    return style
107: (4)                @classmethod
108: (4)                def from_array(cls, style):
109: (8)                    """
110: (8)                    Convert from StyleArray
111: (8)                    """
112: (8)                    return cls(numFmtId=style.numFmtId, fontId=style.fontId,
113: (19)                              fillId=style.fillId, borderId=style.borderId,
xfId=style.xfId,
114: (19)                              quotePrefix=style.quotePrefix,
pivotButton=style.pivotButton,)
115: (4)                @property
116: (4)                def applyProtection(self):
117: (8)                    return self.protection is not None or None
118: (4)                @property
119: (4)                def applyAlignment(self):
120: (8)                    return self.alignment is not None or None
121: (0)            class CellStyleList(Serialisable):
122: (4)                tagname = "cellXfs"
123: (4)                __attrs__ = ("count",)
124: (4)                count = Integer(allow_none=True)
125: (4)                xf = Sequence(expected_type=CellStyle)
126: (4)                alignment = Sequence(expected_type=Alignment)
127: (4)                protection = Sequence(expected_type=Protection)
128: (4)                __elements__ = ('xf',)
129: (4)                def __init__(self,
130: (17)                            count=None,
131: (17)                            xf=(),
132: (16)                           ):
133: (8)                    self.xf = xf
134: (4)                @property
135: (4)                def count(self):
136: (8)                    return len(self.xf)
137: (4)                def __getitem__(self, idx):
138: (8)                    try:
139: (12)                       return self.xf[idx]
140: (8)                    except IndexError:
141: (12)                       print((f"{idx} is out of range"))
142: (8)                    return self.xf[idx]
143: (4)                def _to_array(self):
144: (8)                    """
145: (8)                    Extract protection and alignments, convert to style array
146: (8)                    """
147: (8)                    self.prots = IndexedList([Protection()])
148: (8)                    self.alignments = IndexedList([Alignment()])
149: (8)                    styles = [] # allow duplicates
150: (8)                    for xf in self.xf:
151: (12)                       style = xf.to_array()
152: (12)                       if xf.alignment is not None:
153: (16)                           style.alignmentId = self.alignments.add(xf.alignment)
154: (12)                       if xf.protection is not None:
155: (16)                           style.protectionId = self.prots.add(xf.protection)
156: (12)                       styles.append(style)
157: (8)                    return IndexedList(styles)

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

File 122 - protection.py:

1: (0)              from openpyxl.descriptors import Bool
2: (0)              from openpyxl.descriptors.serialisable import Serialisable
3: (0)              class Protection(Serialisable):
```

```
4: (4)                  """Protection options for use in styles."""
5: (4)                  tagname = "protection"
6: (4)                  locked = Bool()
7: (4)                  hidden = Bool()
8: (4)                  def __init__(self, locked=True, hidden=False):
9: (8)                      self.locked = locked
10: (8)                     self.hidden = hidden

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

File 123 - stylesheet.py:

1: (0)              from warnings import warn
2: (0)              from openpyxl.descriptors.serialisable import Serialisable
3: (0)              from openpyxl.descriptors import (
4: (4)                  Typed,
5: (0)              )
6: (0)              from openpyxl.descriptors.sequence import NestedSequence
7: (0)              from openpyxl.descriptors.excel import ExtensionList
8: (0)              from openpyxl.utils.indexed_list import IndexedList
9: (0)              from openpyxl.xml.constants import ARC_STYLE, SHEET_MAIN_NS
10: (0)             from openpyxl.xml.functions import fromstring
11: (0)             from .builtins import styles
12: (0)             from .colors import ColorList
13: (0)             from .differential import DifferentialStyle
14: (0)             from .table import TableStyleList
15: (0)             from .borders import Border
16: (0)             from .fills import Fill
17: (0)             from .fonts import Font
18: (0)             from .numbers import (
19: (4)                 NumberFormatList,
20: (4)                 BUILTIN_FORMATS,
21: (4)                 BUILTIN_FORMATS_MAX_SIZE,
22: (4)                 BUILTIN_FORMATS_REVERSE,
23: (4)                 is_date_format,
24: (4)                 is_timedelta_format,
25: (4)                 builtin_format_code
26: (0)             )
27: (0)             from .named_styles import (
28: (4)                 _NamedCellStyleList,
29: (4)                 NamedStyleList,
30: (4)                 NamedStyle,
31: (0)             )
32: (0)             from .cell_style import CellStyle, CellStyleList
33: (0)             class Stylesheet(Serialisable):
34: (4)                 tagname = "styleSheet"
35: (4)                 numFmts = Typed(expected_type=NumberFormatList)
36: (4)                 fonts = NestedSequence(expected_type=Font, count=True)
37: (4)                 fills = NestedSequence(expected_type=Fill, count=True)
38: (4)                 borders = NestedSequence(expected_type=Border, count=True)
39: (4)                 cellStyleXfs = Typed(expected_type=CellStyleList)
40: (4)                 cellXfs = Typed(expected_type=CellStyleList)
41: (4)                 cellStyles = Typed(expected_type=_NamedCellStyleList)
42: (4)                 dxfs = NestedSequence(expected_type=DifferentialStyle, count=True)
43: (4)                 tableStyles = Typed(expected_type=TableStyleList, allow_none=True)
44: (4)                 colors = Typed(expected_type=ColorList, allow_none=True)
45: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
46: (4)                 __elements__ = ('numFmts', 'fonts', 'fills', 'borders', 'cellStyleXfs',
47: (20)                                'cellXfs', 'cellStyles', 'dxfs', 'tableStyles', 'colors')
48: (4)                 def __init__(self,
49: (17)                             numFmts=None,
50: (17)                             fonts=(),
51: (17)                             fills=(),
52: (17)                             borders=(),
53: (17)                             cellStyleXfs=None,
54: (17)                             cellXfs=None,
55: (17)                             cellStyles=None,
56: (17)                             dxfs=(),
57: (17)                             tableStyles=None,
58: (17)                         colors=None,
59: (17)                         extLst=None,
60: (16)                        ):
61: (8)                 if numFmts is None:
62: (12)                    numFmts = NumberFormatList()
63: (8)                 self.numFmts = numFmts
64: (8)                 self.number_formats = IndexedList()
65: (8)                 self.fonts = fonts
66: (8)                 self.fills = fills
67: (8)                 self.borders = borders
68: (8)                 if cellStyleXfs is None:
69: (12)                    cellStyleXfs = CellStyleList()
70: (8)                 self.cellStyleXfs = cellStyleXfs
71: (8)                 if cellXfs is None:
72: (12)                    cellXfs = CellStyleList()
73: (8)                 self.cellXfs = cellXfs
74: (8)                 if cellStyles is None:
75: (12)                    cellStyles = _NamedCellStyleList()
76: (8)                 self.cellStyles = cellStyles
77: (8)                 self.dxfs = dxfs
78: (8)                 self.tableStyles = tableStyles
79: (8)                 self.colors = colors
80: (8)                 self.cell_styles = self.cellXfs._to_array()
81: (8)                 self.alignments = self.cellXfs.alignments
82: (8)                 self.protections = self.cellXfs.prots
83: (8)                 self._normalise_numbers()
84: (8)                 self.named_styles = self._merge_named_styles()
85: (4)             @classmethod
86: (4)             def from_tree(cls, node):
87: (8)                 attrs = dict(node.attrib)
88: (8)                 for k in attrs:
89: (12)                    del node.attrib[k]
90: (8)                 return super().from_tree(node)
91: (4)             def _merge_named_styles(self):
92: (8)                 """
93: (8)                 Merge named style names "cellStyles" with their associated styles
94: (8)                 "cellStyleXfs"
95: (8)                 """
96: (8)                 style_refs = self.cellStyles.remove_duplicates()
97: (8)                 from_ref = [self._expand_named_style(style_ref) for style_ref in
style_refs]
98: (8)                 return NamedStyleList(from_ref)
99: (4)             def _expand_named_style(self, style_ref):
100: (8)                """
101: (8)                Expand a named style reference element to a
102: (8)                named style object by binding the relevant
103: (8)                objects from the stylesheet
104: (8)                """
105: (8)                xf = self.cellStyleXfs[style_ref.xfId]
106: (8)                named_style = NamedStyle(
107: (12)                   name=style_ref.name,
108: (12)                   hidden=style_ref.hidden,
109: (12)                   builtinId=style_ref.builtinId,
110: (8)                )
111: (8)                named_style.font = self.fonts[xf.fontId]
112: (8)                named_style.fill = self.fills[xf.fillId]
113: (8)                named_style.border = self.borders[xf.borderId]
114: (8)                if xf.numFmtId < BUILTIN_FORMATS_MAX_SIZE:
115: (12)                   formats = BUILTIN_FORMATS
116: (8)                else:
117: (12)                   formats = self.custom_formats
118: (8)                if xf.numFmtId in formats:
119: (12)                   named_style.number_format = formats[xf.numFmtId]
120: (8)                if xf.alignment:
121: (12)                   named_style.alignment = xf.alignment
122: (8)                if xf.protection:
123: (12)                   named_style.protection = xf.protection
124: (8)                return named_style
125: (4)            def _split_named_styles(self, wb):
```

```
(8)             """
(8)             Convert NamedStyle into separate CellStyle and Xf objects
(8)             """
(8)             for  style in wb._named_styles:
(12)                self.cellStyles.cellStyle.append(style.as_name())
(12)                self.cellStyleXfs.xf.append(style.as_xf())
(4)         @property
(4)         def custom_formats(self):
(8)             return dict([(n.numFmtId, n.formatCode) for n in self.numFmts.numFmt])
(4)         def _normalise_numbers(self):
(8)             """
(8)             Rebase custom numFmtIds with a floor of 164 when reading stylesheet
(8)             And index datetime formats
(8)             """
(8)             date_formats = set()
(8)             timedelta_formats = set()
(8)             custom = self.custom_formats
(8)             formats = self.number_formats
(8)             for idx, style in enumerate(self.cell_styles):
(12)                if style.numFmtId in custom:
(16)                    fmt = custom[style.numFmtId]
(16)                    if fmt in BUILTIN_FORMATS_REVERSE: # remove builtins
(20)                        style.numFmtId = BUILTIN_FORMATS_REVERSE[fmt]
(16)                    else:
(20)                        style.numFmtId = formats.add(fmt) +
BUILTIN_FORMATS_MAX_SIZE
(12)                else:
(16)                    fmt = builtin_format_code(style.numFmtId)
(12)                if is_date_format(fmt):
(16)                    date_formats.add(idx)
(12)                if is_timedelta_format(fmt):
(16)                    timedelta_formats.add(idx)
(8)             self.date_formats = date_formats
(8)             self.timedelta_formats = timedelta_formats
(4)         def to_tree(self, tagname=None, idx=None, namespace=None):
(8)             tree = super().to_tree(tagname, idx, namespace)
(8)             tree.set("xmlns", SHEET_MAIN_NS)
(8)             return tree
(0)     def apply_stylesheet(archive, wb):
(4)         """
(4)         Add styles to workbook if present
(4)         """
(4)         try:
(8)             src = archive.read(ARC_STYLE)
(4)         except KeyError:
(8)             return wb
(4)         node = fromstring(src)
(4)         stylesheet = Stylesheet.from_tree(node)
(4)         if stylesheet.cell_styles:
(8)             wb._borders = IndexedList(stylesheet.borders)
(8)             wb._fonts = IndexedList(stylesheet.fonts)
(8)             wb._fills = IndexedList(stylesheet.fills)
(8)             wb._differential_styles.styles = stylesheet.dxfs
(8)             wb._number_formats = stylesheet.number_formats
(8)             wb._protections = stylesheet.protections
(8)             wb._alignments = stylesheet.alignments
(8)             wb._table_styles = stylesheet.tableStyles
(8)             wb._cell_styles = stylesheet.cell_styles
(8)             wb._named_styles = stylesheet.named_styles
(8)             wb._date_formats = stylesheet.date_formats
(8)             wb._timedelta_formats = stylesheet.timedelta_formats
(8)             for ns in wb._named_styles:
(12)                ns.bind(wb)
(4)         else:
(8)             warn("Workbook contains no stylesheet, using openpyxl's defaults")
(4)         if not wb._named_styles:
(8)             normal = styles['Normal']
(8)             wb.add_named_style(normal)
(8)             warn("Workbook contains no default style, apply openpyxl's default")
```

```
(4)             if stylesheet.colors is not None:
(8)                 wb._colors = stylesheet.colors.index
(0)         def write_stylesheet(wb):
(4)             stylesheet = Stylesheet()
(4)             stylesheet.fonts = wb._fonts
(4)             stylesheet.fills = wb._fills
(4)             stylesheet.borders = wb._borders
(4)             stylesheet.dxfs = wb._differential_styles.styles
(4)             stylesheet.colors = ColorList(indexedColors=wb._colors)
(4)             from .numbers import NumberFormat
(4)             fmts = []
(4)             for idx, code in enumerate(wb._number_formats, BUILTIN_FORMATS_MAX_SIZE):
(8)                 fmt = NumberFormat(idx, code)
(8)                 fmts.append(fmt)
(4)             stylesheet.numFmts.numFmt = fmts
(4)             xfs = []
(4)             for style in wb._cell_styles:
(8)                 xf = CellStyle.from_array(style)
(8)                 if style.alignmentId:
(12)                    xf.alignment = wb._alignments[style.alignmentId]
(8)                 if style.protectionId:
(12)                    xf.protection = wb._protections[style.protectionId]
(8)                 xfs.append(xf)
(4)             stylesheet.cellXfs = CellStyleList(xf=xfs)
(4)             stylesheet._split_named_styles(wb)
(4)             stylesheet.tableStyles = wb._table_styles
(4)             return stylesheet.to_tree()
```

File 124 - differential.py:

```
(0)         from openpyxl.descriptors import (
(4)             Typed,
(4)             Sequence,
(4)             Alias,
(0)         )
(0)         from openpyxl.descriptors.serialisable import Serialisable
(0)         from openpyxl.styles import (
(4)             Font,
(4)             Fill,
(4)             Border,
(4)             Alignment,
(4)             Protection,
(4)             )
(0)         from .numbers import NumberFormat
(0)         class DifferentialStyle(Serialisable):
(4)             tagname = "dxf"
(4)             __elements__ = ("font", "numFmt", "fill", "alignment", "border",
"protection")
(4)             font = Typed(expected_type=Font, allow_none=True)
(4)             numFmt = Typed(expected_type=NumberFormat, allow_none=True)
(4)             fill = Typed(expected_type=Fill, allow_none=True)
(4)             alignment = Typed(expected_type=Alignment, allow_none=True)
(4)             border = Typed(expected_type=Border, allow_none=True)
(4)             protection = Typed(expected_type=Protection, allow_none=True)
(4)             def __init__(self,
(17)                         font=None,
(17)                         numFmt=None,
(17)                         fill=None,
(17)                         alignment=None,
(17)                         border=None,
(17)                         protection=None,
(17)                         extLst=None,
(16)                        ):
(8)                 self.font = font
(8)                 self.numFmt = numFmt
(8)                 self.fill = fill
(8)                 self.alignment = alignment
```

```
(8)                         self.border = border
(8)                         self.protection = protection
(8)                         self.extLst = extLst
(0)                 class DifferentialStyleList(Serialisable):
(4)                     """
(4)                     Dedupable container for differential styles.
(4)                     """
(4)                     tagname = "dxfs"
(4)                     dxf = Sequence(expected_type=DifferentialStyle)
(4)                     styles = Alias("dxf")
(4)                     __attrs__ = ("count",)
(4)                     def __init__(self, dxf=(), count=None):
(8)                         self.dxf = dxf
(4)                     def append(self, dxf):
(8)                         """
(8)                         Check to see whether style already exists and append it if does not.
(8)                         """
(8)                         if not isinstance(dxf, DifferentialStyle):
(12)                            raise TypeError('expected ' + str(DifferentialStyle))
(8)                         if dxf in self.styles:
(12)                            return
(8)                         self.styles.append(dxf)
(4)                     def add(self, dxf):
(8)                         """
(8)                         Add a differential style and return its index
(8)                         """
(8)                         self.append(dxf)
(8)                         return self.styles.index(dxf)
(4)                     def __bool__(self):
(8)                         return bool(self.styles)
(4)                     def __getitem__(self, idx):
(8)                         return self.styles[idx]
(4)                     @property
(4)                     def count(self):
(8)                         return len(self.dxf)
```

File 125 - named_styles.py:

```
(0)                 from openpyxl.compat import safe_string
(0)                 from openpyxl.descriptors import (
(4)                     Typed,
(4)                     Integer,
(4)                     Bool,
(4)                     String,
(4)                     Sequence,
(0)                 )
(0)                 from openpyxl.descriptors.excel import ExtensionList
(0)                 from openpyxl.descriptors.serialisable import Serialisable
(0)                 from .fills import PatternFill, Fill
(0)                 from .fonts import Font
(0)                 from .borders import Border
(0)                 from .alignment import Alignment
(0)                 from .protection import Protection
(0)                 from .numbers import (
(4)                     NumberFormatDescriptor,
(4)                     BUILTIN_FORMATS_MAX_SIZE,
(4)                     BUILTIN_FORMATS_REVERSE,
(0)                 )
(0)                 from .cell_style import (
(4)                     StyleArray,
(4)                     CellStyle,
(0)                 )
(0)                 class NamedStyle(Serialisable):
(4)                     """
(4)                     Named and editable styles
(4)                     """
(4)                     font = Typed(expected_type=Font)
```

```
  30: (4)                 fill = Typed(expected_type=Fill)
  31: (4)                 border = Typed(expected_type=Border)
  32: (4)                 alignment = Typed(expected_type=Alignment)
  33: (4)                 number_format = NumberFormatDescriptor()
  34: (4)                 protection = Typed(expected_type=Protection)
  35: (4)                 builtinId = Integer(allow_none=True)
  36: (4)                 hidden = Bool(allow_none=True)
  37: (4)                 name = String()
  38: (4)                 _wb = None
  39: (4)                 _style = StyleArray()
  40: (4)                 def __init__(self,
  41: (17)                             name="Normal",
  42: (17)                             font=None,
  43: (17)                             fill=None,
  44: (17)                             border=None,
  45: (17)                             alignment=None,
  46: (17)                             number_format=None,
  47: (17)                             protection=None,
  48: (17)                             builtinId=None,
  49: (17)                             hidden=False,
  50: (17)                             ):
  51: (8)                     self.name = name
  52: (8)                     self.font = font or Font()
  53: (8)                     self.fill = fill or PatternFill()
  54: (8)                     self.border = border or Border()
  55: (8)                     self.alignment = alignment or Alignment()
  56: (8)                     self.number_format = number_format
  57: (8)                     self.protection = protection or Protection()
  58: (8)                     self.builtinId = builtinId
  59: (8)                     self.hidden = hidden
  60: (8)                     self._wb = None
  61: (8)                     self._style = StyleArray()
  62: (4)                 def __setattr__(self, attr, value):
  63: (8)                     super().__setattr__(attr, value)
  64: (8)                     if getattr(self, '_wb', None) and attr in (
  65: (11)                       'font', 'fill', 'border', 'alignment', 'number_format',
'protection',
  66: (12)                        ):
  67: (12)                        self._recalculate()
  68: (4)                 def __iter__(self):
  69: (8)                     for key in ('name', 'builtinId', 'hidden', 'xfId'):
  70: (12)                        value = getattr(self, key, None)
  71: (12)                        if value is not None:
  72: (16)                            yield key, safe_string(value)
  73: (4)                 def bind(self, wb):
  74: (8)                     """
  75: (8)                     Bind a named style to a workbook
  76: (8)                     """
  77: (8)                     self._wb = wb
  78: (8)                     self._recalculate()
  79: (4)                 def _recalculate(self):
  80: (8)                     self._style.fontId =  self._wb._fonts.add(self.font)
  81: (8)                     self._style.borderId = self._wb._borders.add(self.border)
  82: (8)                     self._style.fillId =  self._wb._fills.add(self.fill)
  83: (8)                     self._style.protectionId = self._wb._protections.add(self.protection)
  84: (8)                     self._style.alignmentId = self._wb._alignments.add(self.alignment)
  85: (8)                     fmt = self.number_format
  86: (8)                     if fmt in BUILTIN_FORMATS_REVERSE:
  87: (12)                        fmt = BUILTIN_FORMATS_REVERSE[fmt]
  88: (8)                     else:
  89: (12)                        fmt = self._wb._number_formats.add(self.number_format) + (
  90: (18)                              BUILTIN_FORMATS_MAX_SIZE)
  91: (8)                     self._style.numFmtId = fmt
  92: (4)                 def as_tuple(self):
  93: (8)                     """Return a style array representing the current style"""
  94: (8)                     return self._style
  95: (4)                 def as_xf(self):
  96: (8)                     """
  97: (8)                     Return equivalent XfStyle
```

```
98: (8)                     """
99: (8)                     xf = CellStyle.from_array(self._style)
100: (8)                    xf.xfId = None
101: (8)                    xf.pivotButton = None
102: (8)                    xf.quotePrefix = None
103: (8)                    if self.alignment != Alignment():
104: (12)                       xf.alignment = self.alignment
105: (8)                    if self.protection != Protection():
106: (12)                       xf.protection = self.protection
107: (8)                    return xf
108: (4)                def as_name(self):
109: (8)                    """
110: (8)                    Return relevant named style
111: (8)                    """
112: (8)                    named = _NamedCellStyle(
113: (12)                       name=self.name,
114: (12)                       builtinId=self.builtinId,
115: (12)                       hidden=self.hidden,
116: (12)                       xfId=self._style.xfId
117: (8)                    )
118: (8)                    return named
119: (0)            class NamedStyleList(list):
120: (4)                """
121: (4)                Named styles are editable and can be applied to multiple objects
122: (4)                As only the index is stored in referencing objects the order mus
123: (4)                be preserved.
124: (4)                Returns a list of NamedStyles
125: (4)                """
126: (4)                def __init__(self, iterable=()):
127: (8)                    """
128: (8)                    Allow a list of named styles to be passed in and index them.
129: (8)                    """
130: (8)                    for idx, s in enumerate(iterable, len(self)):
131: (12)                       s._style.xfId = idx
132: (8)                    super().__init__(iterable)
133: (4)                @property
134: (4)                def names(self):
135: (8)                    return [s.name for s in self]
136: (4)                def __getitem__(self, key):
137: (8)                    if isinstance(key, int):
138: (12)                       return super().__getitem__(key)
139: (8)                    for idx, name in enumerate(self.names):
140: (12)                       if name == key:
141: (16)                           return self[idx]
142: (8)                    raise KeyError("No named style with the name{0} exists".format(key))
143: (4)                def append(self, style):
144: (8)                    if not isinstance(style, NamedStyle):
145: (12)                       raise TypeError("""Only NamedStyle instances can be added""")
146: (8)                    elif style.name in self.names: # hotspot
147: (12)                       raise ValueError("""Style {0} exists
already""".format(style.name))
148: (8)                    style._style.xfId = (len(self))
149: (8)                    super().append(style)
150: (0)            class _NamedCellStyle(Serialisable):
151: (4)                """
152: (4)                Pointer-based representation of named styles in XML
153: (4)                xfId refers to the corresponding CellStyleXfs
154: (4)                Not used in client code.
155: (4)                """
156: (4)                tagname = "cellStyle"
157: (4)                name = String()
158: (4)                xfId = Integer()
159: (4)                builtinId = Integer(allow_none=True)
160: (4)                iLevel = Integer(allow_none=True)
161: (4)                hidden = Bool(allow_none=True)
162: (4)                customBuiltin = Bool(allow_none=True)
163: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
164: (4)                __elements__ = ()
165: (4)                def __init__(self,
166: (17)                               name=None,
167: (17)                               xfId=None,
168: (17)                               builtinId=None,
169: (17)                               iLevel=None,
170: (17)                               hidden=None,
171: (17)                               customBuiltin=None,
172: (17)                               extLst=None,
173: (16)                              ):
174: (8)                         self.name = name
175: (8)                         self.xfId = xfId
176: (8)                         self.builtinId = builtinId
177: (8)                         self.iLevel = iLevel
178: (8)                         self.hidden = hidden
179: (8)                         self.customBuiltin = customBuiltin
180: (0)                 class _NamedCellStyleList(Serialisable):
181: (4)                     """
182: (4)                     Container for named cell style objects
183: (4)                     Not used in client code
184: (4)                     """
185: (4)                     tagname = "cellStyles"
186: (4)                     count = Integer(allow_none=True)
187: (4)                     cellStyle = Sequence(expected_type=_NamedCellStyle)
188: (4)                     __attrs__ = ("count",)
189: (4)                     def __init__(self,
190: (17)                               count=None,
191: (17)                               cellStyle=(),
192: (16)                              ):
193: (8)                         self.cellStyle = cellStyle
194: (4)                     @property
195: (4)                     def count(self):
196: (8)                         return len(self.cellStyle)
197: (4)                     def remove_duplicates(self):
198: (8)                         """
199: (8)                         Some applications contain duplicate definitions either by name or
200: (8)                         referenced style.
201: (8)                         As the references are 0-based indices, styles are sorted by
202: (8)                         index.
203: (8)                         Returns a list of style references with duplicates removed
204: (8)                         """
205: (8)                         def sort_fn(v):
206: (12)                            return v.xfId
207: (8)                         styles = []
208: (8)                         names = set()
209: (8)                         ids = set()
210: (8)                         for ns in sorted(self.cellStyle, key=sort_fn):
211: (12)                            if ns.xfId in ids or ns.name in names: # skip duplicates
212: (16)                                continue
213: (12)                            ids.add(ns.xfId)
214: (12)                            names.add(ns.name)
215: (12)                            styles.append(ns)
216: (8)                         return styles

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

File 126 - table.py:

1: (0)                   from openpyxl.descriptors.serialisable import Serialisable
2: (0)                   from openpyxl.descriptors import (
3: (4)                       Typed,
4: (4)                       Float,
5: (4)                       Bool,
6: (4)                       Set,
7: (4)                       Integer,
8: (4)                       NoneSet,
9: (4)                       String,
10: (4)                      Sequence
11: (0)                  )
12: (0)                  from .colors import Color
13: (0)                  class TableStyleElement(Serialisable):
```

```
14: (4)                 tagname = "tableStyleElement"
15: (4)                 type = Set(values=(['wholeTable', 'headerRow', 'totalRow', 'firstColumn',
16: (24)                                    'lastColumn', 'firstRowStripe', 'secondRowStripe',
'firstColumnStripe',
17: (24)                                    'secondColumnStripe', 'firstHeaderCell',
'lastHeaderCell',
18: (24)                                    'firstTotalCell', 'lastTotalCell',
'firstSubtotalColumn',
19: (24)                                    'secondSubtotalColumn', 'thirdSubtotalColumn',
'firstSubtotalRow',
20: (24)                                    'secondSubtotalRow', 'thirdSubtotalRow', 'blankRow',
21: (24)                                    'firstColumnSubheading', 'secondColumnSubheading',
22: (24)                                    'thirdColumnSubheading', 'firstRowSubheading',
'secondRowSubheading',
23: (24)                                    'thirdRowSubheading', 'pageFieldLabels',
'pageFieldValues']))
24: (4)                 size = Integer(allow_none=True)
25: (4)                 dxfId = Integer(allow_none=True)
26: (4)                 def __init__(self,
27: (17)                             type=None,
28: (17)                             size=None,
29: (17)                             dxfId=None,
30: (16)                            ):
31: (8)                     self.type = type
32: (8)                     self.size = size
33: (8)                     self.dxfId = dxfId
34: (0)             class TableStyle(Serialisable):
35: (4)                 tagname = "tableStyle"
36: (4)                 name = String()
37: (4)                 pivot = Bool(allow_none=True)
38: (4)                 table = Bool(allow_none=True)
39: (4)                 count = Integer(allow_none=True)
40: (4)                 tableStyleElement = Sequence(expected_type=TableStyleElement,
allow_none=True)
41: (4)                 __elements__ = ('tableStyleElement',)
42: (4)                 def __init__(self,
43: (17)                             name=None,
44: (17)                             pivot=None,
45: (17)                             table=None,
46: (17)                             count=None,
47: (17)                             tableStyleElement=(),
48: (16)                            ):
49: (8)                     self.name = name
50: (8)                     self.pivot = pivot
51: (8)                     self.table = table
52: (8)                     self.count = count
53: (8)                     self.tableStyleElement = tableStyleElement
54: (0)             class TableStyleList(Serialisable):
55: (4)                 tagname = "tableStyles"
56: (4)                 defaultTableStyle = String(allow_none=True)
57: (4)                 defaultPivotStyle = String(allow_none=True)
58: (4)                 tableStyle = Sequence(expected_type=TableStyle, allow_none=True)
59: (4)                 __elements__ = ('tableStyle',)
60: (4)                 __attrs__ = ("count", "defaultTableStyle", "defaultPivotStyle")
61: (4)                 def __init__(self,
62: (17)                             count=None,
63: (17)                             defaultTableStyle="TableStyleMedium9",
64: (17)                             defaultPivotStyle="PivotStyleLight16",
65: (17)                             tableStyle=(),
66: (16)                            ):
67: (8)                     self.defaultTableStyle = defaultTableStyle
68: (8)                     self.defaultPivotStyle = defaultPivotStyle
69: (8)                     self.tableStyle = tableStyle
70: (4)                 @property
71: (4)                 def count(self):
72: (8)                     return len(self.tableStyle)

----------------------------------------
```

File 127 - cell.py:

```
"""
Collection of utilities used within the package and also available for client code
"""
from functools import lru_cache
from itertools import chain, product
from string import ascii_uppercase, digits
import re
from .exceptions import CellCoordinatesException
COORD_RE = re.compile(r'^[$]?([A-Za-z]{1,3})[$]?(\d+)$')
COL_RANGE = """[A-Z]{1,3}:[A-Z]{1,3}:"""
ROW_RANGE = r"""\d+:\d+:"""
RANGE_EXPR = r"""
[$]?(?P<min_col>[A-Za-z]{1,3})?
[$]?(?P<min_row>\d+)?
(:[$]?(?P<max_col>[A-Za-z]{1,3})?
[$]?(?P<max_row>\d+)?)?
"""
ABSOLUTE_RE = re.compile('^' + RANGE_EXPR +'$', re.VERBOSE)
SHEET_TITLE = r"""
(('(?P<quoted>([^']|'')*)')|(?P<notquoted>[^'^ ^!]*))!"""
SHEETRANGE_RE = re.compile("""{0}(?P<cells>{1})(?=,?)""".format(
    SHEET_TITLE, RANGE_EXPR), re.VERBOSE)
def get_column_interval(start, end):
    """
    Given the start and end columns, return all the columns in the series.
    The start and end columns can be either column letters or 1-based
    indexes.
    """
    if isinstance(start, str):
        start = column_index_from_string(start)
    if isinstance(end, str):
        end = column_index_from_string(end)
    return [get_column_letter(x) for x in range(start, end + 1)]
def coordinate_from_string(coord_string):
    """Convert a coordinate string like 'B12' to a tuple ('B', 12)"""
    match = COORD_RE.match(coord_string)
    if not match:
        msg = f"Invalid cell coordinates ({coord_string})"
        raise CellCoordinatesException(msg)
    column, row = match.groups()
    row = int(row)
    if not row:
        msg = f"There is no row 0 ({coord_string})"
        raise CellCoordinatesException(msg)
    return column, row
def absolute_coordinate(coord_string):
    """Convert a coordinate to an absolute coordinate string (B12 -> $B$12)"""
    m = ABSOLUTE_RE.match(coord_string)
    if not m:
        raise ValueError(f"{coord_string} is not a valid coordinate range")
    d = m.groupdict('')
    for k, v in d.items():
        if v:
            d[k] = f"${v}"
    if d['max_col'] or d['max_row']:
        fmt = "{min_col}{min_row}:{max_col}{max_row}"
    else:
        fmt = "{min_col}{min_row}"
    return fmt.format(**d)
__decimal_to_alpha = [""] + list(ascii_uppercase)
@lru_cache(maxsize=None)
def get_column_letter(col_idx):
    """
    Convert decimal column position to its ASCII (base 26) form.
    Because column indices are 1-based, strides are actually pow(26, n) + 26
    Hence, a correction is applied between pow(26, n) and pow(26, 2) + 26 to
```

```
67: (4)                 prevent and additional column letter being prepended
68: (4)                 "A" == 1 == pow(26, 0)
69: (4)                 "Z" == 26 == pow(26, 0) + 26 // decimal equivalent 10
70: (4)                 "AA" == 27 == pow(26, 1) + 1
71: (4)                 "ZZ" == 702 == pow(26, 2) + 26 // decimal equivalent 100
72: (4)                 """
73: (4)                 if not 1 <= col_idx <= 18278:
74: (8)                     raise ValueError("Invalid column index {0}".format(col_idx))
75: (4)                 result = []
76: (4)                 if col_idx < 26:
77: (8)                     return __decimal_to_alpha[col_idx]
78: (4)                 while col_idx:
79: (8)                     col_idx, remainder = divmod(col_idx, 26)
80: (8)                     result.insert(0, __decimal_to_alpha[remainder])
81: (8)                     if not remainder:
82: (12)                        col_idx -= 1
83: (12)                        result.insert(0, "Z")
84: (4)                 return "".join(result)
85: (0)             __alpha_to_decimal = {letter:pos for pos, letter in enumerate(ascii_uppercase, 1)}
86: (0)             __powers = (1, 26, 676)
87: (0)             @lru_cache(maxsize=None)
88: (0)             def column_index_from_string(col):
89: (4)                 """
90: (4)                 Convert ASCII column name (base 26) to decimal with 1-based index
91: (4)                 Characters represent descending multiples of powers of 26
92: (4)                 "AFZ" == 26 * pow(26, 0) + 6 * pow(26, 1) + 1 * pow(26, 2)
93: (4)                 """
94: (4)                 error_msg = f"'{col}' is not a valid column name. Column names are from A to ZZZ"
95: (4)                 if len(col) > 3:
96: (8)                     raise ValueError(error_msg)
97: (4)                 idx = 0
98: (4)                 col = reversed(col.upper())
99: (4)                 for letter, power in zip(col, __powers):
100: (8)                    try:
101: (12)                       pos = __alpha_to_decimal[letter]
102: (8)                    except KeyError:
103: (12)                       raise ValueError(error_msg)
104: (8)                    idx += pos * power
105: (4)                if not 0 < idx < 18279:
106: (8)                    raise ValueError(error_msg)
107: (4)                return idx
108: (0)            def range_boundaries(range_string):
109: (4)                """
110: (4)                Convert a range string into a tuple of boundaries:
111: (4)                (min_col, min_row, max_col, max_row)
112: (4)                Cell coordinates will be converted into a range with the cell at both end
113: (4)                """
114: (4)                msg = "{0} is not a valid coordinate or range".format(range_string)
115: (4)                m = ABSOLUTE_RE.match(range_string)
116: (4)                if not m:
117: (8)                    raise ValueError(msg)
118: (4)                min_col, min_row, sep, max_col, max_row = m.groups()
119: (4)                if sep:
120: (8)                    cols = min_col, max_col
121: (8)                    rows = min_row, max_row
122: (8)                    if not (
123: (12)                       all(cols + rows) or
124: (12)                       all(cols) and not any(rows) or
125: (12)                       all(rows) and not any(cols)
126: (8)                    ):
127: (12)                       raise ValueError(msg)
128: (4)                if min_col is not None:
129: (8)                    min_col = column_index_from_string(min_col)
130: (4)                if min_row is not None:
131: (8)                    min_row = int(min_row)
132: (4)                if max_col is not None:
133: (8)                    max_col = column_index_from_string(max_col)
```

```
(4)     else:
(8)         max_col = min_col
(4)     if max_row is not None:
(8)         max_row = int(max_row)
(4)     else:
(8)         max_row = min_row
(4)     return min_col, min_row, max_col, max_row
(0) def rows_from_range(range_string):
(4)     """
(4)     Get individual addresses for every cell in a range.
(4)     Yields one row at a time.
(4)     """
(4)     min_col, min_row, max_col, max_row = range_boundaries(range_string)
(4)     rows = range(min_row, max_row + 1)
(4)     cols = [get_column_letter(col) for col in range(min_col, max_col + 1)]
(4)     for row in rows:
(8)         yield tuple('{0}{1}'.format(col, row) for col in cols)
(0) def cols_from_range(range_string):
(4)     """
(4)     Get individual addresses for every cell in a range.
(4)     Yields one row at a time.
(4)     """
(4)     min_col, min_row, max_col, max_row = range_boundaries(range_string)
(4)     rows = range(min_row, max_row+1)
(4)     cols = (get_column_letter(col) for col in range(min_col, max_col+1))
(4)     for col in cols:
(8)         yield tuple('{0}{1}'.format(col, row) for row in rows)
(0) def coordinate_to_tuple(coordinate):
(4)     """
(4)     Convert an Excel style coordinate to (row, column) tuple
(4)     """
(4)     for idx, c in enumerate(coordinate):
(8)         if c in digits:
(12)            break
(4)     col = coordinate[:idx]
(4)     row = coordinate[idx:]
(4)     return int(row), column_index_from_string(col)
(0) def range_to_tuple(range_string):
(4)     """
(4)     Convert a worksheet range to the sheetname and maximum and minimum
(4)     coordinate indices
(4)     """
(4)     m = SHEETRANGE_RE.match(range_string)
(4)     if m is None:
(8)         raise ValueError("Value must be of the form sheetname!A1:E4")
(4)     sheetname = m.group("quoted") or m.group("notquoted")
(4)     cells = m.group("cells")
(4)     boundaries = range_boundaries(cells)
(4)     return sheetname, boundaries
(0) def quote_sheetname(sheetname):
(4)     """
(4)     Add quotes around sheetnames if they contain spaces.
(4)     """
(4)     if "'" in sheetname:
(8)         sheetname = sheetname.replace("'", "''")
(4)     sheetname = u"'{0}'".format(sheetname)
(4)     return sheetname
```

File 128 - escape.py:

```
(0) """
(0) OOXML has non-standard escaping for characters < \031
(0) """
(0) import re
(0) def escape(value):
(4)     r"""
(4)     Convert ASCII < 31 to OOXML: \n == _x + hex(ord(\n)) + _
```

```
8: (4)                  """
9: (4)                  CHAR_REGEX = re.compile(r"[\001-\031]")
10: (4)                 def _sub(match):
11: (8)                     """
12: (8)                     Callback to escape chars
13: (8)                     """
14: (8)                     return "_x{:0>4x}_".format(ord(match.group(0)))
15: (4)                 return CHAR_REGEX.sub(_sub, value)
16: (0)             def unescape(value):
17: (4)                 r"""
18: (4)                 Convert escaped strings to ASCIII: _x000a_ == \n
19: (4)                 """
20: (4)                 ESCAPED_REGEX = re.compile("_x([0-9A-Fa-f]{4})_")
21: (4)                 def _sub(match):
22: (8)                     """
23: (8)                     Callback to unescape chars
24: (8)                     """
25: (8)                     return chr(int(match.group(1), 16))
26: (4)                 if "_x" in value:
27: (8)                     value = ESCAPED_REGEX.sub(_sub, value)
28: (4)                 return value

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

File 129 - datetime.py:

1: (0)              """Manage Excel date weirdness."""
2: (0)              import datetime
3: (0)              from math import isnan
4: (0)              import re
5: (0)              MAC_EPOCH = datetime.datetime(1904, 1, 1)
6: (0)              WINDOWS_EPOCH = datetime.datetime(1899, 12, 30)
7: (0)              CALENDAR_WINDOWS_1900 = 2415018.5   # Julian date of WINDOWS_EPOCH
8: (0)              CALENDAR_MAC_1904 = 2416480.5       # Julian date of MAC_EPOCH
9: (0)              CALENDAR_WINDOWS_1900 = WINDOWS_EPOCH
10: (0)             CALENDAR_MAC_1904 = MAC_EPOCH
11: (0)             SECS_PER_DAY = 86400
12: (0)             ISO_FORMAT = '%Y-%m-%dT%H:%M:%SZ'
13: (0)             ISO_REGEX = re.compile(r'''
14: (0)             (?P<date>(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2}))?T?
15: (0)             (?P<time>(?P<hour>\d{2}):(?P<minute>\d{2})(:(?P<second>\d{2})(?
P<microsecond>\.\d{1,3})?)?)?Z?''',
16: (39)                                                    re.VERBOSE)
17: (0)             ISO_DURATION = re.compile(r'PT((?P<hours>\d+)H)?((?P<minutes>\d+)M)?((?
P<seconds>\d+(\.\d{1,3})?)S)?')
18: (0)             def to_ISO8601(dt):
19: (4)                 """Convert from a datetime to a timestamp string."""
20: (4)                 if hasattr(dt, "microsecond") and dt.microsecond:
21: (8)                     return dt.isoformat(timespec="milliseconds")
22: (4)                 return dt.isoformat()
23: (0)             def from_ISO8601(formatted_string):
24: (4)                 """Convert from a timestamp string to a datetime object. According to
25: (4)                 18.17.4 in the specification the following ISO 8601 formats are
26: (4)                 supported.
27: (4)                 Dates B.1.1 and B.2.1
28: (4)                 Times B.1.2 and B.2.2
29: (4)                 Datetimes B.1.3 and B.2.3
30: (4)                 There is no concept of timedeltas in the specification, but Excel
31: (4)                 writes them (in strict OOXML mode), so these are also understood.
32: (4)                 """
33: (4)                 if not formatted_string:
34: (8)                     return None
35: (4)                 match = ISO_REGEX.match(formatted_string)
36: (4)                 if match and any(match.groups()):
37: (8)                     parts = match.groupdict(0)
38: (8)                     for key in ["year", "month", "day", "hour", "minute", "second"]:
39: (12)                        if parts[key]:
40: (16)                            parts[key] = int(parts[key])
41: (8)                     if parts["microsecond"]:
```

```
42: (12)                        parts["microsecond"] = int(float(parts['microsecond']) *
1_000_000)
43: (8)                     if not parts["date"]:
44: (12)                        dt = datetime.time(parts['hour'], parts['minute'],
parts['second'], parts["microsecond"])
45: (8)                     elif not parts["time"]:
46: (12)                        dt = datetime.date(parts['year'], parts['month'], parts['day'])
47: (8)                     else:
48: (12)                        del parts["time"]
49: (12)                        del parts["date"]
50: (12)                        dt = datetime.datetime(**parts)
51: (8)                     return dt
52: (4)                 match = ISO_DURATION.match(formatted_string)
53: (4)                 if match and any(match.groups()):
54: (8)                     parts = match.groupdict(0)
55: (8)                     for key, val in parts.items():
56: (12)                        if val:
57: (16)                            parts[key] = float(val)
58: (8)                     return datetime.timedelta(**parts)
59: (4)                 raise ValueError("Invalid datetime value {}".format(formatted_string))
60: (0)             def to_excel(dt, epoch=WINDOWS_EPOCH):
61: (4)                 """Convert Python datetime to Excel serial"""
62: (4)                 if isinstance(dt, datetime.time):
63: (8)                     return time_to_days(dt)
64: (4)                 if isinstance(dt, datetime.timedelta):
65: (8)                     return timedelta_to_days(dt)
66: (4)                 if isnan(dt.year):  # Pandas supports Not a Date
67: (8)                     return
68: (4)                 if not hasattr(dt, "date"):
69: (8)                     dt = datetime.datetime.combine(dt, datetime.time())
70: (4)                 days = (dt - epoch).days
71: (4)                 if 0 < days <= 60 and epoch == WINDOWS_EPOCH:
72: (8)                     days -= 1
73: (4)                 return days + time_to_days(dt)
74: (0)             def from_excel(value, epoch=WINDOWS_EPOCH, timedelta=False):
75: (4)                 """Convert Excel serial to Python datetime"""
76: (4)                 if value is None:
77: (8)                     return
78: (4)                 if timedelta:
79: (8)                     td = datetime.timedelta(days=value)
80: (8)                     if td.microseconds:
81: (12)                        td = datetime.timedelta(seconds=td.total_seconds() // 1,
82: (36)                                                microseconds=round(td.microseconds, -3))
83: (8)                     return td
84: (4)                 day, fraction = divmod(value, 1)
85: (4)                 diff = datetime.timedelta(milliseconds=round(fraction * SECS_PER_DAY *
1000))
86: (4)                 if 0 <= value < 1 and diff.days == 0:
87: (8)                     return days_to_time(diff)
88: (4)                 if 0 < value < 60 and epoch == WINDOWS_EPOCH:
89: (8)                     day += 1
90: (4)                 return epoch + datetime.timedelta(days=day) + diff
91: (0)             def time_to_days(value):
92: (4)                 """Convert a time value to fractions of day"""
93: (4)                 return (
94: (8)                     (value.hour * 3600)
95: (8)                     + (value.minute * 60)
96: (8)                     + value.second
97: (8)                     + value.microsecond / 10**6
98: (8)                     ) / SECS_PER_DAY
99: (0)             def timedelta_to_days(value):
100: (4)                """Convert a timedelta value to fractions of a day"""
101: (4)                return value.total_seconds() / SECS_PER_DAY
102: (0)            def days_to_time(value):
103: (4)                mins, seconds = divmod(value.seconds, 60)
104: (4)                hours, mins = divmod(mins, 60)
105: (4)                return datetime.time(hours, mins, seconds, value.microseconds)

----------------------------------------
```

```
File 130 - __init__.py:

1: (0)              from .cell import (
2: (4)                  absolute_coordinate,
3: (4)                  cols_from_range,
4: (4)                  column_index_from_string,
5: (4)                  coordinate_to_tuple,
6: (4)                  get_column_letter,
7: (4)                  get_column_interval,
8: (4)                  quote_sheetname,
9: (4)                  range_boundaries,
10: (4)                 range_to_tuple,
11: (4)                 rows_from_range,
12: (0)             )
13: (0)             from .formulas import FORMULAE

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

File 131 - dataframe.py:

1: (0)              from itertools import accumulate
2: (0)              import operator
3: (0)              import numpy
4: (0)              from openpyxl.compat.product import prod
5: (0)              def dataframe_to_rows(df, index=True, header=True):
6: (4)                  """
7: (4)                  Convert a Pandas dataframe into something suitable for passing into a
worksheet.
8: (4)                  If index is True then the index will be included, starting one row below
the header.
9: (4)                  If header is True then column headers will be included starting one column
to the right.
10: (4)                 Formatting should be done by client code.
11: (4)                 """
12: (4)                 from pandas import Timestamp
13: (4)                 if header:
14: (8)                     if df.columns.nlevels > 1:
15: (12)                        rows = expand_index(df.columns, header)
16: (8)                     else:
17: (12)                        rows = [list(df.columns.values)]
18: (8)                     for row in rows:
19: (12)                        n = []
20: (12)                        for v in row:
21: (16)                            if isinstance(v, numpy.datetime64):
22: (20)                                v = Timestamp(v)
23: (16)                            n.append(v)
24: (12)                        row = n
25: (12)                        if index:
26: (16)                            row = [None]*df.index.nlevels + row
27: (12)                        yield row
28: (4)                 if index:
29: (8)                     yield df.index.names
30: (4)                 expanded = ([v] for v in df.index)
31: (4)                 if df.index.nlevels > 1:
32: (8)                     expanded = expand_index(df.index)
33: (4)                 for (df_index, row) in zip(expanded, df.itertuples(index=False)):
34: (8)                     row = list(row)
35: (8)                     if index:
36: (12)                        row = df_index + row
37: (8)                     yield row
38: (0)             def expand_index(index, header=False):
39: (4)                 """
40: (4)                 Expand axis or column Multiindex
41: (4)                 For columns use header = True
42: (4)                 For axes use header = False (default)
43: (4)                 """
44: (4)                 values = list(index.values)
45: (4)                 previous_value = [None] * len(values[0])
```

```
46: (4)                 result = []
47: (4)                 for value in values:
48: (8)                     row = [None] * len(value)
49: (8)                     prior_change = False
50: (8)                     for idx, (current_index_member, previous_index_member) in
enumerate(zip(value, previous_value)):
51: (12)                        if current_index_member != previous_index_member or prior_change:
52: (16)                            row[idx] = current_index_member
53: (16)                            prior_change = True
54: (8)                     previous_value = value
55: (8)                     if not header:
56: (12)                        yield row
57: (8)                     else:
58: (12)                        result.append(row)
59: (4)                 if header:
60: (8)                     result = numpy.array(result).transpose().tolist()
61: (8)                     for row in result:
62: (12)                        yield row

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

File 132 - exceptions.py:

1: (0)              """Definitions for openpyxl shared exception classes."""
2: (0)              class CellCoordinatesException(Exception):
3: (4)                  """Error for converting between numeric and A1-style cell references."""
4: (0)              class IllegalCharacterError(Exception):
5: (4)                  """The data submitted which cannot be used directly in Excel files. It
6: (4)                  must be removed or escaped."""
7: (0)              class NamedRangeException(Exception):
8: (4)                  """Error for badly formatted named ranges."""
9: (0)              class SheetTitleException(Exception):
10: (4)                 """Error for bad sheet names."""
11: (0)             class InvalidFileException(Exception):
12: (4)                 """Error for trying to open a non-ooxml file."""
13: (0)             class ReadOnlyWorkbookException(Exception):
14: (4)                 """Error for trying to modify a read-only workbook"""
15: (0)             class WorkbookAlreadySaved(Exception):
16: (4)                 """Error when attempting to perform operations on a dump workbook
17: (4)                 while it has already been dumped once"""

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

File 133 - bound_dictionary.py:

1: (0)              from collections import defaultdict
2: (0)              class BoundDictionary(defaultdict):
3: (4)                  """
4: (4)                  A default dictionary where elements are tightly coupled.
5: (4)                  The factory method is responsible for binding the parent object to the
child.
6: (4)                  If a reference attribute is assigned then child objects will have the key
assigned to this.
7: (4)                  Otherwise it's just a defaultdict.
8: (4)                  """
9: (4)                  def __init__(self, reference=None, *args, **kw):
10: (8)                     self.reference = reference
11: (8)                     super().__init__(*args, **kw)
12: (4)                 def __getitem__(self, key):
13: (8)                     value = super().__getitem__(key)
14: (8)                     if self.reference is not None:
15: (12)                        setattr(value, self.reference, key)
16: (8)                     return value

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

File 134 - formulas.py:

1: (0)              """
```

```
2: (0)              List of builtin formulae
3: (0)              """
4: (0)              FORMULAE = ("CUBEKPIMEMBER", "CUBEMEMBER", "CUBEMEMBERPROPERTY",
"CUBERANKEDMEMBER", "CUBESET", "CUBESETCOUNT", "CUBEVALUE", "DAVERAGE", "DCOUNT", "DCOUNTA",
"DGET", "DMAX", "DMIN", "DPRODUCT", "DSTDEV", "DSTDEVP", "DSUM", "DVAR", "DVARP", "DATE",
"DATEDIF", "DATEVALUE", "DAY", "DAYS360", "EDATE", "EOMONTH", "HOUR", "MINUTE", "MONTH",
"NETWORKDAYS", "NETWORKDAYS.INTL", "NOW", "SECOND", "TIME", "TIMEVALUE", "TODAY", "WEEKDAY",
"WEEKNUM", "WORKDAY", "WORKDAY.INTL", "YEAR", "YEARFRAC", "BESSELI", "BESSELJ", "BESSELK",
"BESSELY", "BIN2DEC", "BIN2HEX", "BIN2OCT", "COMPLEX", "CONVERT", "DEC2BIN", "DEC2HEX", "DEC2OCT",
"DELTA", "ERF", "ERFC", "GESTEP", "HEX2BIN", "HEX2DEC", "HEX2OCT", "IMABS", "IMAGINARY",
"IMARGUMENT", "IMCONJUGATE", "IMCOS", "IMDIV", "IMEXP", "IMLN", "IMLOG10", "IMLOG2", "IMPOWER",
"IMPRODUCT", "IMREAL", "IMSIN", "IMSQRT", "IMSUB", "IMSUM", "OCT2BIN", "OCT2DEC", "OCT2HEX",
"ACCRINT", "ACCRINTM", "AMORDEGRC", "AMORLINC", "COUPDAYBS", "COUPDAYS", "COUPDAYSNC", "COUPNCD",
"COUPNUM", "COUPPCD", "CUMIPMT", "CUMPRINC", "DB", "DDB", "DISC", "DOLLARDE", "DOLLARFR",
"DURATION", "EFFECT", "FV", "FVSCHEDULE", "INTRATE", "IPMT", "IRR", "ISPMT", "MDURATION", "MIRR",
"NOMINAL", "NPER", "NPV", "ODDFPRICE", "ODDFYIELD", "ODDLPRICE", "ODDLYIELD", "PMT", "PPMT",
"PRICE", "PRICEDISC", "PRICEMAT", "PV", "RATE", "RECEIVED", "SLN", "SYD", "TBILLEQ", "TBILLPRICE",
"TBILLYIELD", "VDB", "XIRR", "XNPV", "YIELD", "YIELDDISC", "YIELDMAT", "CELL", "ERROR.TYPE",
"INFO", "ISBLANK", "ISERR", "ISERROR", "ISEVEN", "ISLOGICAL", "ISNA", "ISNONTEXT", "ISNUMBER",
"ISODD", "ISREF", "ISTEXT", "N", "NA", "TYPE", "AND", "FALSE", "IF", "IFERROR", "NOT", "OR",
"TRUE", "ADDRESS", "AREAS", "CHOOSE", "COLUMN", "COLUMNS", "GETPIVOTDATA", "HLOOKUP", "HYPERLINK",
"INDEX", "INDIRECT", "LOOKUP", "MATCH", "OFFSET", "ROW", "ROWS", "RTD", "TRANSPOSE", "VLOOKUP",
"ABS", "ACOS", "ACOSH", "ASIN", "ASINH", "ATAN", "ATAN2", "ATANH", "CEILING", "COMBIN", "COS",
"COSH", "DEGREES", "ECMA.CEILING", "EVEN", "EXP", "FACT", "FACTDOUBLE", "FLOOR", "GCD", "INT",
"ISO.CEILING", "LCM", "LN", "LOG", "LOG10", "MDETERM", "MINVERSE", "MMULT", "MOD", "MROUND",
"MULTINOMIAL", "ODD", "PI", "POWER", "PRODUCT", "QUOTIENT", "RADIANS", "RAND", "RANDBETWEEN",
"ROMAN", "ROUND", "ROUNDDOWN", "ROUNDUP", "SERIESSUM", "SIGN", "SIN", "SINH", "SQRT", "SQRTPI",
"SUBTOTAL", "SUM", "SUMIF", "SUMIFS", "SUMPRODUCT", "SUMSQ", "SUMX2MY2", "SUMX2PY2", "SUMXMY2",
"TAN", "TANH", "TRUNC", "AVEDEV", "AVERAGE", "AVERAGEA", "AVERAGEIF", "AVERAGEIFS", "BETADIST",
"BETAINV", "BINOMDIST", "CHIDIST", "CHIINV", "CHITEST", "CONFIDENCE", "CORREL", "COUNT", "COUNTA",
"COUNTBLANK", "COUNTIF", "COUNTIFS", "COVAR", "CRITBINOM", "DEVSQ", "EXPONDIST", "FDIST", "FINV",
"FISHER", "FISHERINV", "FORECAST", "FREQUENCY", "FTEST", "GAMMADIST", "GAMMAINV", "GAMMALN",
"GEOMEAN", "GROWTH", "HARMEAN", "HYPGEOMDIST", "INTERCEPT", "KURT", "LARGE", "LINEST", "LOGEST",
"LOGINV", "LOGNORMDIST", "MAX", "MAXA", "MEDIAN", "MIN", "MINA", "MODE", "NEGBINOMDIST",
"NORMDIST", "NORMINV", "NORMSDIST", "NORMSINV", "PEARSON", "PERCENTILE", "PERCENTRANK", "PERMUT",
"POISSON", "PROB", "QUARTILE", "RANK", "RSQ", "SKEW", "SLOPE", "SMALL", "STANDARDIZE", "STDEV",
"STDEVA", "STDEVP", "STDEVPA", "STEYX", "TDIST", "TINV", "TREND", "TRIMMEAN", "TTEST", "VAR",
"VARA", "VARP", "VARPA", "WEIBULL", "ZTEST", "ASC", "BAHTTEXT", "CHAR", "CLEAN", "CODE",
"CONCATENATE", "DOLLAR", "EXACT", "FIND", "FINDB", "FIXED", "JIS", "LEFT", "LEFTB", "LEN", "LENB",
"LOWER", "MID", "MIDB", "PHONETIC", "PROPER", "REPLACE", "REPLACEB", "REPT", "RIGHT", "RIGHTB",
"SEARCH", "SEARCHB", "SUBSTITUTE", "T", "TEXT", "TRIM", "UPPER", "VALUE")
5: (0)              FORMULAE = frozenset(FORMULAE)
6: (0)              from openpyxl.formula import Tokenizer
7: (0)              def validate(formula):
8: (4)                  """
9: (4)                  Utility function for checking whether a formula is syntactically correct
10: (4)                 """
11: (4)                 assert formula.startswith("=")
12: (4)                 formula = Tokenizer(formula)
13: (4)                 for t in formula.items:
14: (8)                     if t.type == "FUNC" and t.subtype == "OPEN":
15: (12)                        if not t.value.startswith("_xlfn.") and t.value[:-1] not in
FORMULAE:
16: (16)                            raise ValueError(f"Unknown function {t.value} in
{formula.formula}. The function may need a prefix")

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

File 135 - web.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Sequence,
5: (4)                  String,
6: (4)                  Float,
7: (4)                  Integer,
8: (4)                  Bool,
9: (4)                  NoneSet,
)
class WebPublishObject(Serialisable):
    tagname = "webPublishingObject"
    id = Integer()
    divId = String()
    sourceObject = String(allow_none=True)
    destinationFile = String()
    title = String(allow_none=True)
    autoRepublish = Bool(allow_none=True)
    def __init__(self,
                 id=None,
                 divId=None,
                 sourceObject=None,
                 destinationFile=None,
                 title=None,
                 autoRepublish=None,
                ):
        self.id = id
        self.divId = divId
        self.sourceObject = sourceObject
        self.destinationFile = destinationFile
        self.title = title
        self.autoRepublish = autoRepublish
class WebPublishObjectList(Serialisable):
    tagname ="webPublishingObjects"
    count = Integer(allow_none=True)
    webPublishObject = Sequence(expected_type=WebPublishObject)
    __elements__ = ('webPublishObject',)
    def __init__(self,
                 count=None,
                 webPublishObject=(),
                ):
        self.webPublishObject = webPublishObject
    @property
    def count(self):
        return len(self.webPublishObject)
class WebPublishing(Serialisable):
    tagname = "webPublishing"
    css = Bool(allow_none=True)
    thicket = Bool(allow_none=True)
    longFileNames = Bool(allow_none=True)
    vml = Bool(allow_none=True)
    allowPng = Bool(allow_none=True)
    targetScreenSize = NoneSet(values=(['544x376', '640x480', '720x512', '800x600',
                                    '1024x768', '1152x882', '1152x900', '1280x1024', '1600x1200',
                                    '1800x1440', '1920x1200']))
    dpi = Integer(allow_none=True)
    codePage = Integer(allow_none=True)
    characterSet = String(allow_none=True)
    def __init__(self,
                 css=None,
                 thicket=None,
                 longFileNames=None,
                 vml=None,
                 allowPng=None,
                 targetScreenSize='800x600',
                 dpi=None,
                 codePage=None,
                 characterSet=None,
                ):
        self.css = css
        self.thicket = thicket
        self.longFileNames = longFileNames
        self.vml = vml
        self.allowPng = allowPng
        self.targetScreenSize = targetScreenSize
        self.dpi = dpi
```

```
77: (8)                     self.codePage = codePage
78: (8)                     self.characterSet = characterSet

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

File 136 - units.py:

1: (0)              import math
2: (0)              DEFAULT_ROW_HEIGHT = 15.  # Default row height measured in point size.
3: (0)              BASE_COL_WIDTH = 8 # in characters
4: (0)              DEFAULT_COLUMN_WIDTH = BASE_COL_WIDTH + 5
5: (0)              DEFAULT_LEFT_MARGIN = 0.7 # in inches, = right margin
6: (0)              DEFAULT_TOP_MARGIN = 0.7874 # in inches = bottom margin
7: (0)              DEFAULT_HEADER = 0.3 # in inches
8: (0)              """
9: (0)              From the ECMA Spec (4th Edition part 1)
10: (0)             Page setup: "Left Page Margin in inches" p. 1647
11: (0)             Docs from
12: (0)             http://startbigthinksmall.wordpress.com/2010/01/04/points-inches-and-emus-
measuring-units-in-office-open-xml/
13: (0)             See also http://msdn.microsoft.com/en-us/library/dd560821(v=office.12).aspx
14: (0)             dxa: The main unit in OOXML is a twentieth of a point. Also called twips.
15: (0)             pt: point. In Excel there are 72 points to an inch
16: (0)             hp: half-points are used to specify font sizes. A font-size of 12pt equals 24
half points
17: (0)             pct: Half-points are used to specify font sizes. A font-size of 12pt equals 24
half points
18: (0)             EMU: English Metric Unit, EMUs are used for coordinates in vector-based
19: (0)             drawings and embedded pictures. One inch equates to 914400 EMUs and a
20: (0)             centimeter is 360000. For bitmaps the default resolution is 96 dpi (known as
21: (0)             PixelsPerInch in Excel). Spec p. 1122
22: (0)             For radial geometry Excel uses integer units of 1/60000th of a degree.
23: (0)             """
24: (0)             def inch_to_dxa(value):
25: (4)                 """1 inch = 72 * 20 dxa"""
26: (4)                 return int(value * 20 * 72)
27: (0)             def dxa_to_inch(value):
28: (4)                 return value / 72 / 20
29: (0)             def dxa_to_cm(value):
30: (4)                 return 2.54 * dxa_to_inch(value)
31: (0)             def cm_to_dxa(value):
32: (4)                 emu = cm_to_EMU(value)
33: (4)                 inch = EMU_to_inch(emu)
34: (4)                 return inch_to_dxa(inch)
35: (0)             def pixels_to_EMU(value):
36: (4)                 """1 pixel = 9525 EMUs"""
37: (4)                 return int(value * 9525)
38: (0)             def EMU_to_pixels(value):
39: (4)                 return round(value / 9525)
40: (0)             def cm_to_EMU(value):
41: (4)                 """1 cm = 360000 EMUs"""
42: (4)                 return int(value * 360000)
43: (0)             def EMU_to_cm(value):
44: (4)                 return round(value / 360000, 4)
45: (0)             def inch_to_EMU(value):
46: (4)                 """1 inch = 914400 EMUs"""
47: (4)                 return int(value * 914400)
48: (0)             def EMU_to_inch(value):
49: (4)                 return round(value / 914400, 4)
50: (0)             def pixels_to_points(value, dpi=96):
51: (4)                 """96 dpi, 72i"""
52: (4)                 return value * 72 / dpi
53: (0)             def points_to_pixels(value, dpi=96):
54: (4)                 return int(math.ceil(value * dpi / 72))
55: (0)             def degrees_to_angle(value):
56: (4)                 """1 degree = 60000 angles"""
57: (4)                 return int(round(value * 60000))
58: (0)             def angle_to_degrees(value):
59: (4)                 return round(value / 60000, 2)
```

```
(0)              def short_color(color):
(4)                  """ format a color to its short size """
(4)                  if len(color) > 6:
(8)                      return color[2:]
(4)                  return color

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

File 137 - child.py:

(0)              import re
(0)              import warnings
(0)              from openpyxl.worksheet.header_footer import HeaderFooter
(0)              """
(0)              Base class for worksheets, chartsheets, etc. that can be added to workbooks
(0)              """
(0)              INVALID_TITLE_REGEX = re.compile(r'[\\*?:/\[\]]')
(0)              def avoid_duplicate_name(names, value):
(4)                  """
(4)                  Naive check to see whether name already exists.
(4)                  If name does exist suggest a name using an incrementer
(4)                  Duplicates are case insensitive
(4)                  """
(4)                  match = [n for n in names if n.lower() == value.lower()]
(4)                  if match:
(8)                      names = u",".join(names)
(8)                      sheet_title_regex = re.compile(f'(?P<title>{re.escape(value)})(?P<count>\\d*),?', re.I)
(8)                      matches = sheet_title_regex.findall(names)
(8)                      if matches:
(12)                         counts = [int(idx) for (t, idx) in matches if idx.isdigit()]
(12)                         highest = 0
(12)                         if counts:
(16)                             highest = max(counts)
(12)                         value = u"{0}{1}".format(value, highest + 1)
(4)                  return value
(0)              class _WorkbookChild:
(4)                  __title = ""
(4)                  _id = None
(4)                  _path = "{0}"
(4)                  _parent = None
(4)                  _default_title = "Sheet"
(4)                  def __init__(self, parent=None, title=None):
(8)                      self._parent = parent
(8)                      self.title = title or self._default_title
(8)                      self.HeaderFooter = HeaderFooter()
(4)                  def __repr__(self):
(8)                      return '<{0} "{1}">'.format(self.__class__.__name__, self.title)
(4)                  @property
(4)                  def parent(self):
(8)                      return self._parent
(4)                  @property
(4)                  def encoding(self):
(8)                      return self._parent.encoding
(4)                  @property
(4)                  def title(self):
(8)                      return self.__title
(4)                  @title.setter
(4)                  def title(self, value):
(8)                      """
(8)                      Set a sheet title, ensuring it is valid.
(8)                      Limited to 31 characters, no special characters.
(8)                      Duplicate titles will be incremented numerically
(8)                      """
(8)                      if not self._parent:
(12)                         return
(8)                      if not value:
(12)                         raise ValueError("Title must have at least one character")
(8)                      if hasattr(value, "decode"):
```

```
59: (12)                        if not isinstance(value, str):
60: (16)                            try:
61: (20)                                value = value.decode("ascii")
62: (16)                            except UnicodeDecodeError:
63: (20)                                raise ValueError("Worksheet titles must be str")
64: (8)                     m = INVALID_TITLE_REGEX.search(value)
65: (8)                     if m:
66: (12)                        msg = "Invalid character {0} found in sheet
title".format(m.group(0))
67: (12)                        raise ValueError(msg)
68: (8)                     if self.title is not None and self.title != value:
69: (12)                        value = avoid_duplicate_name(self.parent.sheetnames, value)
70: (8)                     if len(value) > 31:
71: (12)                        warnings.warn("Title is more than 31 characters. Some applications
may not be able to read the file")
72: (8)                     self.__title = value
73: (4)                 @property
74: (4)                 def oddHeader(self):
75: (8)                     return self.HeaderFooter.oddHeader
76: (4)                 @oddHeader.setter
77: (4)                 def oddHeader(self, value):
78: (8)                     self.HeaderFooter.oddHeader = value
79: (4)                 @property
80: (4)                 def oddFooter(self):
81: (8)                     return self.HeaderFooter.oddFooter
82: (4)                 @oddFooter.setter
83: (4)                 def oddFooter(self, value):
84: (8)                     self.HeaderFooter.oddFooter = value
85: (4)                 @property
86: (4)                 def evenHeader(self):
87: (8)                     return self.HeaderFooter.evenHeader
88: (4)                 @evenHeader.setter
89: (4)                 def evenHeader(self, value):
90: (8)                     self.HeaderFooter.evenHeader = value
91: (4)                 @property
92: (4)                 def evenFooter(self):
93: (8)                     return self.HeaderFooter.evenFooter
94: (4)                 @evenFooter.setter
95: (4)                 def evenFooter(self, value):
96: (8)                     self.HeaderFooter.evenFooter = value
97: (4)                 @property
98: (4)                 def firstHeader(self):
99: (8)                     return self.HeaderFooter.firstHeader
100: (4)                @firstHeader.setter
101: (4)                def firstHeader(self, value):
102: (8)                    self.HeaderFooter.firstHeader = value
103: (4)                @property
104: (4)                def firstFooter(self):
105: (8)                    return self.HeaderFooter.firstFooter
106: (4)                @firstFooter.setter
107: (4)                def firstFooter(self, value):
108: (8)                    self.HeaderFooter.firstFooter = value
109: (4)                @property
110: (4)                def path(self):
111: (8)                    return self._path.format(self._id)
```

File 138 - views.py:

```
1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Sequence,
5: (4)                  String,
6: (4)                  Float,
7: (4)                  Integer,
8: (4)                  Bool,
9: (4)                  NoneSet,
(4)                 Set,
(0)             )
(0)             from openpyxl.descriptors.excel import (
(4)                 ExtensionList,
(4)                 Guid,
(0)             )
(0)             class BookView(Serialisable):
(4)                 tagname = "workbookView"
(4)                 visibility = NoneSet(values=(['visible', 'hidden', 'veryHidden']))
(4)                 minimized = Bool(allow_none=True)
(4)                 showHorizontalScroll = Bool(allow_none=True)
(4)                 showVerticalScroll = Bool(allow_none=True)
(4)                 showSheetTabs = Bool(allow_none=True)
(4)                 xWindow = Integer(allow_none=True)
(4)                 yWindow = Integer(allow_none=True)
(4)                 windowWidth = Integer(allow_none=True)
(4)                 windowHeight = Integer(allow_none=True)
(4)                 tabRatio = Integer(allow_none=True)
(4)                 firstSheet = Integer(allow_none=True)
(4)                 activeTab = Integer(allow_none=True)
(4)                 autoFilterDateGrouping = Bool(allow_none=True)
(4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)                 __elements__ = ()
(4)                 def __init__(self,
(17)                             visibility="visible",
(17)                             minimized=False,
(17)                             showHorizontalScroll=True,
(17)                             showVerticalScroll=True,
(17)                             showSheetTabs=True,
(17)                             xWindow=None,
(17)                             yWindow=None,
(17)                             windowWidth=None,
(17)                             windowHeight=None,
(17)                             tabRatio=600,
(17)                             firstSheet=0,
(17)                             activeTab=0,
(17)                             autoFilterDateGrouping=True,
(17)                             extLst=None,
(16)                            ):
(8)                     self.visibility = visibility
(8)                     self.minimized = minimized
(8)                     self.showHorizontalScroll = showHorizontalScroll
(8)                     self.showVerticalScroll = showVerticalScroll
(8)                     self.showSheetTabs = showSheetTabs
(8)                     self.xWindow = xWindow
(8)                     self.yWindow = yWindow
(8)                     self.windowWidth = windowWidth
(8)                     self.windowHeight = windowHeight
(8)                     self.tabRatio = tabRatio
(8)                     self.firstSheet = firstSheet
(8)                     self.activeTab = activeTab
(8)                     self.autoFilterDateGrouping = autoFilterDateGrouping
(0)             class CustomWorkbookView(Serialisable):
(4)                 tagname = "customWorkbookView"
(4)                 name = String()
(4)                 guid = Guid()
(4)                 autoUpdate = Bool(allow_none=True)
(4)                 mergeInterval = Integer(allow_none=True)
(4)                 changesSavedWin = Bool(allow_none=True)
(4)                 onlySync = Bool(allow_none=True)
(4)                 personalView = Bool(allow_none=True)
(4)                 includePrintSettings = Bool(allow_none=True)
(4)                 includeHiddenRowCol = Bool(allow_none=True)
(4)                 maximized = Bool(allow_none=True)
(4)                 minimized = Bool(allow_none=True)
(4)                 showHorizontalScroll = Bool(allow_none=True)
(4)                 showVerticalScroll = Bool(allow_none=True)
(4)                 showSheetTabs = Bool(allow_none=True)
(4)                 xWindow = Integer(allow_none=True)
```

```
79: (4)                 yWindow = Integer(allow_none=True)
80: (4)                 windowWidth = Integer()
81: (4)                 windowHeight = Integer()
82: (4)                 tabRatio = Integer(allow_none=True)
83: (4)                 activeSheetId = Integer()
84: (4)                 showFormulaBar = Bool(allow_none=True)
85: (4)                 showStatusbar = Bool(allow_none=True)
86: (4)                 showComments = NoneSet(values=(['commNone', 'commIndicator',
87: (32)                                             'commIndAndComment']))
88: (4)                 showObjects = NoneSet(values=(['all', 'placeholders']))
89: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
90: (4)                 __elements__ = ()
91: (4)                 def __init__(self,
92: (17)                             name=None,
93: (17)                             guid=None,
94: (17)                             autoUpdate=None,
95: (17)                             mergeInterval=None,
96: (17)                             changesSavedWin=None,
97: (17)                             onlySync=None,
98: (17)                             personalView=None,
99: (17)                             includePrintSettings=None,
100: (17)                            includeHiddenRowCol=None,
101: (17)                            maximized=None,
102: (17)                            minimized=None,
103: (17)                            showHorizontalScroll=None,
104: (17)                            showVerticalScroll=None,
105: (17)                            showSheetTabs=None,
106: (17)                            xWindow=None,
107: (17)                            yWindow=None,
108: (17)                            windowWidth=None,
109: (17)                            windowHeight=None,
110: (17)                            tabRatio=None,
111: (17)                            activeSheetId=None,
112: (17)                            showFormulaBar=None,
113: (17)                            showStatusbar=None,
114: (17)                            showComments="commIndicator",
115: (17)                            showObjects="all",
116: (17)                            extLst=None,
117: (16)                           ):
118: (8)                    self.name = name
119: (8)                    self.guid = guid
120: (8)                    self.autoUpdate = autoUpdate
121: (8)                    self.mergeInterval = mergeInterval
122: (8)                    self.changesSavedWin = changesSavedWin
123: (8)                    self.onlySync = onlySync
124: (8)                    self.personalView = personalView
125: (8)                    self.includePrintSettings = includePrintSettings
126: (8)                    self.includeHiddenRowCol = includeHiddenRowCol
127: (8)                    self.maximized = maximized
128: (8)                    self.minimized = minimized
129: (8)                    self.showHorizontalScroll = showHorizontalScroll
130: (8)                    self.showVerticalScroll = showVerticalScroll
131: (8)                    self.showSheetTabs = showSheetTabs
132: (8)                    self.xWindow = xWindow
133: (8)                    self.yWindow = yWindow
134: (8)                    self.windowWidth = windowWidth
135: (8)                    self.windowHeight = windowHeight
136: (8)                    self.tabRatio = tabRatio
137: (8)                    self.activeSheetId = activeSheetId
138: (8)                    self.showFormulaBar = showFormulaBar
139: (8)                    self.showStatusbar = showStatusbar
140: (8)                    self.showComments = showComments
141: (8)                    self.showObjects = showObjects

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

File 139 - _writer.py:

1: (0)              """Write the workbook global settings to the archive."""
```

```
(0)             from openpyxl.utils import quote_sheetname
(0)             from openpyxl.xml.constants import (
(4)                 ARC_APP,
(4)                 ARC_CORE,
(4)                 ARC_CUSTOM,
(4)                 ARC_WORKBOOK,
(4)                 PKG_REL_NS,
(4)                 CUSTOMUI_NS,
(4)                 ARC_ROOT_RELS,
(0)             )
(0)             from openpyxl.xml.functions import tostring, fromstring
(0)             from openpyxl.packaging.relationship import Relationship, RelationshipList
(0)             from openpyxl.workbook.defined_name import (
(4)                 DefinedName,
(4)                 DefinedNameList,
(0)             )
(0)             from openpyxl.workbook.external_reference import ExternalReference
(0)             from openpyxl.packaging.workbook import ChildSheet, WorkbookPackage, PivotCache
(0)             from openpyxl.workbook.properties import WorkbookProperties
(0)             from openpyxl.utils.datetime import CALENDAR_MAC_1904
(0)             def get_active_sheet(wb):
(4)                 """
(4)                 Return the index of the active sheet.
(4)                 If the sheet set to active is hidden return the next visible sheet or None
(4)                 """
(4)                 visible_sheets = [idx for idx, sheet in enumerate(wb._sheets) if sheet.sheet_state == "visible"]
(4)                 if not visible_sheets:
(8)                     raise IndexError("At least one sheet must be visible")
(4)                 idx = wb._active_sheet_index
(4)                 sheet = wb.active
(4)                 if sheet and sheet.sheet_state == "visible":
(8)                     return idx
(4)                 for idx in visible_sheets[idx:]:
(8)                     wb.active = idx
(8)                     return idx
(4)                 return None
(0)             class WorkbookWriter:
(4)                 def __init__(self, wb):
(8)                     self.wb = wb
(8)                     self.rels = RelationshipList()
(8)                     self.package = WorkbookPackage()
(8)                     self.package.workbookProtection = wb.security
(8)                     self.package.calcPr = wb.calculation
(4)                 def write_properties(self):
(8)                     props = WorkbookProperties() # needs a mapping to the workbook for preservation
(8)                     if self.wb.code_name is not None:
(12)                        props.codeName = self.wb.code_name
(8)                     if self.wb.excel_base_date == CALENDAR_MAC_1904:
(12)                        props.date1904 = True
(8)                     self.package.workbookPr = props
(4)                 def write_worksheets(self):
(8)                     for idx, sheet in enumerate(self.wb._sheets, 1):
(12)                        sheet_node = ChildSheet(name=sheet.title, sheetId=idx, id="rId{0}".format(idx))
(12)                        rel = Relationship(type=sheet._rel_type, Target=sheet.path)
(12)                        self.rels.append(rel)
(12)                        if not sheet.sheet_state == 'visible':
(16)                            if len(self.wb._sheets) == 1:
(20)                                raise ValueError("The only worksheet of a workbook cannot be hidden")
(16)                            sheet_node.state = sheet.sheet_state
(12)                        self.package.sheets.append(sheet_node)
(4)                 def write_refs(self):
(8)                     for link in self.wb._external_links:
(12)                        rId = len(self.wb.rels) + 1
(12)                        rel = Relationship(type=link._rel_type, Target=link.path)
```

```
66: (12)                        self.rels.append(rel)
67: (12)                        ext = ExternalReference(id=rel.id)
68: (12)                        self.package.externalReferences.append(ext)
69: (4)                 def write_names(self):
70: (8)                     defined_names = list(self.wb.defined_names.values())
71: (8)                     for idx, sheet in enumerate(self.wb.worksheets):
72: (12)                        quoted = quote_sheetname(sheet.title)
73: (12)                        if sheet.defined_names:
74: (16)                            names = sheet.defined_names.values()
75: (16)                            for n in names:
76: (20)                                n.localSheetId = idx
77: (16)                            defined_names.extend(names)
78: (12)                        if sheet.auto_filter:
79: (16)                            name = DefinedName(name='_FilterDatabase', localSheetId=idx,
hidden=True)
80: (16)                            name.value = f"{quoted}!{sheet.auto_filter}"
81: (16)                            defined_names.append(name)
82: (12)                        if sheet.print_titles:
83: (16)                            name = DefinedName(name="Print_Titles", localSheetId=idx)
84: (16)                            name.value = sheet.print_titles
85: (16)                            defined_names.append(name)
86: (12)                        if sheet.print_area:
87: (16)                            name = DefinedName(name="Print_Area", localSheetId=idx)
88: (16)                            name.value = sheet.print_area
89: (16)                            defined_names.append(name)
90: (8)                     self.package.definedNames = DefinedNameList(definedName=defined_names)
91: (4)                 def write_pivots(self):
92: (8)                     pivot_caches = set()
93: (8)                     for pivot in self.wb._pivots:
94: (12)                        if pivot.cache not in pivot_caches:
95: (16)                            pivot_caches.add(pivot.cache)
96: (16)                            c = PivotCache(cacheId=pivot.cacheId)
97: (16)                            self.package.pivotCaches.append(c)
98: (16)                            rel = Relationship(Type=pivot.cache.rel_type,
Target=pivot.cache.path)
99: (16)                            self.rels.append(rel)
100: (16)                           c.id = rel.id
101: (4)                def write_views(self):
102: (8)                    active = get_active_sheet(self.wb)
103: (8)                    if self.wb.views:
104: (12)                       self.wb.views[0].activeTab = active
105: (8)                    self.package.bookViews = self.wb.views
106: (4)                def write(self):
107: (8)                    """Write the core workbook xml."""
108: (8)                    self.write_properties()
109: (8)                    self.write_worksheets()
110: (8)                    self.write_names()
111: (8)                    self.write_pivots()
112: (8)                    self.write_views()
113: (8)                    self.write_refs()
114: (8)                    return tostring(self.package.to_tree())
115: (4)                def write_rels(self):
116: (8)                    """Write the workbook relationships xml."""
117: (8)                    styles =  Relationship(type='styles', Target='styles.xml')
118: (8)                    self.rels.append(styles)
119: (8)                    theme =  Relationship(type='theme', Target='theme/theme1.xml')
120: (8)                    self.rels.append(theme)
121: (8)                    if self.wb.vba_archive:
122: (12)                       vba =  Relationship(type='', Target='vbaProject.bin')
123: (12)                       vba.Type
='http://schemas.microsoft.com/office/2006/relationships/vbaProject'
124: (12)                       self.rels.append(vba)
125: (8)                    return tostring(self.rels.to_tree())
126: (4)                def write_root_rels(self):
127: (8)                    """Write the package relationships"""
128: (8)                    rels = RelationshipList()
129: (8)                    rel = Relationship(type="officeDocument", Target=ARC_WORKBOOK)
130: (8)                    rels.append(rel)
131: (8)                    rel = Relationship(Type=f"{PKG_REL_NS}/metadata/core-properties",
Target=ARC_CORE)
132: (8)                    rels.append(rel)
133: (8)                    rel = Relationship(type="extended-properties", Target=ARC_APP)
134: (8)                    rels.append(rel)
135: (8)                    if len(self.wb.custom_doc_props) >= 1:
136: (12)                       rel = Relationship(type="custom-properties", Target=ARC_CUSTOM)
137: (12)                       rels.append(rel)
138: (8)                    if self.wb.vba_archive is not None:
139: (12)                       xml = fromstring(self.wb.vba_archive.read(ARC_ROOT_RELS))
140: (12)                       root_rels = RelationshipList.from_tree(xml)
141: (12)                       for rel in root_rels.find(CUSTOMUI_NS):
142: (16)                           rels.append(rel)
143: (8)                    return tostring(rels.to_tree())

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

File 140 - _reader.py:

1: (0)              """Reader for a single worksheet."""
2: (0)              from copy import copy
3: (0)              from warnings import warn
4: (0)              from openpyxl.xml.functions import iterparse
5: (0)              from openpyxl.cell import Cell, MergedCell
6: (0)              from openpyxl.cell.text import Text
7: (0)              from openpyxl.worksheet.dimensions import (
8: (4)                  ColumnDimension,
9: (4)                  RowDimension,
10: (4)                 SheetFormatProperties,
11: (0)             )
12: (0)             from openpyxl.xml.constants import (
13: (4)                 SHEET_MAIN_NS,
14: (4)                 EXT_TYPES,
15: (0)             )
16: (0)             from openpyxl.formatting.formatting import ConditionalFormatting
17: (0)             from openpyxl.formula.translate import Translator
18: (0)             from openpyxl.utils import (
19: (4)                 get_column_letter,
20: (4)                 coordinate_to_tuple,
21: (4)                 )
22: (0)             from openpyxl.utils.datetime import from_excel, from_ISO8601, WINDOWS_EPOCH
23: (0)             from openpyxl.descriptors.excel import ExtensionList
24: (0)             from openpyxl.cell.rich_text import CellRichText
25: (0)             from .formula import DataTableFormula, ArrayFormula
26: (0)             from .filters import AutoFilter
27: (0)             from .header_footer import HeaderFooter
28: (0)             from .hyperlink import HyperlinkList
29: (0)             from .merge import MergeCells
30: (0)             from .page import PageMargins, PrintOptions, PrintPageSetup
31: (0)             from .pagebreak import RowBreak, ColBreak
32: (0)             from .protection import SheetProtection
33: (0)             from .scenario import ScenarioList
34: (0)             from .views import SheetViewList
35: (0)             from .datavalidation import DataValidationList
36: (0)             from .table import TablePartList
37: (0)             from .properties import WorksheetProperties
38: (0)             from .dimensions import SheetDimension
39: (0)             from .related import Related
40: (0)             CELL_TAG = '{%s}c' % SHEET_MAIN_NS
41: (0)             VALUE_TAG = '{%s}v' % SHEET_MAIN_NS
42: (0)             FORMULA_TAG = '{%s}f' % SHEET_MAIN_NS
43: (0)             MERGE_TAG = '{%s}mergeCells' % SHEET_MAIN_NS
44: (0)             INLINE_STRING = "{%s}is" % SHEET_MAIN_NS
45: (0)             COL_TAG = '{%s}col' % SHEET_MAIN_NS
46: (0)             ROW_TAG = '{%s}row' % SHEET_MAIN_NS
47: (0)             CF_TAG = '{%s}conditionalFormatting' % SHEET_MAIN_NS
48: (0)             LEGACY_TAG = '{%s}legacyDrawing' % SHEET_MAIN_NS
49: (0)             PROT_TAG = '{%s}sheetProtection' % SHEET_MAIN_NS
50: (0)             EXT_TAG = "{%s}extLst" % SHEET_MAIN_NS
51: (0)             HYPERLINK_TAG = "{%s}hyperlinks" % SHEET_MAIN_NS
```

```
  52: (0)              TABLE_TAG = "{%s}tableParts" % SHEET_MAIN_NS
  53: (0)              PRINT_TAG = '{%s}printOptions' % SHEET_MAIN_NS
  54: (0)              MARGINS_TAG = '{%s}pageMargins' % SHEET_MAIN_NS
  55: (0)              PAGE_TAG = '{%s}pageSetup' % SHEET_MAIN_NS
  56: (0)              HEADER_TAG = '{%s}headerFooter' % SHEET_MAIN_NS
  57: (0)              FILTER_TAG = '{%s}autoFilter' % SHEET_MAIN_NS
  58: (0)              VALIDATION_TAG = '{%s}dataValidations' % SHEET_MAIN_NS
  59: (0)              PROPERTIES_TAG = '{%s}sheetPr' % SHEET_MAIN_NS
  60: (0)              VIEWS_TAG = '{%s}sheetViews' % SHEET_MAIN_NS
  61: (0)              FORMAT_TAG = '{%s}sheetFormatPr' % SHEET_MAIN_NS
  62: (0)              ROW_BREAK_TAG = '{%s}rowBreaks' % SHEET_MAIN_NS
  63: (0)              COL_BREAK_TAG = '{%s}colBreaks' % SHEET_MAIN_NS
  64: (0)              SCENARIOS_TAG = '{%s}scenarios' % SHEET_MAIN_NS
  65: (0)              DATA_TAG = '{%s}sheetData' % SHEET_MAIN_NS
  66: (0)              DIMENSION_TAG = '{%s}dimension' % SHEET_MAIN_NS
  67: (0)              CUSTOM_VIEWS_TAG = '{%s}customSheetViews' % SHEET_MAIN_NS
  68: (0)              def _cast_number(value):
  69: (4)                  "Convert numbers as string to an int or float"
  70: (4)                  if "." in value or "E" in value or "e" in value:
  71: (8)                      return float(value)
  72: (4)                  return int(value)
  73: (0)              def parse_richtext_string(element):
  74: (4)                  """
  75: (4)                  Parse inline string and preserve rich text formatting
  76: (4)                  """
  77: (4)                  value = CellRichText.from_tree(element) or ""
  78: (4)                  if len(value) == 1 and isinstance(value[0], str):
  79: (8)                      value = value[0]
  80: (4)                  return value
  81: (0)              class WorkSheetParser:
  82: (4)                  def __init__(self, src, shared_strings, data_only=False,
  83: (17)                              epoch=WINDOWS_EPOCH, date_formats=set(),
  84: (17)                              timedelta_formats=set(), rich_text=False):
  85: (8)                      self.min_row = self.min_col = None
  86: (8)                      self.epoch = epoch
  87: (8)                      self.source = src
  88: (8)                      self.shared_strings = shared_strings
  89: (8)                      self.data_only = data_only
  90: (8)                      self.shared_formulae = {}
  91: (8)                      self.row_counter = self.col_counter = 0
  92: (8)                      self.tables = TablePartList()
  93: (8)                      self.date_formats = date_formats
  94: (8)                      self.timedelta_formats = timedelta_formats
  95: (8)                      self.row_dimensions = {}
  96: (8)                      self.column_dimensions = {}
  97: (8)                      self.number_formats = []
  98: (8)                      self.keep_vba = False
  99: (8)                      self.hyperlinks = HyperlinkList()
  100: (8)                     self.formatting = []
  101: (8)                     self.legacy_drawing = None
  102: (8)                     self.merged_cells = None
  103: (8)                     self.row_breaks = RowBreak()
  104: (8)                     self.col_breaks = ColBreak()
  105: (8)                     self.rich_text = rich_text
  106: (4)                 def parse(self):
  107: (8)                     dispatcher = {
  108: (12)                        COL_TAG: self.parse_column_dimensions,
  109: (12)                        PROT_TAG: self.parse_sheet_protection,
  110: (12)                        EXT_TAG: self.parse_extensions,
  111: (12)                        CF_TAG: self.parse_formatting,
  112: (12)                        LEGACY_TAG: self.parse_legacy,
  113: (12)                        ROW_BREAK_TAG: self.parse_row_breaks,
  114: (12)                        COL_BREAK_TAG: self.parse_col_breaks,
  115: (12)                        CUSTOM_VIEWS_TAG: self.parse_custom_views,
  116: (22)                                  }
  117: (8)                     properties = {
  118: (12)                        PRINT_TAG: ('print_options', PrintOptions),
  119: (12)                        MARGINS_TAG: ('page_margins', PageMargins),
  120: (12)                        PAGE_TAG: ('page_setup', PrintPageSetup),
  121: (12)                       HEADER_TAG: ('HeaderFooter', HeaderFooter),
  122: (12)                       FILTER_TAG: ('auto_filter', AutoFilter),
  123: (12)                       VALIDATION_TAG: ('data_validations', DataValidationList),
  124: (12)                       PROPERTIES_TAG: ('sheet_properties', WorksheetProperties),
  125: (12)                       VIEWS_TAG: ('views', SheetViewList),
  126: (12)                       FORMAT_TAG: ('sheet_format', SheetFormatProperties),
  127: (12)                       SCENARIOS_TAG: ('scenarios', ScenarioList),
  128: (12)                       TABLE_TAG: ('tables', TablePartList),
  129: (12)                       HYPERLINK_TAG: ('hyperlinks', HyperlinkList),
  130: (12)                       MERGE_TAG: ('merged_cells', MergeCells),
  131: (8)                    }
  132: (8)                    it = iterparse(self.source) # add a finaliser to close the source when
  this becomes possible
  133: (8)                    for _, element in it:
  134: (12)                       tag_name = element.tag
  135: (12)                       if tag_name in dispatcher:
  136: (16)                           dispatcher[tag_name](element)
  137: (16)                           element.clear()
  138: (12)                       elif tag_name in properties:
  139: (16)                           prop = properties[tag_name]
  140: (16)                           obj = prop[1].from_tree(element)
  141: (16)                           setattr(self, prop[0], obj)
  142: (16)                           element.clear()
  143: (12)                       elif tag_name == ROW_TAG:
  144: (16)                           row = self.parse_row(element)
  145: (16)                           element.clear()
  146: (16)                           yield row
  147: (4)                def parse_dimensions(self):
  148: (8)                    """
  149: (8)                    Get worksheet dimensions if they are provided.
  150: (8)                    """
  151: (8)                    it = iterparse(self.source)
  152: (8)                    for _event, element in it:
  153: (12)                       if element.tag == DIMENSION_TAG:
  154: (16)                           dim = SheetDimension.from_tree(element)
  155: (16)                           return dim.boundaries
  156: (12)                       elif element.tag == DATA_TAG:
  157: (16)                           break
  158: (12)                       element.clear()
  159: (4)                def parse_cell(self, element):
  160: (8)                    data_type = element.get('t', 'n')
  161: (8)                    coordinate = element.get('r')
  162: (8)                    style_id = element.get('s', 0)
  163: (8)                    if style_id:
  164: (12)                       style_id = int(style_id)
  165: (8)                    if data_type == "inlineStr":
  166: (12)                       value = None
  167: (8)                    else:
  168: (12)                       value = element.findtext(VALUE_TAG, None) or None
  169: (8)                    if coordinate:
  170: (12)                       row, column = coordinate_to_tuple(coordinate)
  171: (12)                       self.col_counter = column
  172: (8)                    else:
  173: (12)                       self.col_counter += 1
  174: (12)                       row, column = self.row_counter, self.col_counter
  175: (8)                    if not self.data_only and element.find(FORMULA_TAG) is not None:
  176: (12)                       data_type = 'f'
  177: (12)                       value = self.parse_formula(element)
  178: (8)                    elif value is not None:
  179: (12)                       if data_type == 'n':
  180: (16)                           value = _cast_number(value)
  181: (16)                           if style_id in self.date_formats:
  182: (20)                               data_type = 'd'
  183: (20)                               try:
  184: (24)                                   value = from_excel(
  185: (28)                                       value, self.epoch, timedelta=style_id in
  self.timedelta_formats
  186: (24)                                   )
  187: (20)                               except (OverflowError, ValueError):
```

```
(24)                        msg = f"""Cell {coordinate} is marked as a date but the serial value {value} is outside the limits for dates. The cell will be treated as an error."""
(24)                        warn(msg)
(24)                        data_type = "e"
(24)                        value = "#VALUE!"
(12)            elif data_type == 's':
(16)                value = self.shared_strings[int(value)]
(12)            elif data_type == 'b':
(16)                value = bool(int(value))
(12)            elif data_type == "str":
(16)                data_type = "s"
(12)            elif data_type == 'd':
(16)                value = from_ISO8601(value)
(8)         elif data_type == 'inlineStr':
(16)                child = element.find(INLINE_STRING)
(16)                if child is not None:
(20)                    data_type = 's'
(20)                    if self.rich_text:
(24)                        value = parse_richtext_string(child)
(20)                    else:
(24)                        value = Text.from_tree(child).content
(8)         return {'row':row, 'column':column, 'value':value, 'data_type':data_type, 'style_id':style_id}
(4)     def parse_formula(self, element):
(8)         """
(8)         possible formulae types: shared, array, datatable
(8)         """
(8)         formula = element.find(FORMULA_TAG)
(8)         formula_type = formula.get('t')
(8)         coordinate = element.get('r')
(8)         value = "="
(8)         if formula.text is not None:
(12)            value += formula.text
(8)         if formula_type == "array":
(12)            value = ArrayFormula(ref=formula.get('ref'), text=value)
(8)         elif formula_type == "shared":
(12)            idx = formula.get('si')
(12)            if idx in self.shared_formulae:
(16)                trans = self.shared_formulae[idx]
(16)                value = trans.translate_formula(coordinate)
(12)            elif value != "=":
(16)                self.shared_formulae[idx] = Translator(value, coordinate)
(8)         elif formula_type == "dataTable":
(12)            value = DataTableFormula(**formula.attrib)
(8)         return value
(4)     def parse_column_dimensions(self, col):
(8)         attrs = dict(col.attrib)
(8)         column = get_column_letter(int(attrs['min']))
(8)         attrs['index'] = column
(8)         self.column_dimensions[column] = attrs
(4)     def parse_row(self, row):
(8)         attrs = dict(row.attrib)
(8)         if "r" in attrs:
(12)            try:
(16)                self.row_counter = int(attrs['r'])
(12)            except ValueError:
(16)                val = float(attrs['r'])
(16)                if val.is_integer():
(20)                    self.row_counter = int(val)
(16)                else:
(20)                    raise ValueError(f"{attrs['r']} is not a valid row number")
(8)         else:
(12)            self.row_counter += 1
(8)         self.col_counter = 0
(8)         keys = {k for k in attrs if not k.startswith('{')}
(8)         if keys - {'r', 'spans'}:
(12)            self.row_dimensions[str(self.row_counter)] = attrs
(8)         cells = [self.parse_cell(el) for el in row]
```

```
254: (8)                        return self.row_counter, cells
255: (4)                    def parse_formatting(self, element):
256: (8)                        try:
257: (12)                           cf = ConditionalFormatting.from_tree(element)
258: (12)                           self.formatting.append(cf)
259: (8)                        except TypeError as e:
260: (12)                           msg = f"Failed to load a conditional formatting rule. It will be
discarded. Cause: {e}"
261: (12)                           warn(msg)
262: (4)                    def parse_sheet_protection(self, element):
263: (8)                        protection = SheetProtection.from_tree(element)
264: (8)                        password = element.get("password")
265: (8)                        if password is not None:
266: (12)                           protection.set_password(password, True)
267: (8)                        self.protection = protection
268: (4)                    def parse_extensions(self, element):
269: (8)                        extLst = ExtensionList.from_tree(element)
270: (8)                        for e in extLst.ext:
271: (12)                           ext_type = EXT_TYPES.get(e.uri.upper(), "Unknown")
272: (12)                           msg = "{0} extension is not supported and will be
removed".format(ext_type)
273: (12)                           warn(msg)
274: (4)                    def parse_legacy(self, element):
275: (8)                        obj = Related.from_tree(element)
276: (8)                        self.legacy_drawing = obj.id
277: (4)                    def parse_row_breaks(self, element):
278: (8)                        brk = RowBreak.from_tree(element)
279: (8)                        self.row_breaks = brk
280: (4)                    def parse_col_breaks(self, element):
281: (8)                        brk = ColBreak.from_tree(element)
282: (8)                        self.col_breaks = brk
283: (4)                    def parse_custom_views(self, element):
284: (8)                        self.row_breaks = RowBreak()
285: (8)                        self.col_breaks = ColBreak()
286: (0)                class WorksheetReader:
287: (4)                    """
288: (4)                    Create a parser and apply it to a workbook
289: (4)                    """
290: (4)                    def __init__(self, ws, xml_source, shared_strings, data_only, rich_text):
291: (8)                        self.ws = ws
292: (8)                        self.parser = WorkSheetParser(xml_source, shared_strings,
293: (16)                               data_only, ws.parent.epoch, ws.parent._date_formats,
294: (16)                               ws.parent._timedelta_formats, rich_text)
295: (8)                        self.tables = []
296: (4)                    def bind_cells(self):
297: (8)                        for idx, row in self.parser.parse():
298: (12)                           for cell in row:
299: (16)                               style = self.ws.parent._cell_styles[cell['style_id']]
300: (16)                               c = Cell(self.ws, row=cell['row'], column=cell['column'],
style_array=style)
301: (16)                               c._value = cell['value']
302: (16)                               c.data_type = cell['data_type']
303: (16)                               self.ws._cells[(cell['row'], cell['column'])] = c
304: (8)                        if self.ws._cells:
305: (12)                           self.ws._current_row = self.ws.max_row # use cells not row
dimensions
306: (4)                    def bind_formatting(self):
307: (8)                        for cf in self.parser.formatting:
308: (12)                           for rule in cf.rules:
309: (16)                               if rule.dxfId is not None:
310: (20)                                   rule.dxf = self.ws.parent._differential_styles[rule.dxfId]
311: (16)                               self.ws.conditional_formatting[cf] = rule
312: (4)                    def bind_tables(self):
313: (8)                        for t in self.parser.tables.tablePart:
314: (12)                           rel = self.ws._rels.get(t.id)
315: (12)                           self.tables.append(rel.Target)
316: (4)                    def bind_merged_cells(self):
317: (8)                        from openpyxl.worksheet.cell_range import MultiCellRange
318: (8)                        from openpyxl.worksheet.merge import MergedCellRange
```

```
319: (8)                    if not self.parser.merged_cells:
320: (12)                       return
321: (8)                    ranges = []
322: (8)                    for cr in self.parser.merged_cells.mergeCell:
323: (12)                       mcr = MergedCellRange(self.ws, cr.ref)
324: (12)                       self.ws._clean_merge_range(mcr)
325: (12)                       ranges.append(mcr)
326: (8)                    self.ws.merged_cells = MultiCellRange(ranges)
327: (4)                def bind_hyperlinks(self):
328: (8)                    for link in self.parser.hyperlinks.hyperlink:
329: (12)                       if link.id:
330: (16)                           rel = self.ws._rels.get(link.id)
331: (16)                           link.target = rel.Target
332: (12)                       if ":" in link.ref:
333: (16)                           for row in self.ws[link.ref]:
334: (20)                               for cell in row:
335: (24)                                   try:
336: (28)                                       cell.hyperlink = copy(link)
337: (24)                                   except AttributeError:
338: (28)                                       pass
339: (12)                       else:
340: (16)                           cell = self.ws[link.ref]
341: (16)                           if isinstance(cell, MergedCell):
342: (20)                               cell = self.normalize_merged_cell_link(cell.coordinate)
343: (16)                           cell.hyperlink = link
344: (4)                def normalize_merged_cell_link(self, coord):
345: (8)                    """
346: (8)                    Returns the appropriate cell to which a hyperlink, which references a
merged cell at the specified coordinates,
347: (8)                    should be bound.
348: (8)                    """
349: (8)                    for rng in self.ws.merged_cells:
350: (12)                       if coord in rng:
351: (16)                           return self.ws.cell(*rng.top[0])
352: (4)                def bind_col_dimensions(self):
353: (8)                    for col, cd in self.parser.column_dimensions.items():
354: (12)                       if 'style' in cd:
355: (16)                           key = int(cd['style'])
356: (16)                           cd['style'] = self.ws.parent._cell_styles[key]
357: (12)                       self.ws.column_dimensions[col] = ColumnDimension(self.ws, **cd)
358: (4)                def bind_row_dimensions(self):
359: (8)                    for row, rd in self.parser.row_dimensions.items():
360: (12)                       if 's' in rd:
361: (16)                           key = int(rd['s'])
362: (16)                           rd['s'] = self.ws.parent._cell_styles[key]
363: (12)                       self.ws.row_dimensions[int(row)] = RowDimension(self.ws, **rd)
364: (4)                def bind_properties(self):
365: (8)                    for k in ('print_options', 'page_margins', 'page_setup',
366: (18)                              'HeaderFooter', 'auto_filter', 'data_validations',
367: (18)                              'sheet_properties', 'views', 'sheet_format',
368: (18)                              'row_breaks', 'col_breaks', 'scenarios', 'legacy_drawing',
369: (18)                              'protection',
370: (18)                              ):
371: (12)                       v = getattr(self.parser, k, None)
372: (12)                       if v is not None:
373: (16)                           setattr(self.ws, k, v)
374: (4)                def bind_all(self):
375: (8)                    self.bind_cells()
376: (8)                    self.bind_merged_cells()
377: (8)                    self.bind_hyperlinks()
378: (8)                    self.bind_formatting()
379: (8)                    self.bind_col_dimensions()
380: (8)                    self.bind_row_dimensions()
381: (8)                    self.bind_tables()
382: (8)                    self.bind_properties()

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

File 141 - workbook.py:
```

```
(0)     """Workbook is the top-level container for all document information."""
(0)     from copy import copy
(0)     from openpyxl.compat import deprecated
(0)     from openpyxl.worksheet.worksheet import Worksheet
(0)     from openpyxl.worksheet._read_only import ReadOnlyWorksheet
(0)     from openpyxl.worksheet._write_only import WriteOnlyWorksheet
(0)     from openpyxl.worksheet.copier import WorksheetCopy
(0)     from openpyxl.utils import quote_sheetname
(0)     from openpyxl.utils.indexed_list import IndexedList
(0)     from openpyxl.utils.datetime  import WINDOWS_EPOCH, MAC_EPOCH
(0)     from openpyxl.utils.exceptions import ReadOnlyWorkbookException
(0)     from openpyxl.writer.excel import save_workbook
(0)     from openpyxl.styles.cell_style import StyleArray
(0)     from openpyxl.styles.named_styles import NamedStyle
(0)     from openpyxl.styles.differential import DifferentialStyleList
(0)     from openpyxl.styles.alignment import Alignment
(0)     from openpyxl.styles.borders import DEFAULT_BORDER
(0)     from openpyxl.styles.fills import DEFAULT_EMPTY_FILL, DEFAULT_GRAY_FILL
(0)     from openpyxl.styles.fonts import DEFAULT_FONT
(0)     from openpyxl.styles.protection import Protection
(0)     from openpyxl.styles.colors import COLOR_INDEX
(0)     from openpyxl.styles.named_styles import NamedStyleList
(0)     from openpyxl.styles.table import TableStyleList
(0)     from openpyxl.chartsheet import Chartsheet
(0)     from .defined_name import DefinedName, DefinedNameDict
(0)     from openpyxl.packaging.core import DocumentProperties
(0)     from openpyxl.packaging.custom import CustomPropertyList
(0)     from openpyxl.packaging.relationship import RelationshipList
(0)     from .child import _WorkbookChild
(0)     from .protection import DocumentSecurity
(0)     from .properties import CalcProperties
(0)     from .views import BookView
(0)     from openpyxl.xml.constants import (
(4)         XLSM,
(4)         XLSX,
(4)         XLTM,
(4)         XLTX
(0)     )
(0)     INTEGER_TYPES = (int,)
(0)     class Workbook:
(4)         """Workbook is the container for all other parts of the document."""
(4)         _read_only = False
(4)         _data_only = False
(4)         template = False
(4)         path = "/xl/workbook.xml"
(4)         def __init__(self,
(17)                     write_only=False,
(17)                     iso_dates=False,
(17)                     ):
(8)             self._sheets = []
(8)             self._pivots = []
(8)             self._active_sheet_index = 0
(8)             self.defined_names = DefinedNameDict()
(8)             self._external_links = []
(8)             self.properties = DocumentProperties()
(8)             self.custom_doc_props = CustomPropertyList()
(8)             self.security = DocumentSecurity()
(8)             self.__write_only = write_only
(8)             self.shared_strings = IndexedList()
(8)             self._setup_styles()
(8)             self.loaded_theme = None
(8)             self.vba_archive = None
(8)             self.is_template = False
(8)             self.code_name = None
(8)             self.epoch = WINDOWS_EPOCH
(8)             self.encoding = "utf-8"
(8)             self.iso_dates = iso_dates
(8)             if not self.write_only:
```

```
69: (12)                        self._sheets.append(Worksheet(self))
70: (8)                     self.rels = RelationshipList()
71: (8)                     self.calculation = CalcProperties()
72: (8)                     self.views = [BookView()]
73: (4)                 def _setup_styles(self):
74: (8)                     """Bootstrap styles"""
75: (8)                     self._fonts = IndexedList()
76: (8)                     self._fonts.add(DEFAULT_FONT)
77: (8)                     self._alignments = IndexedList([Alignment()])
78: (8)                     self._borders = IndexedList()
79: (8)                     self._borders.add(DEFAULT_BORDER)
80: (8)                     self._fills = IndexedList()
81: (8)                     self._fills.add(DEFAULT_EMPTY_FILL)
82: (8)                     self._fills.add(DEFAULT_GRAY_FILL)
83: (8)                     self._number_formats = IndexedList()
84: (8)                     self._date_formats = {}
85: (8)                     self._timedelta_formats = {}
86: (8)                     self._protections = IndexedList([Protection()])
87: (8)                     self._colors = COLOR_INDEX
88: (8)                     self._cell_styles = IndexedList([StyleArray()])
89: (8)                     self._named_styles = NamedStyleList()
90: (8)                     self.add_named_style(NamedStyle(font=copy(DEFAULT_FONT),
border=copy(DEFAULT_BORDER), builtinId=0))
91: (8)                     self._table_styles = TableStyleList()
92: (8)                     self._differential_styles = DifferentialStyleList()
93: (4)                 @property
94: (4)                 def epoch(self):
95: (8)                     if self._epoch == WINDOWS_EPOCH:
96: (12)                        return WINDOWS_EPOCH
97: (8)                     return MAC_EPOCH
98: (4)                 @epoch.setter
99: (4)                 def epoch(self, value):
100: (8)                    if value not in (WINDOWS_EPOCH, MAC_EPOCH):
101: (12)                       raise ValueError("The epoch must be either 1900 or 1904")
102: (8)                    self._epoch = value
103: (4)                @property
104: (4)                def read_only(self):
105: (8)                    return self._read_only
106: (4)                @property
107: (4)                def data_only(self):
108: (8)                    return self._data_only
109: (4)                @property
110: (4)                def write_only(self):
111: (8)                    return self.__write_only
112: (4)                @property
113: (4)                def excel_base_date(self):
114: (8)                    return self.epoch
115: (4)                @property
116: (4)                def active(self):
117: (8)                    """Get the currently active sheet or None
118: (8)                    :type: :class:`openpyxl.worksheet.worksheet.Worksheet`
119: (8)                    """
120: (8)                    try:
121: (12)                       return self._sheets[self._active_sheet_index]
122: (8)                    except IndexError:
123: (12)                       pass
124: (4)                @active.setter
125: (4)                def active(self, value):
126: (8)                    """Set the active sheet"""
127: (8)                    if not isinstance(value, (_WorkbookChild, INTEGER_TYPES)):
128: (12)                       raise TypeError("Value must be either a worksheet, chartsheet or
numerical index")
129: (8)                    if isinstance(value, INTEGER_TYPES):
130: (12)                       self._active_sheet_index = value
131: (12)                       return
132: (8)                    if value not in self._sheets:
133: (12)                       raise ValueError("Worksheet is not in the workbook")
134: (8)                    if value.sheet_state != "visible":
135: (12)                       raise ValueError("Only visible sheets can be made active")
```

```
136: (8)                    idx = self._sheets.index(value)
137: (8)                    self._active_sheet_index = idx
138: (4)                def create_sheet(self, title=None, index=None):
139: (8)                    """Create a worksheet (at an optional index).
140: (8)                    :param title: optional title of the sheet
141: (8)                    :type title: str
142: (8)                    :param index: optional position at which the sheet will be inserted
143: (8)                    :type index: int
144: (8)                    """
145: (8)                    if self.read_only:
146: (12)                       raise ReadOnlyWorkbookException('Cannot create new sheet in a
read-only workbook')
147: (8)                    if self.write_only :
148: (12)                       new_ws = WriteOnlyWorksheet(parent=self, title=title)
149: (8)                    else:
150: (12)                       new_ws = Worksheet(parent=self, title=title)
151: (8)                    self._add_sheet(sheet=new_ws, index=index)
152: (8)                    return new_ws
153: (4)                def _add_sheet(self, sheet, index=None):
154: (8)                    """Add an worksheet (at an optional index)."""
155: (8)                    if not isinstance(sheet, (Worksheet, WriteOnlyWorksheet, Chartsheet)):
156: (12)                       raise TypeError("Cannot be added to a workbook")
157: (8)                    if sheet.parent != self:
158: (12)                       raise ValueError("You cannot add worksheets from another
workbook.")
159: (8)                    if index is None:
160: (12)                       self._sheets.append(sheet)
161: (8)                    else:
162: (12)                       self._sheets.insert(index, sheet)
163: (4)                def move_sheet(self, sheet, offset=0):
164: (8)                    """
165: (8)                    Move a sheet or sheetname
166: (8)                    """
167: (8)                    if not isinstance(sheet, Worksheet):
168: (12)                       sheet = self[sheet]
169: (8)                    idx = self._sheets.index(sheet)
170: (8)                    del self._sheets[idx]
171: (8)                    new_pos = idx + offset
172: (8)                    self._sheets.insert(new_pos, sheet)
173: (4)                def remove(self, worksheet):
174: (8)                    """Remove `worksheet` from this workbook."""
175: (8)                    idx = self._sheets.index(worksheet)
176: (8)                    self._sheets.remove(worksheet)
177: (4)                @deprecated("Use wb.remove(worksheet) or del wb[sheetname]")
178: (4)                def remove_sheet(self, worksheet):
179: (8)                    """Remove `worksheet` from this workbook."""
180: (8)                    self.remove(worksheet)
181: (4)                def create_chartsheet(self, title=None, index=None):
182: (8)                    if self.read_only:
183: (12)                       raise ReadOnlyWorkbookException("Cannot create new sheet in a
read-only workbook")
184: (8)                    cs = Chartsheet(parent=self, title=title)
185: (8)                    self._add_sheet(cs, index)
186: (8)                    return cs
187: (4)                @deprecated("Use wb[sheetname]")
188: (4)                def get_sheet_by_name(self, name):
189: (8)                    """Returns a worksheet by its name.
190: (8)                    :param name: the name of the worksheet to look for
191: (8)                    :type name: string
192: (8)                    """
193: (8)                    return self[name]
194: (4)                def __contains__(self, key):
195: (8)                    return key in self.sheetnames
196: (4)                def index(self, worksheet):
197: (8)                    """Return the index of a worksheet."""
198: (8)                    return self.worksheets.index(worksheet)
199: (4)                @deprecated("Use wb.index(worksheet)")
200: (4)                def get_index(self, worksheet):
201: (8)                    """Return the index of the worksheet."""
```

```
202: (8)                    return self.index(worksheet)
203: (4)                def __getitem__(self, key):
204: (8)                    """Returns a worksheet by its name.
205: (8)                    :param name: the name of the worksheet to look for
206: (8)                    :type name: string
207: (8)                    """
208: (8)                    for sheet in self.worksheets + self.chartsheets:
209: (12)                       if sheet.title == key:
210: (16)                           return sheet
211: (8)                    raise KeyError("Worksheet {0} does not exist.".format(key))
212: (4)                def __delitem__(self, key):
213: (8)                    sheet = self[key]
214: (8)                    self.remove(sheet)
215: (4)                def __iter__(self):
216: (8)                    return iter(self.worksheets)
217: (4)                @deprecated("Use wb.sheetnames")
218: (4)                def get_sheet_names(self):
219: (8)                    return self.sheetnames
220: (4)                @property
221: (4)                def worksheets(self):
222: (8)                    """A list of sheets in this workbook
223: (8)                    :type: list of :class:`openpyxl.worksheet.worksheet.Worksheet`
224: (8)                    """
225: (8)                    return [s for s in self._sheets if isinstance(s, (Worksheet, ReadOnlyWorksheet, WriteOnlyWorksheet))]
226: (4)                @property
227: (4)                def chartsheets(self):
228: (8)                    """A list of Chartsheets in this workbook
229: (8)                    :type: list of :class:`openpyxl.chartsheet.chartsheet.Chartsheet`
230: (8)                    """
231: (8)                    return [s for s in self._sheets if isinstance(s, Chartsheet)]
232: (4)                @property
233: (4)                def sheetnames(self):
234: (8)                    """Returns the list of the names of worksheets in this workbook.
235: (8)                    Names are returned in the worksheets order.
236: (8)                    :type: list of strings
237: (8)                    """
238: (8)                    return [s.title for s in self._sheets]
239: (4)                @deprecated("Assign scoped named ranges directly to worksheets or global ones to the workbook. Deprecated in 3.1")
240: (4)                def create_named_range(self, name, worksheet=None, value=None, scope=None):
241: (8)                    """Create a new named_range on a worksheet
242: (8)                    """
243: (8)                    defn = DefinedName(name=name)
244: (8)                    if worksheet is not None:
245: (12)                       defn.value = "{0}!{1}".format(quote_sheetname(worksheet.title), value)
246: (8)                    else:
247: (12)                       defn.value = value
248: (8)                    self.defined_names[name] = defn
249: (4)                def add_named_style(self, style):
250: (8)                    """
251: (8)                    Add a named style
252: (8)                    """
253: (8)                    self._named_styles.append(style)
254: (8)                    style.bind(self)
255: (4)                @property
256: (4)                def named_styles(self):
257: (8)                    """
258: (8)                    List available named styles
259: (8)                    """
260: (8)                    return self._named_styles.names
261: (4)                @property
262: (4)                def mime_type(self):
263: (8)                    """
264: (8)                    The mime type is determined by whether a workbook is a template or
265: (8)                    not and whether it contains macros or not. Excel requires the file
266: (8)                    extension to match but openpyxl does not enforce this.
```

```
267: (8)                        """
268: (8)                        ct = self.template and XLTX or XLSX
269: (8)                        if self.vba_archive:
270: (12)                           ct = self.template and XLTM or XLSM
271: (8)                        return ct
272: (4)                    def save(self, filename):
273: (8)                        """Save the current workbook under the given `filename`.
274: (8)                        Use this function instead of using an `ExcelWriter`.
275: (8)                        .. warning::
276: (12)                           When creating your workbook using `write_only` set to True,
277: (12)                           you will only be able to call this function once. Subsequent
attempts to
278: (12)                           modify or save the file will raise an
:class:`openpyxl.shared.exc.WorkbookAlreadySaved` exception.
279: (8)                        """
280: (8)                        if self.read_only:
281: (12)                           raise TypeError("""Workbook is read-only""")
282: (8)                        if self.write_only and not self.worksheets:
283: (12)                           self.create_sheet()
284: (8)                        save_workbook(self, filename)
285: (4)                    @property
286: (4)                    def style_names(self):
287: (8)                        """
288: (8)                        List of named styles
289: (8)                        """
290: (8)                        return [s.name for s in self._named_styles]
291: (4)                    def copy_worksheet(self, from_worksheet):
292: (8)                        """Copy an existing worksheet in the current workbook
293: (8)                        .. warning::
294: (12)                           This function cannot copy worksheets between workbooks.
295: (12)                           worksheets can only be copied within the workbook that they belong
296: (8)                        :param from_worksheet: the worksheet to be copied from
297: (8)                        :return: copy of the initial worksheet
298: (8)                        """
299: (8)                        if self.__write_only or self._read_only:
300: (12)                           raise ValueError("Cannot copy worksheets in read-only or write-
only mode")
301: (8)                        new_title = u"{0} Copy".format(from_worksheet.title)
302: (8)                        to_worksheet = self.create_sheet(title=new_title)
303: (8)                        cp = WorksheetCopy(source_worksheet=from_worksheet,
target_worksheet=to_worksheet)
304: (8)                        cp.copy_worksheet()
305: (8)                        return to_worksheet
306: (4)                    def close(self):
307: (8)                        """
308: (8)                        Close workbook file if open. Only affects read-only and write-only
modes.
309: (8)                        """
310: (8)                        if hasattr(self, '_archive'):
311: (12)                           self._archive.close()
312: (4)                    def _duplicate_name(self, name):
313: (8)                        """
314: (8)                        Check for duplicate name in defined name list and table list of each
worksheet.
315: (8)                        Names are not case sensitive.
316: (8)                        """
317: (8)                        name = name.lower()
318: (8)                        for sheet in self.worksheets:
319: (12)                           for t in sheet.tables:
320: (16)                               if name == t.lower():
321: (20)                                   return True
322: (8)                        if name in self.defined_names:
323: (12)                           return True

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

File 142 - __init__.py:

1: (0)              from .workbook import Workbook
```

```
----------------------------------------

File 143 - external.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  String,
5: (4)                  Bool,
6: (4)                  Integer,
7: (4)                  NoneSet,
8: (4)                  Sequence,
9: (0)              )
10: (0)             from openpyxl.descriptors.excel import Relation
11: (0)             from openpyxl.descriptors.nested import NestedText
12: (0)             from openpyxl.descriptors.sequence import NestedSequence, ValueSequence
13: (0)             from openpyxl.packaging.relationship import (
14: (4)                 Relationship,
15: (4)                 get_rels_path,
16: (4)                 get_dependents
17: (4)                 )
18: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS
19: (0)             from openpyxl.xml.functions import fromstring
20: (0)             """Manage links to external Workbooks"""
21: (0)             class ExternalCell(Serialisable):
22: (4)                 r = String()
23: (4)                 t = NoneSet(values=(['b', 'd', 'n', 'e', 's', 'str', 'inlineStr']))
24: (4)                 vm = Integer(allow_none=True)
25: (4)                 v = NestedText(allow_none=True, expected_type=str)
26: (4)                 def __init__(self,
27: (17)                             r=None,
28: (17)                             t=None,
29: (17)                             vm=None,
30: (17)                             v=None,
31: (16)                            ):
32: (8)                     self.r = r
33: (8)                     self.t = t
34: (8)                     self.vm = vm
35: (8)                     self.v = v
36: (0)             class ExternalRow(Serialisable):
37: (4)                 r = Integer()
38: (4)                 cell = Sequence(expected_type=ExternalCell)
39: (4)                 __elements__ = ('cell',)
40: (4)                 def __init__(self,
41: (17)                             r=(),
42: (17)                             cell=None,
43: (16)                            ):
44: (8)                     self.r = r
45: (8)                     self.cell = cell
46: (0)             class ExternalSheetData(Serialisable):
47: (4)                 sheetId = Integer()
48: (4)                 refreshError = Bool(allow_none=True)
49: (4)                 row = Sequence(expected_type=ExternalRow)
50: (4)                 __elements__ = ('row',)
51: (4)                 def __init__(self,
52: (17)                             sheetId=None,
53: (17)                             refreshError=None,
54: (17)                             row=(),
55: (16)                            ):
56: (8)                     self.sheetId = sheetId
57: (8)                     self.refreshError = refreshError
58: (8)                     self.row = row
59: (0)             class ExternalSheetDataSet(Serialisable):
60: (4)                 sheetData = Sequence(expected_type=ExternalSheetData, )
61: (4)                 __elements__ = ('sheetData',)
62: (4)                 def __init__(self,
63: (17)                             sheetData=None,
64: (16)                            ):
```

```
65: (8)                 self.sheetData = sheetData
66: (0)             class ExternalSheetNames(Serialisable):
67: (4)                 sheetName = ValueSequence(expected_type=str)
68: (4)                 __elements__ = ('sheetName',)
69: (4)                 def __init__(self,
70: (17)                             sheetName=(),
71: (16)                            ):
72: (8)                     self.sheetName = sheetName
73: (0)             class ExternalDefinedName(Serialisable):
74: (4)                 tagname = "definedName"
75: (4)                 name = String()
76: (4)                 refersTo = String(allow_none=True)
77: (4)                 sheetId = Integer(allow_none=True)
78: (4)                 def __init__(self,
79: (17)                             name=None,
80: (17)                             refersTo=None,
81: (17)                             sheetId=None,
82: (16)                            ):
83: (8)                     self.name = name
84: (8)                     self.refersTo = refersTo
85: (8)                     self.sheetId = sheetId
86: (0)             class ExternalBook(Serialisable):
87: (4)                 tagname = "externalBook"
88: (4)                 sheetNames = Typed(expected_type=ExternalSheetNames, allow_none=True)
89: (4)                 definedNames = NestedSequence(expected_type=ExternalDefinedName)
90: (4)                 sheetDataSet = Typed(expected_type=ExternalSheetDataSet, allow_none=True)
91: (4)                 id = Relation()
92: (4)                 __elements__ = ('sheetNames', 'definedNames', 'sheetDataSet')
93: (4)                 def __init__(self,
94: (17)                             sheetNames=None,
95: (17)                             definedNames=(),
96: (17)                             sheetDataSet=None,
97: (17)                             id=None,
98: (16)                            ):
99: (8)                     self.sheetNames = sheetNames
100: (8)                    self.definedNames = definedNames
101: (8)                    self.sheetDataSet = sheetDataSet
102: (8)                    self.id = id
103: (0)            class ExternalLink(Serialisable):
104: (4)                tagname = "externalLink"
105: (4)                _id = None
106: (4)                _path = "/xl/externalLinks/externalLink{0}.xml"
107: (4)                _rel_type = "externalLink"
108: (4)                mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.externalLink+xml"
109: (4)                externalBook = Typed(expected_type=ExternalBook, allow_none=True)
110: (4)                file_link = Typed(expected_type=Relationship, allow_none=True) # link to
external file
111: (4)                __elements__ = ('externalBook', )
112: (4)                def __init__(self,
113: (17)                            externalBook=None,
114: (17)                            ddeLink=None,
115: (17)                            oleLink=None,
116: (17)                            extLst=None,
117: (16)                           ):
118: (8)                    self.externalBook = externalBook
119: (4)                def to_tree(self):
120: (8)                    node = super().to_tree()
121: (8)                    node.set("xmlns", SHEET_MAIN_NS)
122: (8)                    return node
123: (4)                @property
124: (4)                def path(self):
125: (8)                    return self._path.format(self._id)
126: (0)            def read_external_link(archive, book_path):
127: (4)                src = archive.read(book_path)
128: (4)                node = fromstring(src)
129: (4)                book = ExternalLink.from_tree(node)
130: (4)                link_path = get_rels_path(book_path)
131: (4)                deps = get_dependents(archive, link_path)
```

```
132: (4)                book.file_link = deps[0]
133: (4)                return book

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

File 144 - __init__.py:

1: (0)              from .external import ExternalLink

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

File 145 - __init__.py:

1: (0)

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

File 146 - inference.py:

1: (0)              """
2: (0)              Type inference functions
3: (0)              """
4: (0)              import datetime
5: (0)              import re
6: (0)              from openpyxl.styles import numbers
7: (0)              PERCENT_REGEX = re.compile(r'^(?P<number>\-?[0-9]*\.?[0-9]*\s?)\%$')
8: (0)              TIME_REGEX = re.compile(r"""
9: (0)              ^(?: # HH:MM and HH:MM:SS
10: (0)             (?P<hour>[0-1]{0,1}[0-9]{2}):
11: (0)             (?P<minute>[0-5][0-9]):?
12: (0)             (?P<second>[0-5][0-9])?$)
13: (0)             |
14: (0)             ^(?: # MM:SS.
15: (0)             ([0-5][0-9]):
16: (0)             ([0-5][0-9])?\.
17: (0)             (?P<microsecond>\d{1,6}))
18: (0)             """, re.VERBOSE)
19: (0)             NUMBER_REGEX = re.compile(r'^-?([\d]|[\d]+\.[\d]*|\.[\d]+|[1-9][\d]+\.?[\d]*)
((E|e)[-+]?[\d]+)?$')
20: (0)             def cast_numeric(value):
21: (4)                 """Explicitly convert a string to a numeric value"""
22: (4)                 if NUMBER_REGEX.match(value):
23: (8)                     try:
24: (12)                        return int(value)
25: (8)                     except ValueError:
26: (12)                        return float(value)
27: (0)             def cast_percentage(value):
28: (4)                 """Explicitly convert a string to numeric value and format as a
29: (4)                 percentage"""
30: (4)                 match = PERCENT_REGEX.match(value)
31: (4)                 if match:
32: (8)                     return float(match.group('number')) / 100
33: (0)             def cast_time(value):
34: (4)                 """Explicitly convert a string to a number and format as datetime or
35: (4)                 time"""
36: (4)                 match = TIME_REGEX.match(value)
37: (4)                 if match:
38: (8)                     if match.group("microsecond") is not None:
39: (12)                        value = value[:12]
40: (12)                        pattern = "%M:%S.%f"
41: (8)                     elif match.group('second') is None:
42: (12)                        pattern = "%H:%M"
43: (8)                     else:
44: (12)                        pattern = "%H:%M:%S"
45: (8)                     value = datetime.datetime.strptime(value, pattern)
46: (8)                     return value.time()

----------------------------------------
```

File 147 - protection.py:

```
(0)              def hash_password(plaintext_password=''):
(4)                  """
(4)                  Create a password hash from a given string for protecting a worksheet
(4)                  only. This will not work for encrypting a workbook.
(4)                  This method is based on the algorithm provided by
(4)                  Daniel Rentz of OpenOffice and the PEAR package
(4)                  Spreadsheet_Excel_Writer by Xavier Noguer <xnoguer@rezebra.com>.
(4)                  See also http://blogs.msdn.com/b/ericwhite/archive/2008/02/23/the-legacy-hashing-algorithm-in-open-xml.aspx
(4)                  """
(4)                  password = 0x0000
(4)                  for idx, char in enumerate(plaintext_password, 1):
(8)                      value = ord(char) << idx
(8)                      rotated_bits = value >> 15
(8)                      value &= 0x7fff
(8)                      password ^= (value | rotated_bits)
(4)                  password ^= len(plaintext_password)
(4)                  password ^= 0xCE4B
(4)                  return str(hex(password)).upper()[2:]
```

File 148 - properties.py:

```
(0)              from openpyxl.descriptors.serialisable import Serialisable
(0)              from openpyxl.descriptors import (
(4)                  String,
(4)                  Float,
(4)                  Integer,
(4)                  Bool,
(4)                  NoneSet,
(4)                  Set,
(0)              )
(0)              from openpyxl.descriptors.excel import Guid
(0)              class WorkbookProperties(Serialisable):
(4)                  tagname = "workbookPr"
(4)                  date1904 = Bool(allow_none=True)
(4)                  dateCompatibility = Bool(allow_none=True)
(4)                  showObjects = NoneSet(values=(['all', 'placeholders']))
(4)                  showBorderUnselectedTables = Bool(allow_none=True)
(4)                  filterPrivacy = Bool(allow_none=True)
(4)                  promptedSolutions = Bool(allow_none=True)
(4)                  showInkAnnotation = Bool(allow_none=True)
(4)                  backupFile = Bool(allow_none=True)
(4)                  saveExternalLinkValues = Bool(allow_none=True)
(4)                  updateLinks = NoneSet(values=(['userSet', 'never', 'always']))
(4)                  codeName = String(allow_none=True)
(4)                  hidePivotFieldList = Bool(allow_none=True)
(4)                  showPivotChartFilter = Bool(allow_none=True)
(4)                  allowRefreshQuery = Bool(allow_none=True)
(4)                  publishItems = Bool(allow_none=True)
(4)                  checkCompatibility = Bool(allow_none=True)
(4)                  autoCompressPictures = Bool(allow_none=True)
(4)                  refreshAllConnections = Bool(allow_none=True)
(4)                  defaultThemeVersion = Integer(allow_none=True)
(4)                  def __init__(self,
(17)                              date1904=None,
(17)                              dateCompatibility=None,
(17)                              showObjects=None,
(17)                              showBorderUnselectedTables=None,
(17)                              filterPrivacy=None,
(17)                              promptedSolutions=None,
(17)                              showInkAnnotation=None,
(17)                              backupFile=None,
(17)                              saveExternalLinkValues=None,
(17)                              updateLinks=None,
(17)                              codeName=None,
44: (17)                                 hidePivotFieldList=None,
45: (17)                                 showPivotChartFilter=None,
46: (17)                                 allowRefreshQuery=None,
47: (17)                                 publishItems=None,
48: (17)                                 checkCompatibility=None,
49: (17)                                 autoCompressPictures=None,
50: (17)                                 refreshAllConnections=None,
51: (17)                                 defaultThemeVersion=None,
52: (16)                                ):
53: (8)                         self.date1904 = date1904
54: (8)                         self.dateCompatibility = dateCompatibility
55: (8)                         self.showObjects = showObjects
56: (8)                         self.showBorderUnselectedTables = showBorderUnselectedTables
57: (8)                         self.filterPrivacy = filterPrivacy
58: (8)                         self.promptedSolutions = promptedSolutions
59: (8)                         self.showInkAnnotation = showInkAnnotation
60: (8)                         self.backupFile = backupFile
61: (8)                         self.saveExternalLinkValues = saveExternalLinkValues
62: (8)                         self.updateLinks = updateLinks
63: (8)                         self.codeName = codeName
64: (8)                         self.hidePivotFieldList = hidePivotFieldList
65: (8)                         self.showPivotChartFilter = showPivotChartFilter
66: (8)                         self.allowRefreshQuery = allowRefreshQuery
67: (8)                         self.publishItems = publishItems
68: (8)                         self.checkCompatibility = checkCompatibility
69: (8)                         self.autoCompressPictures = autoCompressPictures
70: (8)                         self.refreshAllConnections = refreshAllConnections
71: (8)                         self.defaultThemeVersion = defaultThemeVersion
72: (0)                 class CalcProperties(Serialisable):
73: (4)                     tagname = "calcPr"
74: (4)                     calcId = Integer()
75: (4)                     calcMode = NoneSet(values=(['manual', 'auto', 'autoNoTable']))
76: (4)                     fullCalcOnLoad = Bool(allow_none=True)
77: (4)                     refMode = NoneSet(values=(['A1', 'R1C1']))
78: (4)                     iterate = Bool(allow_none=True)
79: (4)                     iterateCount = Integer(allow_none=True)
80: (4)                     iterateDelta = Float(allow_none=True)
81: (4)                     fullPrecision = Bool(allow_none=True)
82: (4)                     calcCompleted = Bool(allow_none=True)
83: (4)                     calcOnSave = Bool(allow_none=True)
84: (4)                     concurrentCalc = Bool(allow_none=True)
85: (4)                     concurrentManualCount = Integer(allow_none=True)
86: (4)                     forceFullCalc = Bool(allow_none=True)
87: (4)                     def __init__(self,
88: (17)                                 calcId=124519,
89: (17)                                 calcMode=None,
90: (17)                                 fullCalcOnLoad=True,
91: (17)                                 refMode=None,
92: (17)                                 iterate=None,
93: (17)                                 iterateCount=None,
94: (17)                                 iterateDelta=None,
95: (17)                                 fullPrecision=None,
96: (17)                                 calcCompleted=None,
97: (17)                                 calcOnSave=None,
98: (17)                                 concurrentCalc=None,
99: (17)                                 concurrentManualCount=None,
100: (17)                                forceFullCalc=None,
101: (16)                               ):
102: (8)                        self.calcId = calcId
103: (8)                        self.calcMode = calcMode
104: (8)                        self.fullCalcOnLoad = fullCalcOnLoad
105: (8)                        self.refMode = refMode
106: (8)                        self.iterate = iterate
107: (8)                        self.iterateCount = iterateCount
108: (8)                        self.iterateDelta = iterateDelta
109: (8)                        self.fullPrecision = fullPrecision
110: (8)                        self.calcCompleted = calcCompleted
111: (8)                        self.calcOnSave = calcOnSave
112: (8)                        self.concurrentCalc = concurrentCalc
```

```
113: (8)                    self.concurrentManualCount = concurrentManualCount
114: (8)                    self.forceFullCalc = forceFullCalc
115: (0)            class FileVersion(Serialisable):
116: (4)                tagname = "fileVersion"
117: (4)                appName = String(allow_none=True)
118: (4)                lastEdited = String(allow_none=True)
119: (4)                lowestEdited = String(allow_none=True)
120: (4)                rupBuild = String(allow_none=True)
121: (4)                codeName = Guid(allow_none=True)
122: (4)                def __init__(self,
123: (17)                             appName=None,
124: (17)                             lastEdited=None,
125: (17)                             lowestEdited=None,
126: (17)                             rupBuild=None,
127: (17)                             codeName=None,
128: (16)                            ):
129: (8)                    self.appName = appName
130: (8)                    self.lastEdited = lastEdited
131: (8)                    self.lowestEdited = lowestEdited
132: (8)                    self.rupBuild = rupBuild
133: (8)                    self.codeName = codeName

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

File 149 - protection.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Alias,
4: (4)                  Typed,
5: (4)                  String,
6: (4)                  Float,
7: (4)                  Integer,
8: (4)                  Bool,
9: (4)                  NoneSet,
10: (4)                 Set,
11: (0)             )
12: (0)             from openpyxl.descriptors.excel import (
13: (4)                 ExtensionList,
14: (4)                 HexBinary,
15: (4)                 Guid,
16: (4)                 Relation,
17: (4)                 Base64Binary,
18: (0)             )
19: (0)             from openpyxl.utils.protection import hash_password
20: (0)             class WorkbookProtection(Serialisable):
21: (4)                 _workbook_password, _revisions_password = None, None
22: (4)                 tagname = "workbookPr"
23: (4)                 workbook_password = Alias("workbookPassword")
24: (4)                 workbookPasswordCharacterSet = String(allow_none=True)
25: (4)                 revision_password = Alias("revisionsPassword")
26: (4)                 revisionsPasswordCharacterSet = String(allow_none=True)
27: (4)                 lockStructure = Bool(allow_none=True)
28: (4)                 lock_structure = Alias("lockStructure")
29: (4)                 lockWindows = Bool(allow_none=True)
30: (4)                 lock_windows = Alias("lockWindows")
31: (4)                 lockRevision = Bool(allow_none=True)
32: (4)                 lock_revision = Alias("lockRevision")
33: (4)                 revisionsAlgorithmName = String(allow_none=True)
34: (4)                 revisionsHashValue = Base64Binary(allow_none=True)
35: (4)                 revisionsSaltValue = Base64Binary(allow_none=True)
36: (4)                 revisionsSpinCount = Integer(allow_none=True)
37: (4)                 workbookAlgorithmName = String(allow_none=True)
38: (4)                 workbookHashValue = Base64Binary(allow_none=True)
39: (4)                 workbookSaltValue = Base64Binary(allow_none=True)
40: (4)                 workbookSpinCount = Integer(allow_none=True)
41: (4)                 __attrs__ = ('workbookPassword', 'workbookPasswordCharacterSet',
'revisionsPassword',
42: (17)                             'revisionsPasswordCharacterSet', 'lockStructure',
'lockWindows', 'lockRevision',
43: (17)                             'revisionsAlgorithmName', 'revisionsHashValue',
'revisionsSaltValue',
44: (17)                             'revisionsSpinCount', 'workbookAlgorithmName',
'workbookHashValue',
45: (17)                             'workbookSaltValue', 'workbookSpinCount')
46: (4)                 def __init__(self,
47: (17)                             workbookPassword=None,
48: (17)                             workbookPasswordCharacterSet=None,
49: (17)                             revisionsPassword=None,
50: (17)                             revisionsPasswordCharacterSet=None,
51: (17)                             lockStructure=None,
52: (17)                             lockWindows=None,
53: (17)                             lockRevision=None,
54: (17)                             revisionsAlgorithmName=None,
55: (17)                             revisionsHashValue=None,
56: (17)                             revisionsSaltValue=None,
57: (17)                             revisionsSpinCount=None,
58: (17)                             workbookAlgorithmName=None,
59: (17)                             workbookHashValue=None,
60: (17)                             workbookSaltValue=None,
61: (17)                             workbookSpinCount=None,
62: (16)                            ):
63: (8)                     if workbookPassword is not None:
64: (12)                        self.workbookPassword = workbookPassword
65: (8)                     self.workbookPasswordCharacterSet = workbookPasswordCharacterSet
66: (8)                     if revisionsPassword is not None:
67: (12)                        self.revisionsPassword = revisionsPassword
68: (8)                     self.revisionsPasswordCharacterSet = revisionsPasswordCharacterSet
69: (8)                     self.lockStructure = lockStructure
70: (8)                     self.lockWindows = lockWindows
71: (8)                     self.lockRevision = lockRevision
72: (8)                     self.revisionsAlgorithmName = revisionsAlgorithmName
73: (8)                     self.revisionsHashValue = revisionsHashValue
74: (8)                     self.revisionsSaltValue = revisionsSaltValue
75: (8)                     self.revisionsSpinCount = revisionsSpinCount
76: (8)                     self.workbookAlgorithmName = workbookAlgorithmName
77: (8)                     self.workbookHashValue = workbookHashValue
78: (8)                     self.workbookSaltValue = workbookSaltValue
79: (8)                     self.workbookSpinCount = workbookSpinCount
80: (4)                 def set_workbook_password(self, value='', already_hashed=False):
81: (8)                     """Set a password on this workbook."""
82: (8)                     if not already_hashed:
83: (12)                        value = hash_password(value)
84: (8)                     self._workbook_password = value
85: (4)                 @property
86: (4)                 def workbookPassword(self):
87: (8)                     """Return the workbook password value, regardless of hash."""
88: (8)                     return self._workbook_password
89: (4)                 @workbookPassword.setter
90: (4)                 def workbookPassword(self, value):
91: (8)                     """Set a workbook password directly, forcing a hash step."""
92: (8)                     self.set_workbook_password(value)
93: (4)                 def set_revisions_password(self, value='', already_hashed=False):
94: (8)                     """Set a revision password on this workbook."""
95: (8)                     if not already_hashed:
96: (12)                        value = hash_password(value)
97: (8)                     self._revisions_password = value
98: (4)                 @property
99: (4)                 def revisionsPassword(self):
100: (8)                    """Return the revisions password value, regardless of hash."""
101: (8)                    return self._revisions_password
102: (4)                @revisionsPassword.setter
103: (4)                def revisionsPassword(self, value):
104: (8)                    """Set a revisions password directly, forcing a hash step."""
105: (8)                    self.set_revisions_password(value)
106: (4)                @classmethod
107: (4)                def from_tree(cls, node):
108: (8)                    """Don't hash passwords when deserialising from XML"""
```

```
109: (8)                    self = super().from_tree(node)
110: (8)                    if self.workbookPassword:
111: (12)                       self.set_workbook_password(node.get('workbookPassword'),
already_hashed=True)
112: (8)                    if self.revisionsPassword:
113: (12)                       self.set_revisions_password(node.get('revisionsPassword'),
already_hashed=True)
114: (8)                    return self
115: (0)            DocumentSecurity = WorkbookProtection
116: (0)            class FileSharing(Serialisable):
117: (4)                tagname = "fileSharing"
118: (4)                readOnlyRecommended = Bool(allow_none=True)
119: (4)                userName = String(allow_none=True)
120: (4)                reservationPassword = HexBinary(allow_none=True)
121: (4)                algorithmName = String(allow_none=True)
122: (4)                hashValue = Base64Binary(allow_none=True)
123: (4)                saltValue = Base64Binary(allow_none=True)
124: (4)                spinCount = Integer(allow_none=True)
125: (4)                def __init__(self,
126: (17)                            readOnlyRecommended=None,
127: (17)                            userName=None,
128: (17)                            reservationPassword=None,
129: (17)                            algorithmName=None,
130: (17)                            hashValue=None,
131: (17)                            saltValue=None,
132: (17)                            spinCount=None,
133: (16)                           ):
134: (8)                    self.readOnlyRecommended = readOnlyRecommended
135: (8)                    self.userName = userName
136: (8)                    self.reservationPassword = reservationPassword
137: (8)                    self.algorithmName = algorithmName
138: (8)                    self.hashValue = hashValue
139: (8)                    self.saltValue = saltValue
140: (8)                    self.spinCount = spinCount

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

File 150 - smart_tags.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence,
4: (4)                  String,
5: (4)                  Bool,
6: (4)                  NoneSet,
7: (0)              )
8: (0)              class SmartTag(Serialisable):
9: (4)                  tagname = "smartTagType"
10: (4)                 namespaceUri = String(allow_none=True)
11: (4)                 name = String(allow_none=True)
12: (4)                 url = String(allow_none=True)
13: (4)                 def __init__(self,
14: (17)                             namespaceUri=None,
15: (17)                             name=None,
16: (17)                             url=None,
17: (16)                            ):
18: (8)                     self.namespaceUri = namespaceUri
19: (8)                     self.name = name
20: (8)                     self.url = url
21: (0)             class SmartTagList(Serialisable):
22: (4)                 tagname = "smartTagTypes"
23: (4)                 smartTagType = Sequence(expected_type=SmartTag, allow_none=True)
24: (4)                 __elements__ = ('smartTagType',)
25: (4)                 def __init__(self,
26: (17)                             smartTagType=(),
27: (16)                            ):
28: (8)                     self.smartTagType = smartTagType
29: (0)             class SmartTagProperties(Serialisable):
30: (4)                 tagname = "smartTagPr"
```

```
31: (4)                 embed = Bool(allow_none=True)
32: (4)                 show = NoneSet(values=(['all', 'noIndicator']))
33: (4)                 def __init__(self,
34: (17)                             embed=None,
35: (17)                             show=None,
36: (16)                            ):
37: (8)                     self.embed = embed
38: (8)                     self.show = show

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

File 151 - _read_only.py:

1: (0)              """ Read worksheets on-demand
2: (0)              """
3: (0)              from .worksheet import Worksheet
4: (0)              from openpyxl.cell.read_only import ReadOnlyCell, EMPTY_CELL
5: (0)              from openpyxl.utils import get_column_letter
6: (0)              from ._reader import WorkSheetParser
7: (0)              from openpyxl.workbook.defined_name import DefinedNameDict
8: (0)              def read_dimension(source):
9: (4)                  parser = WorkSheetParser(source, [])
10: (4)                 return parser.parse_dimensions()
11: (0)             class ReadOnlyWorksheet:
12: (4)                 _min_column = 1
13: (4)                 _min_row = 1
14: (4)                 _max_column = _max_row = None
15: (4)                 cell = Worksheet.cell
16: (4)                 iter_rows = Worksheet.iter_rows
17: (4)                 values = Worksheet.values
18: (4)                 rows = Worksheet.rows
19: (4)                 __getitem__ = Worksheet.__getitem__
20: (4)                 __iter__ = Worksheet.__iter__
21: (4)                 def __init__(self, parent_workbook, title, worksheet_path,
shared_strings):
22: (8)                     self.parent = parent_workbook
23: (8)                     self.title = title
24: (8)                     self.sheet_state = 'visible'
25: (8)                     self._current_row = None
26: (8)                     self._worksheet_path = worksheet_path
27: (8)                     self._shared_strings = shared_strings
28: (8)                     self._get_size()
29: (8)                     self.defined_names = DefinedNameDict()
30: (4)                 def _get_size(self):
31: (8)                     src = self._get_source()
32: (8)                     parser = WorkSheetParser(src, [])
33: (8)                     dimensions = parser.parse_dimensions()
34: (8)                     src.close()
35: (8)                     if dimensions is not None:
36: (12)                        self._min_column, self._min_row, self._max_column, self._max_row =
dimensions
37: (4)                 def _get_source(self):
38: (8)                     """Parse xml source on demand, must close after use"""
39: (8)                     return self.parent._archive.open(self._worksheet_path)
40: (4)                 def _cells_by_row(self, min_col, min_row, max_col, max_row,
values_only=False):
41: (8)                     """
42: (8)                     The source worksheet file may have columns or rows missing.
43: (8)                     Missing cells will be created.
44: (8)                     """
45: (8)                     filler = EMPTY_CELL
46: (8)                     if values_only:
47: (12)                        filler = None
48: (8)                     max_col = max_col or self.max_column
49: (8)                     max_row = max_row or self.max_row
50: (8)                     empty_row = []
51: (8)                     if max_col is not None:
52: (12)                        empty_row = (filler,) * (max_col + 1 - min_col)
53: (8)                     counter = min_row
```

```
(8)                         idx = 1
(8)                         with self._get_source() as src:
(12)                            parser = WorkSheetParser(src,
(37)                                                     self._shared_strings,
(37)                                                     data_only=self.parent.data_only,
(37)                                                     epoch=self.parent.epoch,
(37)                                                     date_formats=self.parent._date_formats,
(37)                                                     timedelta_formats=self.parent._timedelta_formats)
(12)                            for idx, row in parser.parse():
(16)                                if max_row is not None and idx > max_row:
(20)                                    break
(16)                                for _ in range(counter, idx):
(20)                                    counter += 1
(20)                                    yield empty_row
(16)                                if counter <= idx:
(20)                                    row = self._get_row(row, min_col, max_col, values_only)
(20)                                    counter += 1
(20)                                    yield row
(8)                         if max_row is not None and max_row < idx:
(12)                            for _ in range(counter, max_row+1):
(16)                                yield empty_row
(4)                     def _get_row(self, row, min_col=1, max_col=None, values_only=False):
(8)                         """
(8)                         Make sure a row contains always the same number of cells or values
(8)                         """
(8)                         if not row and not max_col: # in case someone wants to force rows where there aren't any
(12)                            return ()
(8)                         max_col = max_col or  row[-1]['column']
(8)                         row_width = max_col + 1 - min_col
(8)                         new_row = [EMPTY_CELL] * row_width
(8)                         if values_only:
(12)                            new_row = [None] * row_width
(8)                         for cell in row:
(12)                            counter = cell['column']
(12)                            if min_col <= counter <= max_col:
(16)                                idx = counter - min_col # position in list of cells returned
(16)                                new_row[idx] = cell['value']
(16)                                if not values_only:
(20)                                    new_row[idx] = ReadOnlyCell(self, **cell)
(8)                         return tuple(new_row)
(4)                     def _get_cell(self, row, column):
(8)                         """Cells are returned by a generator which can be empty"""
(8)                         for row in self._cells_by_row(column, row, column, row):
(12)                            if row:
(16)                                return row[0]
(8)                         return EMPTY_CELL
(4)                     def calculate_dimension(self, force=False):
(8)                         if not all([self.max_column, self.max_row]):
(12)                            if force:
(16)                                self._calculate_dimension()
(12)                            else:
(16)                                raise ValueError("Worksheet is unsized, use calculate_dimension(force=True)")
(8)                         return f"{get_column_letter(self.min_column)}{self.min_row}:{get_column_letter(self.max_column)}{self.max_row}"
(4)                     def _calculate_dimension(self):
(8)                         """
(8)                         Loop through all the cells to get the size of a worksheet.
(8)                         Do this only if it is explicitly requested.
(8)                         """
(8)                         max_col = 0
(8)                         for r in self.rows:
(12)                            if not r:
(16)                                continue
(12)                            cell = r[-1]
(12)                            max_col = max(max_col, cell.column)
(8)                         self._max_row = cell.row
```

```
119: (8)                        self._max_column = max_col
120: (4)                    def reset_dimensions(self):
121: (8)                        """
122: (8)                        Remove worksheet dimensions if these are incorrect in the worksheet
source.
123: (8)                        NB. This probably indicates a bug in the library or application that
created
124: (8)                        the workbook.
125: (8)                        """
126: (8)                        self._max_row = self._max_column = None
127: (4)                    @property
128: (4)                    def min_row(self):
129: (8)                        return self._min_row
130: (4)                    @property
131: (4)                    def max_row(self):
132: (8)                        return self._max_row
133: (4)                    @property
134: (4)                    def min_column(self):
135: (8)                        return self._min_column
136: (4)                    @property
137: (4)                    def max_column(self):
138: (8)                        return self._max_column

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

File 152 - _write_only.py:

1: (0)                  """Write worksheets to xml representations in an optimized way"""
2: (0)                  from inspect import isgenerator
3: (0)                  from openpyxl.cell import Cell, WriteOnlyCell
4: (0)                  from openpyxl.workbook.child import _WorkbookChild
5: (0)                  from .worksheet import Worksheet
6: (0)                  from openpyxl.utils.exceptions import WorkbookAlreadySaved
7: (0)                  from ._writer import WorksheetWriter
8: (0)                  class WriteOnlyWorksheet(_WorkbookChild):
9: (4)                      """
10: (4)                     Streaming worksheet. Optimised to reduce memory by writing rows just in
11: (4)                     time.
12: (4)                     Cells can be styled and have comments Styles for rows and columns
13: (4)                     must be applied before writing cells
14: (4)                     """
15: (4)                     __saved = False
16: (4)                     _writer = None
17: (4)                     _rows = None
18: (4)                     _rel_type = Worksheet._rel_type
19: (4)                     _path = Worksheet._path
20: (4)                     mime_type = Worksheet.mime_type
21: (4)                     _add_row = Worksheet._add_row
22: (4)                     _add_column = Worksheet._add_column
23: (4)                     add_chart = Worksheet.add_chart
24: (4)                     add_image = Worksheet.add_image
25: (4)                     add_table = Worksheet.add_table
26: (4)                     tables = Worksheet.tables
27: (4)                     print_titles = Worksheet.print_titles
28: (4)                     print_title_cols = Worksheet.print_title_cols
29: (4)                     print_title_rows = Worksheet.print_title_rows
30: (4)                     freeze_panes = Worksheet.freeze_panes
31: (4)                     print_area = Worksheet.print_area
32: (4)                     sheet_view = Worksheet.sheet_view
33: (4)                     _setup = Worksheet._setup
34: (4)                     def __init__(self, parent, title):
35: (8)                         super().__init__(parent, title)
36: (8)                         self._max_col = 0
37: (8)                         self._max_row = 0
38: (8)                         self._setup()
39: (4)                     @property
40: (4)                     def closed(self):
41: (8)                         return self.__saved
42: (4)                     def _write_rows(self):
        """
        Send rows to the writer's stream
        """
        try:
            xf = self._writer.xf.send(True)
        except StopIteration:
            self._already_saved()
        with xf.element("sheetData"):
            row_idx = 1
            try:
                while True:
                    row = (yield)
                    row = self._values_to_row(row, row_idx)
                    self._writer.write_row(xf, row, row_idx)
                    row_idx += 1
            except GeneratorExit:
                pass
        self._writer.xf.send(None)
    def _get_writer(self):
        if self._writer is None:
            self._writer = WorksheetWriter(self)
            self._writer.write_top()
    def close(self):
        if self.__saved:
            self._already_saved()
        self._get_writer()
        if self._rows is None:
            self._writer.write_rows()
        else:
            self._rows.close()
        self._writer.write_tail()
        self._writer.close()
        self.__saved = True
    def append(self, row):
        """
        :param row: iterable containing values to append
        :type row: iterable
        """
        if (not isgenerator(row) and
            not isinstance(row, (list, tuple, range))
            ):
            self._invalid_row(row)
        self._get_writer()
        if self._rows is None:
            self._rows = self._write_rows()
            next(self._rows)
        self._rows.send(row)
    def _values_to_row(self, values, row_idx):
        """
        Convert whatever has been appended into a form suitable for work_rows
        """
        cell = WriteOnlyCell(self)
        for col_idx, value in enumerate(values, 1):
            if value is None:
                continue
            try:
                cell.value = value
            except ValueError:
                if isinstance(value, Cell):
                    cell = value
                else:
                    raise ValueError
            cell.column = col_idx
            cell.row = row_idx
            if cell.hyperlink is not None:
                cell.hyperlink.ref = cell.coordinate
            yield cell
            if cell.has_style or cell.hyperlink:
                cell = WriteOnlyCell(self)
```

```
(4)                 def _already_saved(self):
(8)                     raise WorkbookAlreadySaved('Workbook has already been saved and cannot be modified or saved anymore.')
(4)                 def _invalid_row(self, iterable):
(8)                     raise TypeError('Value must be a list, tuple, range or a generator Supplied value is {0}'.format(
(12)                        type(iterable))
(24)                                    )
```

File 153 - indexed_list.py:

```
(0)             class IndexedList(list):
(4)                 """
(4)                 List with optimised access by value
(4)                 Based on Alex Martelli's recipe
(4)                 http://code.activestate.com/recipes/52303-the-auxiliary-dictionary-idiom-for-sequences-with-/
(4)                 """
(4)                 _dict = {}
(4)                 def __init__(self, iterable=None):
(8)                     self.clean = True
(8)                     self._dict = {}
(8)                     if iterable is not None:
(12)                        self.clean = False
(12)                        for idx, val in enumerate(iterable):
(16)                            self._dict[val] = idx
(16)                            list.append(self, val)
(4)                 def _rebuild_dict(self):
(8)                     self._dict = {}
(8)                     idx = 0
(8)                     for value in self:
(12)                        if value not in self._dict:
(16)                            self._dict[value] = idx
(16)                            idx += 1
(8)                     self.clean = True
(4)                 def __contains__(self, value):
(8)                     if not self.clean:
(12)                        self._rebuild_dict()
(8)                     return value in self._dict
(4)                 def index(self, value):
(8)                     if value in self:
(12)                        return self._dict[value]
(8)                     raise ValueError
(4)                 def append(self, value):
(8)                     if value not in self._dict:
(12)                        self._dict[value] = len(self)
(12)                        list.append(self, value)
(4)                 def add(self, value):
(8)                     self.append(value)
(8)                     return self._dict[value]
```

File 154 - defined_name.py:

```
(0)             from collections import defaultdict
(0)             import re
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 Alias,
(4)                 String,
(4)                 Integer,
(4)                 Bool,
(4)                 Sequence,
(4)                 Descriptor,
(0)             )
(0)             from openpyxl.compat import safe_string
```

```
from openpyxl.formula import Tokenizer
from openpyxl.utils.cell import SHEETRANGE_RE
RESERVED = frozenset(["Print_Area", "Print_Titles", "Criteria",
                      "_FilterDatabase", "Extract", "Consolidate_Area",
                      "Sheet_Title"])
_names = "|".join(RESERVED)
RESERVED_REGEX = re.compile(r"^_xlnm\.(?P<name>{0})".format(_names))
class DefinedName(Serialisable):
    tagname = "definedName"
    name = String() # unique per workbook/worksheet
    comment = String(allow_none=True)
    customMenu = String(allow_none=True)
    description = String(allow_none=True)
    help = String(allow_none=True)
    statusBar = String(allow_none=True)
    localSheetId = Integer(allow_none=True)
    hidden = Bool(allow_none=True)
    function = Bool(allow_none=True)
    vbProcedure = Bool(allow_none=True)
    xlm = Bool(allow_none=True)
    functionGroupId = Integer(allow_none=True)
    shortcutKey = String(allow_none=True)
    publishToServer = Bool(allow_none=True)
    workbookParameter = Bool(allow_none=True)
    attr_text = Descriptor()
    value = Alias("attr_text")
    def __init__(self,
                 name=None,
                 comment=None,
                 customMenu=None,
                 description=None,
                 help=None,
                 statusBar=None,
                 localSheetId=None,
                 hidden=None,
                 function=None,
                 vbProcedure=None,
                 xlm=None,
                 functionGroupId=None,
                 shortcutKey=None,
                 publishToServer=None,
                 workbookParameter=None,
                 attr_text=None
                ):
        self.name = name
        self.comment = comment
        self.customMenu = customMenu
        self.description = description
        self.help = help
        self.statusBar = statusBar
        self.localSheetId = localSheetId
        self.hidden = hidden
        self.function = function
        self.vbProcedure = vbProcedure
        self.xlm = xlm
        self.functionGroupId = functionGroupId
        self.shortcutKey = shortcutKey
        self.publishToServer = publishToServer
        self.workbookParameter = workbookParameter
        self.attr_text = attr_text
    @property
    def type(self):
        tok = Tokenizer("=" + self.value)
        parsed = tok.items[0]
        if parsed.type == "OPERAND":
            return parsed.subtype
        return parsed.type
    @property
    def destinations(self):
```

```
(8)             if self.type == "RANGE":
(12)                tok = Tokenizer("=" + self.value)
(12)                for part in tok.items:
(16)                    if part.subtype == "RANGE":
(20)                        m = SHEETRANGE_RE.match(part.value)
(20)                        sheetname = m.group('notquoted') or m.group('quoted')
(20)                        yield sheetname, m.group('cells')
(4)         @property
(4)         def is_reserved(self):
(8)             m = RESERVED_REGEX.match(self.name)
(8)             if m:
(12)                return m.group("name")
(4)         @property
(4)         def is_external(self):
(8)             return re.compile(r"^\[\d+\].*").match(self.value) is not None
(4)         def __iter__(self):
(8)             for key in self.__attrs__:
(12)                if key == "attr_text":
(16)                    continue
(12)                v = getattr(self, key)
(12)                if v is not None:
(16)                    if v in RESERVED:
(20)                        v = "_xlnm." + v
(16)                    yield key, safe_string(v)
(0)     class DefinedNameDict(dict):
(4)         """
(4)         Utility class for storing defined names.
(4)         Allows access by name and separation of global and scoped names
(4)         """
(4)         def __setitem__(self, key, value):
(8)             if not isinstance(value, DefinedName):
(12)                raise TypeError("Value must be a an instance of DefinedName")
(8)             elif value.name != key:
(12)                raise ValueError("Key must be the same as the name")
(8)             super().__setitem__(key, value)
(4)         def add(self, value):
(8)             """
(8)             Add names without worrying about key and name matching.
(8)             """
(8)             self[value.name] = value
(0)     class DefinedNameList(Serialisable):
(4)         tagname = "definedNames"
(4)         definedName = Sequence(expected_type=DefinedName)
(4)         def __init__(self, definedName=()):
(8)             self.definedName = definedName
(4)         def by_sheet(self):
(8)             """
(8)             Break names down into sheet locals and globals
(8)             """
(8)             names = defaultdict(DefinedNameDict)
(8)             for defn in self.definedName:
(12)                if defn.localSheetId is None:
(16)                    if defn.name in ("_xlnm.Print_Titles", "_xlnm.Print_Area",
"_xlnm._FilterDatabase"):
(20)                        continue
(16)                    names["global"][defn.name] = defn
(12)                else:
(16)                    sheet = int(defn.localSheetId)
(16)                    names[sheet][defn.name] = defn
(8)             return names
(4)         def _duplicate(self, defn):
(8)             """
(8)             Check for whether DefinedName with the same name and scope already
(8)             exists
(8)             """
(8)             for d in self.definedName:
(12)                if d.name == defn.name and d.localSheetId == defn.localSheetId:
(16)                    return True
(4)         def __len__(self):
```

```
150: (8)                    return len(self.definedName)

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

File 155 - function_group.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence,
4: (4)                  String,
5: (4)                  Integer,
6: (0)              )
7: (0)              class FunctionGroup(Serialisable):
8: (4)                  tagname = "functionGroup"
9: (4)                  name = String()
10: (4)                 def __init__(self,
11: (17)                             name=None,
12: (16)                            ):
13: (8)                     self.name = name
14: (0)             class FunctionGroupList(Serialisable):
15: (4)                 tagname = "functionGroups"
16: (4)                 builtInGroupCount = Integer(allow_none=True)
17: (4)                 functionGroup = Sequence(expected_type=FunctionGroup, allow_none=True)
18: (4)                 __elements__ = ('functionGroup',)
19: (4)                 def __init__(self,
20: (17)                             builtInGroupCount=16,
21: (17)                             functionGroup=(),
22: (16)                            ):
23: (8)                     self.builtInGroupCount = builtInGroupCount
24: (8)                     self.functionGroup = functionGroup

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

File 156 - external_reference.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence
4: (0)              )
5: (0)              from openpyxl.descriptors.excel import (
6: (4)                  Relation,
7: (0)              )
8: (0)              class ExternalReference(Serialisable):
9: (4)                  tagname = "externalReference"
10: (4)                 id = Relation()
11: (4)                 def __init__(self, id):
12: (8)                     self.id = id

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

File 157 - ole.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Integer,
5: (4)                  String,
6: (4)                  Set,
7: (4)                  Bool,
8: (4)                  Sequence,
9: (0)              )
10: (0)             from openpyxl.drawing.spreadsheet_drawing import AnchorMarker
11: (0)             from openpyxl.xml.constants import SHEET_DRAWING_NS
12: (0)             class ObjectAnchor(Serialisable):
13: (4)                 tagname = "anchor"
14: (4)                 _from = Typed(expected_type=AnchorMarker, namespace=SHEET_DRAWING_NS)
15: (4)                 to = Typed(expected_type=AnchorMarker, namespace=SHEET_DRAWING_NS)
16: (4)                 moveWithCells = Bool(allow_none=True)
17: (4)                 sizeWithCells = Bool(allow_none=True)
```

```
 18: (4)                    z_order = Integer(allow_none=True, hyphenated=True)
 19: (4)                    def __init__(self,
 20: (17)                                _from=None,
 21: (17)                                to=None,
 22: (17)                                moveWithCells=False,
 23: (17)                                sizeWithCells=False,
 24: (17)                                z_order=None,
 25: (16)                               ):
 26: (8)                        self._from = _from
 27: (8)                        self.to = to
 28: (8)                        self.moveWithCells = moveWithCells
 29: (8)                        self.sizeWithCells = sizeWithCells
 30: (8)                        self.z_order = z_order
 31: (0)                class ObjectPr(Serialisable):
 32: (4)                    tagname = "objectPr"
 33: (4)                    anchor = Typed(expected_type=ObjectAnchor, )
 34: (4)                    locked = Bool(allow_none=True)
 35: (4)                    defaultSize = Bool(allow_none=True)
 36: (4)                    _print = Bool(allow_none=True)
 37: (4)                    disabled = Bool(allow_none=True)
 38: (4)                    uiObject = Bool(allow_none=True)
 39: (4)                    autoFill = Bool(allow_none=True)
 40: (4)                    autoLine = Bool(allow_none=True)
 41: (4)                    autoPict = Bool(allow_none=True)
 42: (4)                    macro = String()
 43: (4)                    altText = String(allow_none=True)
 44: (4)                    dde = Bool(allow_none=True)
 45: (4)                    __elements__ = ('anchor',)
 46: (4)                    def __init__(self,
 47: (17)                                anchor=None,
 48: (17)                                locked=True,
 49: (17)                                defaultSize=True,
 50: (17)                                _print=True,
 51: (17)                                disabled=False,
 52: (17)                                uiObject=False,
 53: (17)                                autoFill=True,
 54: (17)                                autoLine=True,
 55: (17)                                autoPict=True,
 56: (17)                                macro=None,
 57: (17)                                altText=None,
 58: (17)                                dde=False,
 59: (16)                               ):
 60: (8)                        self.anchor = anchor
 61: (8)                        self.locked = locked
 62: (8)                        self.defaultSize = defaultSize
 63: (8)                        self._print = _print
 64: (8)                        self.disabled = disabled
 65: (8)                        self.uiObject = uiObject
 66: (8)                        self.autoFill = autoFill
 67: (8)                        self.autoLine = autoLine
 68: (8)                        self.autoPict = autoPict
 69: (8)                        self.macro = macro
 70: (8)                        self.altText = altText
 71: (8)                        self.dde = dde
 72: (0)                class OleObject(Serialisable):
 73: (4)                    tagname = "oleObject"
 74: (4)                    objectPr = Typed(expected_type=ObjectPr, allow_none=True)
 75: (4)                    progId = String(allow_none=True)
 76: (4)                    dvAspect = Set(values=(['DVASPECT_CONTENT', 'DVASPECT_ICON']))
 77: (4)                    link = String(allow_none=True)
 78: (4)                    oleUpdate = Set(values=(['OLEUPDATE_ALWAYS', 'OLEUPDATE_ONCALL']))
 79: (4)                    autoLoad = Bool(allow_none=True)
 80: (4)                    shapeId = Integer()
 81: (4)                    __elements__ = ('objectPr',)
 82: (4)                    def __init__(self,
 83: (17)                                objectPr=None,
 84: (17)                                progId=None,
 85: (17)                                dvAspect='DVASPECT_CONTENT',
 86: (17)                                link=None,
 87: (17)                            oleUpdate=None,
 88: (17)                            autoLoad=False,
 89: (17)                            shapeId=None,
 90: (16)                           ):
 91: (8)                     self.objectPr = objectPr
 92: (8)                     self.progId = progId
 93: (8)                     self.dvAspect = dvAspect
 94: (8)                     self.link = link
 95: (8)                     self.oleUpdate = oleUpdate
 96: (8)                     self.autoLoad = autoLoad
 97: (8)                     self.shapeId = shapeId
 98: (0)             class OleObjects(Serialisable):
 99: (4)                 tagname = "oleObjects"
 100: (4)                oleObject = Sequence(expected_type=OleObject)
 101: (4)                __elements__ = ('oleObject',)
 102: (4)                def __init__(self,
 103: (17)                            oleObject=(),
 104: (16)                           ):
 105: (8)                    self.oleObject = oleObject

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

 File 158 - page.py:

 1: (0)              from openpyxl.descriptors.serialisable import Serialisable
 2: (0)              from openpyxl.descriptors import (
 3: (4)                  Float,
 4: (4)                  Bool,
 5: (4)                  Integer,
 6: (4)                  NoneSet,
 7: (4)                  )
 8: (0)              from openpyxl.descriptors.excel import UniversalMeasure, Relation
 9: (0)              class PrintPageSetup(Serialisable):
 10: (4)                 """ Worksheet print page setup """
 11: (4)                 tagname = "pageSetup"
 12: (4)                 orientation = NoneSet(values=("default", "portrait", "landscape"))
 13: (4)                 paperSize = Integer(allow_none=True)
 14: (4)                 scale = Integer(allow_none=True)
 15: (4)                 fitToHeight = Integer(allow_none=True)
 16: (4)                 fitToWidth = Integer(allow_none=True)
 17: (4)                 firstPageNumber = Integer(allow_none=True)
 18: (4)                 useFirstPageNumber = Bool(allow_none=True)
 19: (4)                 paperHeight = UniversalMeasure(allow_none=True)
 20: (4)                 paperWidth = UniversalMeasure(allow_none=True)
 21: (4)                 pageOrder = NoneSet(values=("downThenOver", "overThenDown"))
 22: (4)                 usePrinterDefaults = Bool(allow_none=True)
 23: (4)                 blackAndWhite = Bool(allow_none=True)
 24: (4)                 draft = Bool(allow_none=True)
 25: (4)                 cellComments = NoneSet(values=("asDisplayed", "atEnd"))
 26: (4)                 errors = NoneSet(values=("displayed", "blank", "dash", "NA"))
 27: (4)                 horizontalDpi = Integer(allow_none=True)
 28: (4)                 verticalDpi = Integer(allow_none=True)
 29: (4)                 copies = Integer(allow_none=True)
 30: (4)                 id = Relation()
 31: (4)                 def __init__(self,
 32: (17)                            worksheet=None,
 33: (17)                            orientation=None,
 34: (17)                            paperSize=None,
 35: (17)                            scale=None,
 36: (17)                            fitToHeight=None,
 37: (17)                            fitToWidth=None,
 38: (17)                            firstPageNumber=None,
 39: (17)                            useFirstPageNumber=None,
 40: (17)                            paperHeight=None,
 41: (17)                            paperWidth=None,
 42: (17)                            pageOrder=None,
 43: (17)                            usePrinterDefaults=None,
 44: (17)                            blackAndWhite=None,
 45: (17)                            draft=None,
 (17)                   cellComments=None,
 (17)                   errors=None,
 (17)                   horizontalDpi=None,
 (17)                   verticalDpi=None,
 (17)                   copies=None,
 (17)                   id=None):
 (8)           self._parent = worksheet
 (8)           self.orientation = orientation
 (8)           self.paperSize = paperSize
 (8)           self.scale = scale
 (8)           self.fitToHeight = fitToHeight
 (8)           self.fitToWidth = fitToWidth
 (8)           self.firstPageNumber = firstPageNumber
 (8)           self.useFirstPageNumber = useFirstPageNumber
 (8)           self.paperHeight = paperHeight
 (8)           self.paperWidth = paperWidth
 (8)           self.pageOrder = pageOrder
 (8)           self.usePrinterDefaults = usePrinterDefaults
 (8)           self.blackAndWhite = blackAndWhite
 (8)           self.draft = draft
 (8)           self.cellComments = cellComments
 (8)           self.errors = errors
 (8)           self.horizontalDpi = horizontalDpi
 (8)           self.verticalDpi = verticalDpi
 (8)           self.copies = copies
 (8)           self.id = id
 (4)       def __bool__(self):
 (8)           return bool(dict(self))
 (4)       @property
 (4)       def sheet_properties(self):
 (8)           """
 (8)           Proxy property
 (8)           """
 (8)           return self._parent.sheet_properties.pageSetUpPr
 (4)       @property
 (4)       def fitToPage(self):
 (8)           return self.sheet_properties.fitToPage
 (4)       @fitToPage.setter
 (4)       def fitToPage(self, value):
 (8)           self.sheet_properties.fitToPage = value
 (4)       @property
 (4)       def autoPageBreaks(self):
 (8)           return self.sheet_properties.autoPageBreaks
 (4)       @autoPageBreaks.setter
 (4)       def autoPageBreaks(self, value):
 (8)           self.sheet_properties.autoPageBreaks = value
 (4)       @classmethod
 (4)       def from_tree(cls, node):
 (8)           self = super().from_tree(node)
 (8)           self.id = None # strip link to binary settings
 (8)           return self
 (0)   class PrintOptions(Serialisable):
 (4)       """ Worksheet print options """
 (4)       tagname = "printOptions"
 (4)       horizontalCentered = Bool(allow_none=True)
 (4)       verticalCentered = Bool(allow_none=True)
 (4)       headings = Bool(allow_none=True)
 (4)       gridLines = Bool(allow_none=True)
 (4)       gridLinesSet = Bool(allow_none=True)
 (4)       def __init__(self, horizontalCentered=None,
 (17)                   verticalCentered=None,
 (17)                   headings=None,
 (17)                   gridLines=None,
 (17)                   gridLinesSet=None,
 (17)                   ):
 (8)           self.horizontalCentered = horizontalCentered
 (8)           self.verticalCentered = verticalCentered
 (8)           self.headings = headings
 (8)           self.gridLines = gridLines
```

```
115: (8)                        self.gridLinesSet = gridLinesSet
116: (4)                    def __bool__(self):
117: (8)                        return bool(dict(self))
118: (0)                class PageMargins(Serialisable):
119: (4)                    """
120: (4)                    Information about page margins for view/print layouts.
121: (4)                    Standard values (in inches)
122: (4)                    left, right = 0.75
123: (4)                    top, bottom = 1
124: (4)                    header, footer = 0.5
125: (4)                    """
126: (4)                    tagname = "pageMargins"
127: (4)                    left = Float()
128: (4)                    right = Float()
129: (4)                    top = Float()
130: (4)                    bottom = Float()
131: (4)                    header = Float()
132: (4)                    footer = Float()
133: (4)                    def __init__(self, left=0.75, right=0.75, top=1, bottom=1, header=0.5,
134: (17)                                footer=0.5):
135: (8)                        self.left = left
136: (8)                        self.right = right
137: (8)                        self.top = top
138: (8)                        self.bottom = bottom
139: (8)                        self.header = header
140: (8)                        self.footer = footer

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

File 159 - merge.py:

1: (0)                  import copy
2: (0)                  from openpyxl.descriptors.serialisable import Serialisable
3: (0)                  from openpyxl.descriptors import (
4: (4)                      Integer,
5: (4)                      Sequence,
6: (0)                  )
7: (0)                  from openpyxl.cell.cell import MergedCell
8: (0)                  from openpyxl.styles.borders import Border
9: (0)                  from .cell_range import CellRange
10: (0)                 class MergeCell(CellRange):
11: (4)                     tagname = "mergeCell"
12: (4)                     ref = CellRange.coord
13: (4)                     __attrs__ = ("ref",)
14: (4)                     def __init__(self,
15: (17)                                 ref=None,
16: (16)                                ):
17: (8)                         super().__init__(ref)
18: (4)                     def __copy__(self):
19: (8)                         return self.__class__(self.ref)
20: (0)                 class MergeCells(Serialisable):
21: (4)                     tagname = "mergeCells"
22: (4)                     count = Integer(allow_none=True)
23: (4)                     mergeCell = Sequence(expected_type=MergeCell, )
24: (4)                     __elements__ = ('mergeCell',)
25: (4)                     __attrs__ = ('count',)
26: (4)                     def __init__(self,
27: (17)                                 count=None,
28: (17)                                 mergeCell=(),
29: (16)                                ):
30: (8)                         self.mergeCell = mergeCell
31: (4)                     @property
32: (4)                     def count(self):
33: (8)                         return len(self.mergeCell)
34: (0)                 class MergedCellRange(CellRange):
35: (4)                     """
36: (4)                     MergedCellRange stores the border information of a merged cell in the top
37: (4)                     left cell of the merged cell.
38: (4)                     The remaining cells in the merged cell are stored as MergedCell objects
```

```
and
    get their border information from the upper left cell.
    """
    def __init__(self, worksheet, coord):
        self.ws = worksheet
        super().__init__(range_string=coord)
        self.start_cell = None
        self._get_borders()
    def _get_borders(self):
        """
        If the upper left cell of the merged cell does not yet exist, it is
        created.
        The upper left cell gets the border information of the bottom and right
        border from the bottom right cell of the merged cell, if available.
        """
        self.start_cell = self.ws._cells.get((self.min_row, self.min_col))
        if self.start_cell is None:
            self.start_cell = self.ws.cell(row=self.min_row, column=self.min_col)
        end_cell = self.ws._cells.get((self.max_row, self.max_col))
        if end_cell is not None:
            self.start_cell.border += Border(right=end_cell.border.right,
                                             bottom=end_cell.border.bottom)
    def format(self):
        """
        Each cell of the merged cell is created as MergedCell if it does not
        already exist.
        The MergedCells at the edge of the merged cell gets its borders from
        the upper left cell.
         - The top MergedCells get the top border from the top left cell.
         - The bottom MergedCells get the bottom border from the top left cell.
         - The left MergedCells get the left border from the top left cell.
         - The right MergedCells get the right border from the top left cell.
        """
        names = ['top', 'left', 'right', 'bottom']
        for name in names:
            side = getattr(self.start_cell.border, name)
            if side and side.style is None:
                continue # don't need to do anything if there is no border style
            border = Border(**{name:side})
            for coord in getattr(self, name):
                cell = self.ws._cells.get(coord)
                if cell is None:
                    row, col = coord
                    cell = MergedCell(self.ws, row=row, column=col)
                    self.ws._cells[(cell.row, cell.column)] = cell
                cell.border += border
        protected = self.start_cell.protection is not None
        if protected:
            protection = copy.copy(self.start_cell.protection)
        for coord in self.cells:
            cell = self.ws._cells.get(coord)
            if cell is None:
                row, col = coord
                cell = MergedCell(self.ws, row=row, column=col)
                self.ws._cells[(cell.row, cell.column)] = cell
            if protected:
                cell.protection = protection
    def __contains__(self, coord):
        return coord in CellRange(self.coord)
    def __copy__(self):
        return self.__class__(self.ws, self.coord)
```

File 160 - copier.py:

```
(0)             from copy import copy
(0)             from .worksheet import Worksheet
(0)             class WorksheetCopy:
(4)                 """
(4)                 Copy the values, styles, dimensions, merged cells, margins, and
(4)                 print/page setup from one worksheet to another within the same
(4)                 workbook.
(4)                 """
(4)                 def __init__(self, source_worksheet, target_worksheet):
(8)                     self.source = source_worksheet
(8)                     self.target = target_worksheet
(8)                     self._verify_resources()
(4)                 def _verify_resources(self):
(8)                     if (not isinstance(self.source, Worksheet)
(12)                        and not isinstance(self.target, Worksheet)):
(12)                        raise TypeError("Can only copy worksheets")
(8)                     if self.source is self.target:
(12)                        raise ValueError("Cannot copy a worksheet to itself")
(8)                     if self.source.parent != self.target.parent:
(12)                        raise ValueError('Cannot copy between worksheets from different workbooks')
(4)                 def copy_worksheet(self):
(8)                     self._copy_cells()
(8)                     self._copy_dimensions()
(8)                     self.target.sheet_format = copy(self.source.sheet_format)
(8)                     self.target.sheet_properties = copy(self.source.sheet_properties)
(8)                     self.target.merged_cells = copy(self.source.merged_cells)
(8)                     self.target.page_margins = copy(self.source.page_margins)
(8)                     self.target.page_setup = copy(self.source.page_setup)
(8)                     self.target.print_options = copy(self.source.print_options)
(4)                 def _copy_cells(self):
(8)                     for (row, col), source_cell  in self.source._cells.items():
(12)                        target_cell = self.target.cell(column=col, row=row)
(12)                        target_cell._value = source_cell._value
(12)                        target_cell.data_type = source_cell.data_type
(12)                        if source_cell.has_style:
(16)                            target_cell._style = copy(source_cell._style)
(12)                        if source_cell.hyperlink:
(16)                            target_cell._hyperlink = copy(source_cell.hyperlink)
(12)                        if source_cell.comment:
(16)                            target_cell.comment = copy(source_cell.comment)
(4)                 def _copy_dimensions(self):
(8)                     for attr in ('row_dimensions', 'column_dimensions'):
(12)                        src = getattr(self.source, attr)
(12)                        target = getattr(self.target, attr)
(12)                        for key, dim in src.items():
(16)                            target[key] = copy(dim)
(16)                            target[key].worksheet = self.target
```

File 161 - custom.py:

```
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 String,
(4)                 Sequence,
(0)             )
(0)             class CustomProperty(Serialisable):
(4)                 tagname = "customProperty"
(4)                 name = String()
(4)                 def __init__(self,
(17)                             name=None,
(16)                            ):
(8)                     self.name = name
(0)             class CustomProperties(Serialisable):
(4)                 tagname = "customProperties"
(4)                 customPr = Sequence(expected_type=CustomProperty)
```

```
16: (4)                 __elements__ = ('customPr',)
17: (4)                 def __init__(self,
18: (17)                             customPr=(),
19: (16)                            ):
20: (8)                     self.customPr = customPr

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

File 162 - errors.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  String,
5: (4)                  Bool,
6: (4)                  Sequence,
7: (0)              )
8: (0)              from openpyxl.descriptors.excel import CellRange
9: (0)              class Extension(Serialisable):
10: (4)                 tagname = "extension"
11: (4)                 uri = String(allow_none=True)
12: (4)                 def __init__(self,
13: (17)                             uri=None,
14: (16)                            ):
15: (8)                     self.uri = uri
16: (0)             class ExtensionList(Serialisable):
17: (4)                 tagname = "extensionList"
18: (4)                 ext = Sequence(expected_type=Extension)
19: (4)                 __elements__ = ('ext',)
20: (4)                 def __init__(self,
21: (17)                             ext=(),
22: (16)                            ):
23: (8)                     self.ext = ext
24: (0)             class IgnoredError(Serialisable):
25: (4)                 tagname = "ignoredError"
26: (4)                 sqref = CellRange
27: (4)                 evalError = Bool(allow_none=True)
28: (4)                 twoDigitTextYear = Bool(allow_none=True)
29: (4)                 numberStoredAsText = Bool(allow_none=True)
30: (4)                 formula = Bool(allow_none=True)
31: (4)                 formulaRange = Bool(allow_none=True)
32: (4)                 unlockedFormula = Bool(allow_none=True)
33: (4)                 emptyCellReference = Bool(allow_none=True)
34: (4)                 listDataValidation = Bool(allow_none=True)
35: (4)                 calculatedColumn = Bool(allow_none=True)
36: (4)                 def __init__(self,
37: (17)                             sqref=None,
38: (17)                             evalError=False,
39: (17)                             twoDigitTextYear=False,
40: (17)                             numberStoredAsText=False,
41: (17)                             formula=False,
42: (17)                             formulaRange=False,
43: (17)                             unlockedFormula=False,
44: (17)                             emptyCellReference=False,
45: (17)                             listDataValidation=False,
46: (17)                             calculatedColumn=False,
47: (16)                            ):
48: (8)                     self.sqref = sqref
49: (8)                     self.evalError = evalError
50: (8)                     self.twoDigitTextYear = twoDigitTextYear
51: (8)                     self.numberStoredAsText = numberStoredAsText
52: (8)                     self.formula = formula
53: (8)                     self.formulaRange = formulaRange
54: (8)                     self.unlockedFormula = unlockedFormula
55: (8)                     self.emptyCellReference = emptyCellReference
56: (8)                     self.listDataValidation = listDataValidation
57: (8)                     self.calculatedColumn = calculatedColumn
58: (0)             class IgnoredErrors(Serialisable):
59: (4)                 tagname = "ignoredErrors"
```

```
60: (4)                 ignoredError = Sequence(expected_type=IgnoredError)
61: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
62: (4)                 __elements__ = ('ignoredError', 'extLst')
63: (4)                 def __init__(self,
64: (17)                             ignoredError=(),
65: (17)                             extLst=None,
66: (16)                            ):
67: (8)                     self.ignoredError = ignoredError
68: (8)                     self.extLst = extLst

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

File 163 - drawing.py:

1: (0)               from openpyxl.descriptors.serialisable import Serialisable
2: (0)               from openpyxl.descriptors.excel import Relation
3: (0)               class Drawing(Serialisable):
4: (4)                   tagname = "drawing"
5: (4)                   id = Relation()
6: (4)                   def __init__(self, id=None):
7: (8)                       self.id = id

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

File 164 - filters.py:

1: (0)               import re
2: (0)               from openpyxl.descriptors.serialisable import Serialisable
3: (0)               from openpyxl.descriptors import (
4: (4)                   Alias,
5: (4)                   Typed,
6: (4)                   Set,
7: (4)                   Float,
8: (4)                   DateTime,
9: (4)                   NoneSet,
10: (4)                  Bool,
11: (4)                  Integer,
12: (4)                  String,
13: (4)                  Sequence,
14: (4)                  MinMax,
15: (0)              )
16: (0)              from openpyxl.descriptors.excel import ExtensionList, CellRange
17: (0)              from openpyxl.descriptors.sequence import ValueSequence
18: (0)              from openpyxl.utils import absolute_coordinate
19: (0)              class SortCondition(Serialisable):
20: (4)                  tagname = "sortCondition"
21: (4)                  descending = Bool(allow_none=True)
22: (4)                  sortBy = NoneSet(values=(['value', 'cellColor', 'fontColor', 'icon']))
23: (4)                  ref = CellRange()
24: (4)                  customList = String(allow_none=True)
25: (4)                  dxfId = Integer(allow_none=True)
26: (4)                  iconSet = NoneSet(values=(['3Arrows', '3ArrowsGray', '3Flags',
27: (27)                                           '3TrafficLights1', '3TrafficLights2', '3Signs',
'3Symbols', '3Symbols2',
28: (27)                                           '4Arrows', '4ArrowsGray', '4RedToBlack', '4Rating',
'4TrafficLights',
29: (27)                                           '5Arrows', '5ArrowsGray', '5Rating', '5Quarters']))
30: (4)                  iconId = Integer(allow_none=True)
31: (4)                  def __init__(self,
32: (17)                              ref=None,
33: (17)                              descending=None,
34: (17)                              sortBy=None,
35: (17)                              customList=None,
36: (17)                              dxfId=None,
37: (17)                              iconSet=None,
38: (17)                              iconId=None,
39: (16)                             ):
40: (8)                      self.descending = descending
41: (8)                      self.sortBy = sortBy
```

```
42: (8)                     self.ref = ref
43: (8)                     self.customList = customList
44: (8)                     self.dxfId = dxfId
45: (8)                     self.iconSet = iconSet
46: (8)                     self.iconId = iconId
47: (0)             class SortState(Serialisable):
48: (4)                 tagname = "sortState"
49: (4)                 columnSort = Bool(allow_none=True)
50: (4)                 caseSensitive = Bool(allow_none=True)
51: (4)                 sortMethod = NoneSet(values=(['stroke', 'pinYin']))
52: (4)                 ref = CellRange()
53: (4)                 sortCondition = Sequence(expected_type=SortCondition, allow_none=True)
54: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
55: (4)                 __elements__ = ('sortCondition',)
56: (4)                 def __init__(self,
57: (17)                             columnSort=None,
58: (17)                             caseSensitive=None,
59: (17)                             sortMethod=None,
60: (17)                             ref=None,
61: (17)                             sortCondition=(),
62: (17)                             extLst=None,
63: (16)                            ):
64: (8)                     self.columnSort = columnSort
65: (8)                     self.caseSensitive = caseSensitive
66: (8)                     self.sortMethod = sortMethod
67: (8)                     self.ref = ref
68: (8)                     self.sortCondition = sortCondition
69: (4)                 def __bool__(self):
70: (8)                     return self.ref is not None
71: (0)             class IconFilter(Serialisable):
72: (4)                 tagname = "iconFilter"
73: (4)                 iconSet = Set(values=(['3Arrows', '3ArrowsGray', '3Flags',
74: (27)                                      '3TrafficLights1', '3TrafficLights2', '3Signs',
'3Symbols', '3Symbols2',
75: (27)                                      '4Arrows', '4ArrowsGray', '4RedToBlack', '4Rating',
'4TrafficLights',
76: (27)                                      '5Arrows', '5ArrowsGray', '5Rating', '5Quarters']))
77: (4)                 iconId = Integer(allow_none=True)
78: (4)                 def __init__(self,
79: (17)                             iconSet=None,
80: (17)                             iconId=None,
81: (16)                            ):
82: (8)                     self.iconSet = iconSet
83: (8)                     self.iconId = iconId
84: (0)             class ColorFilter(Serialisable):
85: (4)                 tagname = "colorFilter"
86: (4)                 dxfId = Integer(allow_none=True)
87: (4)                 cellColor = Bool(allow_none=True)
88: (4)                 def __init__(self,
89: (17)                             dxfId=None,
90: (17)                             cellColor=None,
91: (16)                            ):
92: (8)                     self.dxfId = dxfId
93: (8)                     self.cellColor = cellColor
94: (0)             class DynamicFilter(Serialisable):
95: (4)                 tagname = "dynamicFilter"
96: (4)                 type = Set(values=(['null', 'aboveAverage', 'belowAverage', 'tomorrow',
97: (24)                                   'today', 'yesterday', 'nextWeek', 'thisWeek',
'lastWeek', 'nextMonth',
98: (24)                                   'thisMonth', 'lastMonth', 'nextQuarter',
'thisQuarter', 'lastQuarter',
99: (24)                                   'nextYear', 'thisYear', 'lastYear', 'yearToDate',
'Q1', 'Q2', 'Q3', 'Q4',
100: (24)                                  'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9',
'M10', 'M11',
101: (24)                                  'M12']))
102: (4)                val = Float(allow_none=True)
103: (4)                valIso = DateTime(allow_none=True)
104: (4)                maxVal = Float(allow_none=True)
```

```
(4)          maxValIso = DateTime(allow_none=True)
(4)          def __init__(self,
(17)                      type=None,
(17)                      val=None,
(17)                      valIso=None,
(17)                      maxVal=None,
(17)                      maxValIso=None,
(16)                     ):
(8)              self.type = type
(8)              self.val = val
(8)              self.valIso = valIso
(8)              self.maxVal = maxVal
(8)              self.maxValIso = maxValIso
(0)      class CustomFilter(Serialisable):
(4)          tagname = "customFilter"
(4)          val = String()
(4)          operator = Set(values=['equal', 'lessThan', 'lessThanOrEqual',
(27)                                'notEqual', 'greaterThanOrEqual', 'greaterThan'])
(4)          def __init__(self, operator="equal", val=None):
(8)              self.operator = operator
(8)              self.val = val
(4)          def _get_subtype(self):
(8)              if self.val == " ":
(12)                 subtype = BlankFilter
(8)              else:
(12)                 try:
(16)                     float(self.val)
(16)                     subtype = NumberFilter
(12)                 except ValueError:
(16)                     subtype = StringFilter
(8)              return subtype
(4)          def convert(self):
(8)              """Convert to more specific filter"""
(8)              typ = self._get_subtype()
(8)              if typ in (BlankFilter, NumberFilter):
(12)                 return typ(**dict(self))
(8)              operator, term = StringFilter._guess_operator(self.val)
(8)              flt = StringFilter(operator, term)
(8)              if self.operator == "notEqual":
(12)                 flt.exclude = True
(8)              return flt
(0)      class BlankFilter(CustomFilter):
(4)          """
(4)          Exclude blanks
(4)          """
(4)          __attrs__ = ("operator", "val")
(4)          def __init__(self, **kw):
(8)              pass
(4)          @property
(4)          def operator(self):
(8)              return "notEqual"
(4)          @property
(4)          def val(self):
(8)              return " "
(0)      class NumberFilter(CustomFilter):
(4)          operator = Set(values=
(19)                        ['equal', 'lessThan', 'lessThanOrEqual',
(20)                         'notEqual', 'greaterThanOrEqual', 'greaterThan'])
(4)          val = Float()
(4)          def __init__(self, operator="equal", val=None):
(8)              self.operator = operator
(8)              self.val = val
(0)      string_format_mapping = {
(4)          "contains": "*{}*",
(4)          "startswith": "{}*",
(4)          "endswith": "*{}",
(4)          "wildcard":  "{}",
(0)      }
(0)      class StringFilter(CustomFilter):
    operator = Set(values=['contains', 'startswith', 'endswith', 'wildcard']
                   )
    val = String()
    exclude = Bool()
    def __init__(self, operator="contains", val=None, exclude=False):
        self.operator = operator
        self.val = val
        self.exclude = exclude
    def _escape(self):
        """Escape wildcards ~, * ? when serialising"""
        if self.operator == "wildcard":
            return self.val
        return re.sub(r"~|\*|\?", r"~\g<0>", self.val)
    @staticmethod
    def _unescape(value):
        """
        Unescape value
        """
        return re.sub(r"~(?P<op>[~*?])", r"\g<op>", value)
    @staticmethod
    def _guess_operator(value):
        value = StringFilter._unescape(value)
        endswith = r"^(?P<endswith>\*)(?P<term>[^\*\?]*$)"
        startswith = r"^(?P<term>[^\*\?]*)(?P<startswith>\*)$"
        contains = r"^(?P<contains>\*)(?P<term>[^\*\?]*)\*$"
        d = {"wildcard": True, "term": value}
        for pat in [contains, startswith, endswith]:
            m = re.match(pat, value)
            if m:
                d = m.groupdict()
        term = d.pop("term")
        op = list(d)[0]
        return op, term
    def to_tree(self, tagname=None, idx=None, namespace=None):
        fmt = string_format_mapping[self.operator]
        op = self.exclude and "notEqual" or "equal"
        value = fmt.format(self._escape())
        flt = CustomFilter(op, value)
        return flt.to_tree(tagname, idx, namespace)
class CustomFilters(Serialisable):
    tagname = "customFilters"
    _and = Bool(allow_none=True)
    customFilter = Sequence(expected_type=CustomFilter) # min 1, max 2
    __elements__ = ('customFilter',)
    def __init__(self,
                 _and=None,
                 customFilter=(),
                ):
        self._and = _and
        self.customFilter = customFilter
class Top10(Serialisable):
    tagname = "top10"
    top = Bool(allow_none=True)
    percent = Bool(allow_none=True)
    val = Float()
    filterVal = Float(allow_none=True)
    def __init__(self,
                 top=None,
                 percent=None,
                 val=None,
                 filterVal=None,
                ):
        self.top = top
        self.percent = percent
        self.val = val
        self.filterVal = filterVal
class DateGroupItem(Serialisable):
    tagname = "dateGroupItem"
    year = Integer()
```

```
243: (4)                month = MinMax(min=1, max=12, allow_none=True)
244: (4)                day = MinMax(min=1, max=31, allow_none=True)
245: (4)                hour = MinMax(min=0, max=23, allow_none=True)
246: (4)                minute = MinMax(min=0, max=59, allow_none=True)
247: (4)                second = Integer(min=0, max=59, allow_none=True)
248: (4)                dateTimeGrouping = Set(values=(['year', 'month', 'day', 'hour', 'minute',
249: (36)                                               'second']))
250: (4)                def __init__(self,
251: (17)                             year=None,
252: (17)                             month=None,
253: (17)                             day=None,
254: (17)                             hour=None,
255: (17)                             minute=None,
256: (17)                             second=None,
257: (17)                             dateTimeGrouping=None,
258: (16)                            ):
259: (8)                    self.year = year
260: (8)                    self.month = month
261: (8)                    self.day = day
262: (8)                    self.hour = hour
263: (8)                    self.minute = minute
264: (8)                    self.second = second
265: (8)                    self.dateTimeGrouping = dateTimeGrouping
266: (0)            class Filters(Serialisable):
267: (4)                tagname = "filters"
268: (4)                blank = Bool(allow_none=True)
269: (4)                calendarType = NoneSet(values=["gregorian","gregorianUs",
270: (35)                                             "gregorianMeFrench","gregorianArabic",
"hijri","hebrew",
271: (35)                                             "taiwan","japan", "thai","korea",
272: (35)
"saka","gregorianXlitEnglish","gregorianXlitFrench"])
273: (4)                filter = ValueSequence(expected_type=str)
274: (4)                dateGroupItem = Sequence(expected_type=DateGroupItem, allow_none=True)
275: (4)                __elements__ = ('filter', 'dateGroupItem')
276: (4)                def __init__(self,
277: (17)                             blank=None,
278: (17)                             calendarType=None,
279: (17)                             filter=(),
280: (17)                             dateGroupItem=(),
281: (16)                            ):
282: (8)                    self.blank = blank
283: (8)                    self.calendarType = calendarType
284: (8)                    self.filter = filter
285: (8)                    self.dateGroupItem = dateGroupItem
286: (0)            class FilterColumn(Serialisable):
287: (4)                tagname = "filterColumn"
288: (4)                colId = Integer()
289: (4)                col_id = Alias('colId')
290: (4)                hiddenButton = Bool(allow_none=True)
291: (4)                showButton = Bool(allow_none=True)
292: (4)                filters = Typed(expected_type=Filters, allow_none=True)
293: (4)                top10 = Typed(expected_type=Top10, allow_none=True)
294: (4)                customFilters = Typed(expected_type=CustomFilters, allow_none=True)
295: (4)                dynamicFilter = Typed(expected_type=DynamicFilter, allow_none=True)
296: (4)                colorFilter = Typed(expected_type=ColorFilter, allow_none=True)
297: (4)                iconFilter = Typed(expected_type=IconFilter, allow_none=True)
298: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
299: (4)                __elements__ = ('filters', 'top10', 'customFilters', 'dynamicFilter',
300: (20)                                'colorFilter', 'iconFilter')
301: (4)                def __init__(self,
302: (17)                             colId=None,
303: (17)                             hiddenButton=False,
304: (17)                             showButton=True,
305: (17)                             filters=None,
306: (17)                             top10=None,
307: (17)                             customFilters=None,
308: (17)                             dynamicFilter=None,
309: (17)                             colorFilter=None,
310: (17)                          iconFilter=None,
311: (17)                          extLst=None,
312: (17)                          blank=None,
313: (17)                          vals=None,
314: (16)                         ):
315: (8)                  self.colId = colId
316: (8)                  self.hiddenButton = hiddenButton
317: (8)                  self.showButton = showButton
318: (8)                  self.filters = filters
319: (8)                  self.top10 = top10
320: (8)                  self.customFilters = customFilters
321: (8)                  self.dynamicFilter = dynamicFilter
322: (8)                  self.colorFilter = colorFilter
323: (8)                  self.iconFilter = iconFilter
324: (8)                  if blank is not None and self.filters:
325: (12)                     self.filters.blank = blank
326: (8)                  if vals is not None and self.filters:
327: (12)                     self.filters.filter = vals
328: (0)          class AutoFilter(Serialisable):
329: (4)              tagname = "autoFilter"
330: (4)              ref = CellRange()
331: (4)              filterColumn = Sequence(expected_type=FilterColumn, allow_none=True)
332: (4)              sortState = Typed(expected_type=SortState, allow_none=True)
333: (4)              extLst = Typed(expected_type=ExtensionList, allow_none=True)
334: (4)              __elements__ = ('filterColumn', 'sortState')
335: (4)              def __init__(self,
336: (17)                          ref=None,
337: (17)                          filterColumn=(),
338: (17)                          sortState=None,
339: (17)                          extLst=None,
340: (16)                         ):
341: (8)                  self.ref = ref
342: (8)                  self.filterColumn = filterColumn
343: (8)                  self.sortState = sortState
344: (4)              def __bool__(self):
345: (8)                  return self.ref is not None
346: (4)              def __str__(self):
347: (8)                  return absolute_coordinate(self.ref)
348: (4)              def add_filter_column(self, col_id, vals, blank=False):
349: (8)                  """
350: (8)                  Add row filter for specified column.
351: (8)                  :param col_id: Zero-origin column id. 0 means first column.
352: (8)                  :type  col_id: int
353: (8)                  :param vals: Value list to show.
354: (8)                  :type  vals: str[]
355: (8)                  :param blank: Show rows that have blank cell if True
(default=``False``)
356: (8)                  :type  blank: bool
357: (8)                  """
358: (8)                  self.filterColumn.append(FilterColumn(colId=col_id,
filters=Filters(blank=blank, filter=vals)))
359: (4)              def add_sort_condition(self, ref, descending=False):
360: (8)                  """
361: (8)                  Add sort condition for cpecified range of cells.
362: (8)                  :param ref: range of the cells (e.g. 'A2:A150')
363: (8)                  :type  ref: string, is the same as that of the filter
364: (8)                  :param descending: Descending sort order (default=``False``)
365: (8)                  :type  descending: bool
366: (8)                  """
367: (8)                  cond = SortCondition(ref, descending)
368: (8)                  if self.sortState is None:
369: (12)                     self.sortState = SortState(ref=self.ref)
370: (8)                  self.sortState.sortCondition.append(cond)
```

File 165 - formula.py:

```
1: (0)          from openpyxl.compat import safe_string
```

```
2: (0)              class DataTableFormula:
3: (4)                  t = "dataTable"
4: (4)                  def __init__(self,
5: (17)                              ref,
6: (17)                              ca=False,
7: (17)                              dt2D=False,
8: (17)                              dtr=False,
9: (17)                              r1=None,
10: (17)                             r2=None,
11: (17)                             del1=False,
12: (17)                             del2=False,
13: (17)                             **kw):
14: (8)                     self.ref = ref
15: (8)                     self.ca = ca
16: (8)                     self.dt2D = dt2D
17: (8)                     self.dtr = dtr
18: (8)                     self.r1 = r1
19: (8)                     self.r2 = r2
20: (8)                     self.del1 = del1
21: (8)                     self.del2 = del2
22: (4)                 def __iter__(self):
23: (8)                     for k in ["t", "ref", "dt2D", "dtr", "r1", "r2", "del1", "del2",
"ca"]:
24: (12)                        v = getattr(self, k)
25: (12)                        if v:
26: (16)                            yield k, safe_string(v)
27: (0)             class ArrayFormula:
28: (4)                 t = "array"
29: (4)                 def __init__(self, ref, text=None):
30: (8)                     self.ref = ref
31: (8)                     self.text = text
32: (4)                 def __iter__(self):
33: (8)                     for k in ["t", "ref"]:
34: (12)                        v = getattr(self, k)
35: (12)                        if v:
36: (16)                            yield k, safe_string(v)

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

File 166 - picture.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              class SheetBackgroundPicture(Serialisable):
3: (4)                  tagname = "sheetBackgroundPicture"

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

File 167 - _writer.py:

1: (0)              import atexit
2: (0)              from collections import defaultdict
3: (0)              from io import BytesIO
4: (0)              import os
5: (0)              from tempfile import NamedTemporaryFile
6: (0)              from warnings import warn
7: (0)              from openpyxl.xml.functions import xmlfile
8: (0)              from openpyxl.xml.constants import SHEET_MAIN_NS
9: (0)              from openpyxl.comments.comment_sheet import CommentRecord
10: (0)             from openpyxl.packaging.relationship import Relationship, RelationshipList
11: (0)             from openpyxl.styles.differential import DifferentialStyle
12: (0)             from .dimensions import SheetDimension
13: (0)             from .hyperlink import HyperlinkList
14: (0)             from .merge import MergeCell, MergeCells
15: (0)             from .related import Related
16: (0)             from .table import TablePartList
17: (0)             from openpyxl.cell._writer import write_cell
18: (0)             ALL_TEMP_FILES = []
19: (0)             @atexit.register
20: (0)             def _openpyxl_shutdown():
```

```
(4)             for path in ALL_TEMP_FILES:
(8)                 if os.path.exists(path):
(12)                    os.remove(path)
(0)         def create_temporary_file(suffix=''):
(4)             fobj = NamedTemporaryFile(mode='w+', suffix=suffix,
(30)                                      prefix='openpyxl.', delete=False)
(4)             filename = fobj.name
(4)             fobj.close()
(4)             ALL_TEMP_FILES.append(filename)
(4)             return filename
(0)         class WorksheetWriter:
(4)             def __init__(self, ws, out=None):
(8)                 self.ws = ws
(8)                 self.ws._hyperlinks = []
(8)                 self.ws._comments = []
(8)                 if out is None:
(12)                    out = create_temporary_file()
(8)                 self.out = out
(8)                 self._rels = RelationshipList()
(8)                 self.xf = self.get_stream()
(8)                 next(self.xf) # start generator
(4)             def write_properties(self):
(8)                 props = self.ws.sheet_properties
(8)                 self.xf.send(props.to_tree())
(4)             def write_dimensions(self):
(8)                 """
(8)                 Write worksheet size if known
(8)                 """
(8)                 ref = getattr(self.ws, 'calculate_dimension', None)
(8)                 if ref:
(12)                    dim = SheetDimension(ref())
(12)                    self.xf.send(dim.to_tree())
(4)             def write_format(self):
(8)                 self.ws.sheet_format.outlineLevelCol = 
self.ws.column_dimensions.max_outline
(8)                 fmt = self.ws.sheet_format
(8)                 self.xf.send(fmt.to_tree())
(4)             def write_views(self):
(8)                 views = self.ws.views
(8)                 self.xf.send(views.to_tree())
(4)             def write_cols(self):
(8)                 cols = self.ws.column_dimensions
(8)                 self.xf.send(cols.to_tree())
(4)             def write_top(self):
(8)                 """
(8)                 Write all elements up to rows:
(8)                 properties
(8)                 dimensions
(8)                 views
(8)                 format
(8)                 cols
(8)                 """
(8)                 self.write_properties()
(8)                 self.write_dimensions()
(8)                 self.write_views()
(8)                 self.write_format()
(8)                 self.write_cols()
(4)             def rows(self):
(8)                 """Return all rows, and any cells that they contain"""
(8)                 rows = defaultdict(list)
(8)                 for (row, col), cell in sorted(self.ws._cells.items()):
(12)                    rows[row].append(cell)
(8)                 for row in self.ws.row_dimensions.keys() - rows.keys():
(12)                    rows[row] = []
(8)                 return sorted(rows.items())
(4)             def write_rows(self):
(8)                 xf = self.xf.send(True)
(8)                 with xf.element("sheetData"):
(12)                    for row_idx, row in self.rows():
```

```
89: (16)                            self.write_row(xf, row, row_idx)
90: (8)                     self.xf.send(None) # return control to generator
91: (4)                 def write_row(self, xf, row, row_idx):
92: (8)                     attrs = {'r': f"{row_idx}"}
93: (8)                     dims = self.ws.row_dimensions
94: (8)                     attrs.update(dims.get(row_idx, {}))
95: (8)                     with xf.element("row", attrs):
96: (12)                        for cell in row:
97: (16)                            if cell._comment is not None:
98: (20)                                comment = CommentRecord.from_cell(cell)
99: (20)                                self.ws._comments.append(comment)
100: (16)                           if (
101: (20)                               cell._value is None
102: (20)                               and not cell.has_style
103: (20)                               and not cell._comment
104: (20)                               ):
105: (20)                               continue
106: (16)                           write_cell(xf, self.ws, cell, cell.has_style)
107: (4)                def write_protection(self):
108: (8)                    prot = self.ws.protection
109: (8)                    if prot:
110: (12)                       self.xf.send(prot.to_tree())
111: (4)                def write_scenarios(self):
112: (8)                    scenarios = self.ws.scenarios
113: (8)                    if scenarios:
114: (12)                       self.xf.send(scenarios.to_tree())
115: (4)                def write_filter(self):
116: (8)                    flt = self.ws.auto_filter
117: (8)                    if flt:
118: (12)                       self.xf.send(flt.to_tree())
119: (4)                def write_sort(self):
120: (8)                    """
121: (8)                    As per discusion with the OOXML Working Group global sort state is not
required.
122: (8)                    openpyxl never reads it from existing files
123: (8)                    """
124: (8)                    pass
125: (4)                def write_merged_cells(self):
126: (8)                    merged = self.ws.merged_cells
127: (8)                    if merged:
128: (12)                       cells = [MergeCell(str(ref)) for ref in self.ws.merged_cells]
129: (12)                       self.xf.send(MergeCells(mergeCell=cells).to_tree())
130: (4)                def write_formatting(self):
131: (8)                    df = DifferentialStyle()
132: (8)                    wb = self.ws.parent
133: (8)                    for cf in self.ws.conditional_formatting:
134: (12)                       for rule in cf.rules:
135: (16)                           if rule.dxf and rule.dxf != df:
136: (20)                               rule.dxfId = wb._differential_styles.add(rule.dxf)
137: (12)                       self.xf.send(cf.to_tree())
138: (4)                def write_validations(self):
139: (8)                    dv = self.ws.data_validations
140: (8)                    if dv:
141: (12)                       self.xf.send(dv.to_tree())
142: (4)                def write_hyperlinks(self):
143: (8)                    links = self.ws._hyperlinks
144: (8)                    for link in links:
145: (12)                       if link.target:
146: (16)                           rel = Relationship(type="hyperlink", TargetMode="External",
Target=link.target)
147: (16)                           self._rels.append(rel)
148: (16)                           link.id = rel.id
149: (8)                    if links:
150: (12)                       self.xf.send(HyperlinkList(links).to_tree())
151: (4)                def write_print(self):
152: (8)                    print_options = self.ws.print_options
153: (8)                    if print_options:
154: (12)                       self.xf.send(print_options.to_tree())
155: (4)                def write_margins(self):
```

```
(8)                 margins = self.ws.page_margins
(8)                 if margins:
(12)                    self.xf.send(margins.to_tree())
(4)             def write_page(self):
(8)                 setup = self.ws.page_setup
(8)                 if setup:
(12)                    self.xf.send(setup.to_tree())
(4)             def write_header(self):
(8)                 hf = self.ws.HeaderFooter
(8)                 if hf:
(12)                    self.xf.send(hf.to_tree())
(4)             def write_breaks(self):
(8)                 brks = (self.ws.row_breaks, self.ws.col_breaks)
(8)                 for brk in brks:
(12)                    if brk:
(16)                        self.xf.send(brk.to_tree())
(4)             def write_drawings(self):
(8)                 if self.ws._charts or self.ws._images:
(12)                    rel = Relationship(type="drawing", Target="")
(12)                    self._rels.append(rel)
(12)                    drawing = Related()
(12)                    drawing.id = rel.id
(12)                    self.xf.send(drawing.to_tree("drawing"))
(4)             def write_legacy(self):
(8)                 """
(8)                 Comments & VBA controls use VML and require an additional element
(8)                 that is no longer in the specification.
(8)                 """
(8)                 if (self.ws.legacy_drawing is not None or self.ws._comments):
(12)                    legacy = Related(id="anysvml")
(12)                    self.xf.send(legacy.to_tree("legacyDrawing"))
(4)             def write_tables(self):
(8)                 tables = TablePartList()
(8)                 for table in self.ws.tables.values():
(12)                    if not table.tableColumns:
(16)                        table._initialise_columns()
(16)                        if table.headerRowCount:
(20)                            try:
(24)                                row = self.ws[table.ref][0]
(24)                                for cell, col in zip(row, table.tableColumns):
(28)                                    if cell.data_type != "s":
(32)                                        warn("File may not be readable: column headings must be strings.")
(28)                                    col.name = str(cell.value)
(20)                            except TypeError:
(24)                                warn("Column headings are missing, file may not be readable")
(12)                    rel = Relationship(Type=table._rel_type, Target="")
(12)                    self._rels.append(rel)
(12)                    table._rel_id = rel.Id
(12)                    tables.append(Related(id=rel.Id))
(8)                 if tables:
(12)                    self.xf.send(tables.to_tree())
(4)             def get_stream(self):
(8)                 with xmlfile(self.out) as xf:
(12)                    with xf.element("worksheet", xmlns=SHEET_MAIN_NS):
(16)                        try:
(20)                            while True:
(24)                                el = (yield)
(24)                                if el is True:
(28)                                    yield xf
(24)                                elif el is None: # et_xmlfile chokes
(28)                                    continue
(24)                                else:
(28)                                    xf.write(el)
(16)                        except GeneratorExit:
(20)                            pass
(4)             def write_tail(self):
(8)                 """
```

```
(8)                         Write all elements after the rows
(8)                         calc properties
(8)                         protection
(8)                         protected ranges #
(8)                         scenarios
(8)                         filters
(8)                         sorts # always ignored
(8)                         data consolidation #
(8)                         custom views #
(8)                         merged cells
(8)                         phonetic properties #
(8)                         conditional formatting
(8)                         data validation
(8)                         hyperlinks
(8)                         print options
(8)                         page margins
(8)                         page setup
(8)                         header
(8)                         row breaks
(8)                         col breaks
(8)                         custom properties #
(8)                         cell watches #
(8)                         ignored errors #
(8)                         smart tags #
(8)                         drawing
(8)                         drawingHF #
(8)                         background #
(8)                         OLE objects #
(8)                         controls #
(8)                         web publishing #
(8)                         tables
(8)                         """
(8)                         self.write_protection()
(8)                         self.write_scenarios()
(8)                         self.write_filter()
(8)                         self.write_merged_cells()
(8)                         self.write_formatting()
(8)                         self.write_validations()
(8)                         self.write_hyperlinks()
(8)                         self.write_print()
(8)                         self.write_margins()
(8)                         self.write_page()
(8)                         self.write_header()
(8)                         self.write_breaks()
(8)                         self.write_drawings()
(8)                         self.write_legacy()
(8)                         self.write_tables()
(4)                     def write(self):
(8)                         """
(8)                         High level
(8)                         """
(8)                         self.write_top()
(8)                         self.write_rows()
(8)                         self.write_tail()
(8)                         self.close()
(4)                     def close(self):
(8)                         """
(8)                         Close the context manager
(8)                         """
(8)                         if self.xf:
(12)                            self.xf.close()
(4)                     def read(self):
(8)                         """
(8)                         Close the context manager and return serialised XML
(8)                         """
(8)                         self.close()
(8)                         if isinstance(self.out, BytesIO):
(12)                            return self.out.getvalue()
(8)                         with open(self.out, "rb") as src:
```

```
292: (12)                       out = src.read()
293: (8)                    return out
294: (4)                def cleanup(self):
295: (8)                    """
296: (8)                    Remove tempfile
297: (8)                    """
298: (8)                    os.remove(self.out)
299: (8)                    ALL_TEMP_FILES.remove(self.out)
```

File 168 - controls.py:

```
1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Typed,
4: (4)                  Bool,
5: (4)                  Integer,
6: (4)                  String,
7: (4)                  Sequence,
8: (0)              )
9: (0)              from openpyxl.descriptors.excel import Relation
10: (0)             from .ole import ObjectAnchor
11: (0)             class ControlProperty(Serialisable):
12: (4)                 tagname = "controlPr"
13: (4)                 anchor = Typed(expected_type=ObjectAnchor, )
14: (4)                 locked = Bool(allow_none=True)
15: (4)                 defaultSize = Bool(allow_none=True)
16: (4)                 _print = Bool(allow_none=True)
17: (4)                 disabled = Bool(allow_none=True)
18: (4)                 recalcAlways = Bool(allow_none=True)
19: (4)                 uiObject = Bool(allow_none=True)
20: (4)                 autoFill = Bool(allow_none=True)
21: (4)                 autoLine = Bool(allow_none=True)
22: (4)                 autoPict = Bool(allow_none=True)
23: (4)                 macro = String(allow_none=True)
24: (4)                 altText = String(allow_none=True)
25: (4)                 linkedCell = String(allow_none=True)
26: (4)                 listFillRange = String(allow_none=True)
27: (4)                 cf = String(allow_none=True)
28: (4)                 id = Relation(allow_none=True)
29: (4)                 __elements__ = ('anchor',)
30: (4)                 def __init__(self,
31: (17)                             anchor=None,
32: (17)                             locked=True,
33: (17)                             defaultSize=True,
34: (17)                             _print=True,
35: (17)                             disabled=False,
36: (17)                             recalcAlways=False,
37: (17)                             uiObject=False,
38: (17)                             autoFill=True,
39: (17)                             autoLine=True,
40: (17)                             autoPict=True,
41: (17)                             macro=None,
42: (17)                             altText=None,
43: (17)                             linkedCell=None,
44: (17)                             listFillRange=None,
45: (17)                             cf='pict',
46: (17)                             id=None,
47: (16)                            ):
48: (8)                     self.anchor = anchor
49: (8)                     self.locked = locked
50: (8)                     self.defaultSize = defaultSize
51: (8)                     self._print = _print
52: (8)                     self.disabled = disabled
53: (8)                     self.recalcAlways = recalcAlways
54: (8)                     self.uiObject = uiObject
55: (8)                     self.autoFill = autoFill
56: (8)                     self.autoLine = autoLine
```

```
(8)                     self.autoPict = autoPict
(8)                     self.macro = macro
(8)                     self.altText = altText
(8)                     self.linkedCell = linkedCell
(8)                     self.listFillRange = listFillRange
(8)                     self.cf = cf
(8)                     self.id = id
(0)             class Control(Serialisable):
(4)                 tagname = "control"
(4)                 controlPr = Typed(expected_type=ControlProperty, allow_none=True)
(4)                 shapeId = Integer()
(4)                 name = String(allow_none=True)
(4)                 __elements__ = ('controlPr',)
(4)                 def __init__(self,
(17)                             controlPr=None,
(17)                             shapeId=None,
(17)                             name=None,
(16)                            ):
(8)                     self.controlPr = controlPr
(8)                     self.shapeId = shapeId
(8)                     self.name = name
(0)             class Controls(Serialisable):
(4)                 tagname = "controls"
(4)                 control = Sequence(expected_type=Control)
(4)                 __elements__ = ('control',)
(4)                 def __init__(self,
(17)                             control=(),
(16)                            ):
(8)                     self.control = control
```

File 169 - hyperlink.py:

```
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 String,
(4)                 Sequence,
(0)             )
(0)             from openpyxl.descriptors.excel import Relation
(0)             class Hyperlink(Serialisable):
(4)                 tagname = "hyperlink"
(4)                 ref = String()
(4)                 location = String(allow_none=True)
(4)                 tooltip = String(allow_none=True)
(4)                 display = String(allow_none=True)
(4)                 id = Relation()
(4)                 target = String(allow_none=True)
(4)                 __attrs__ = ("ref", "location", "tooltip", "display", "id")
(4)                 def __init__(self,
(17)                             ref=None,
(17)                             location=None,
(17)                             tooltip=None,
(17)                             display=None,
(17)                             id=None,
(17)                             target=None,
(16)                            ):
(8)                     self.ref = ref
(8)                     self.location = location
(8)                     self.tooltip = tooltip
(8)                     self.display = display
(8)                     self.id = id
(8)                     self.target = target
(0)             class HyperlinkList(Serialisable):
(4)                 tagname = "hyperlinks"
(4)                 __expected_type = Hyperlink
(4)                 hyperlink = Sequence(expected_type=__expected_type)
(4)                 def __init__(self, hyperlink=()):
(8)                     self.hyperlink = hyperlink
```

```
----------------------------------------

File 170 - pagebreak.py:

(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             from openpyxl.descriptors import (
(4)                 Integer,
(4)                 Bool,
(4)                 Sequence,
(0)             )
(0)             class Break(Serialisable):
(4)                 tagname = "brk"
(4)                 id = Integer(allow_none=True)
(4)                 min = Integer(allow_none=True)
(4)                 max = Integer(allow_none=True)
(4)                 man = Bool(allow_none=True)
(4)                 pt = Bool(allow_none=True)
(4)                 def __init__(self,
(17)                             id=0,
(17)                             min=0,
(17)                             max=16383,
(17)                             man=True,
(17)                             pt=None,
(16)                            ):
(8)                     self.id = id
(8)                     self.min = min
(8)                     self.max = max
(8)                     self.man = man
(8)                     self.pt = pt
(0)             class RowBreak(Serialisable):
(4)                 tagname = "rowBreaks"
(4)                 count = Integer(allow_none=True)
(4)                 manualBreakCount = Integer(allow_none=True)
(4)                 brk = Sequence(expected_type=Break, allow_none=True)
(4)                 __elements__ = ('brk',)
(4)                 __attrs__ = ("count", "manualBreakCount",)
(4)                 def __init__(self,
(17)                             count=None,
(17)                             manualBreakCount=None,
(17)                             brk=(),
(16)                            ):
(8)                     self.brk = brk
(4)                 def __bool__(self):
(8)                     return len(self.brk) > 0
(4)                 def __len__(self):
(8)                     return len(self.brk)
(4)                 @property
(4)                 def count(self):
(8)                     return len(self)
(4)                 @property
(4)                 def manualBreakCount(self):
(8)                     return len(self)
(4)                 def append(self, brk=None):
(8)                     """
(8)                     Add a page break
(8)                     """
(8)                     vals = list(self.brk)
(8)                     if not isinstance(brk, Break):
(12)                        brk = Break(id=self.count+1)
(8)                     vals.append(brk)
(8)                     self.brk = vals
(0)             PageBreak = RowBreak
(0)             class ColBreak(RowBreak):
(4)                 tagname = "colBreaks"
(4)                 count = RowBreak.count
(4)                 manualBreakCount = RowBreak.manualBreakCount
(4)                 brk = RowBreak.brk
(4)                 __attrs__ = RowBreak.__attrs__
```

```
----------------------------------------

File 171 - cell_range.py:

1: (0)              from copy import copy
2: (0)              from operator import attrgetter
3: (0)              from openpyxl.descriptors import Strict
4: (0)              from openpyxl.descriptors import MinMax
5: (0)              from openpyxl.descriptors.sequence import UniqueSequence
6: (0)              from openpyxl.descriptors.serialisable import Serialisable
7: (0)              from openpyxl.utils import (
8: (4)                  range_boundaries,
9: (4)                  range_to_tuple,
10: (4)                 get_column_letter,
11: (4)                 quote_sheetname,
12: (0)             )
13: (0)             class CellRange(Serialisable):
14: (4)                 """
15: (4)                 Represents a range in a sheet: title and coordinates.
16: (4)                 This object is used to perform operations on ranges, like:
17: (4)                 - shift, expand or shrink
18: (4)                 - union/intersection with another sheet range,
19: (4)                 We can check whether a range is:
20: (4)                 - equal or not equal to another,
21: (4)                 - disjoint of another,
22: (4)                 - contained in another.
23: (4)                 We can get:
24: (4)                 - the size of a range.
25: (4)                 - the range bounds (vertices)
26: (4)                 - the coordinates,
27: (4)                 - the string representation,
28: (4)                 """
29: (4)                 min_col = MinMax(min=1, max=18278, expected_type=int)
30: (4)                 min_row = MinMax(min=1, max=1048576, expected_type=int)
31: (4)                 max_col = MinMax(min=1, max=18278, expected_type=int)
32: (4)                 max_row = MinMax(min=1, max=1048576, expected_type=int)
33: (4)                 def __init__(self, range_string=None, min_col=None, min_row=None,
34: (17)                             max_col=None, max_row=None, title=None):
35: (8)                     if range_string is not None:
36: (12)                        if "!" in range_string:
37: (16)                            title, (min_col, min_row, max_col, max_row) =
range_to_tuple(range_string)
38: (12)                        else:
39: (16)                            min_col, min_row, max_col, max_row =
range_boundaries(range_string)
40: (8)                     self.min_col = min_col
41: (8)                     self.min_row = min_row
42: (8)                     self.max_col = max_col
43: (8)                     self.max_row = max_row
44: (8)                     self.title = title
45: (8)                     if min_col > max_col:
46: (12)                        fmt = "{max_col} must be greater than {min_col}"
47: (12)                        raise ValueError(fmt.format(min_col=min_col, max_col=max_col))
48: (8)                     if min_row > max_row:
49: (12)                        fmt = "{max_row} must be greater than {min_row}"
50: (12)                        raise ValueError(fmt.format(min_row=min_row, max_row=max_row))
51: (4)                 @property
52: (4)                 def bounds(self):
53: (8)                     """
54: (8)                     Vertices of the range as a tuple
55: (8)                     """
56: (8)                     return self.min_col, self.min_row, self.max_col, self.max_row
57: (4)                 @property
58: (4)                 def coord(self):
59: (8)                     """
60: (8)                     Excel-style representation of the range
61: (8)                     """
62: (8)                     fmt = "{min_col}{min_row}:{max_col}{max_row}"
```

```
(8)             if (self.min_col == self.max_col
(12)                and self.min_row == self.max_row):
(12)                fmt = "{min_col}{min_row}"
(8)             return fmt.format(
(12)                min_col=get_column_letter(self.min_col),
(12)                min_row=self.min_row,
(12)                max_col=get_column_letter(self.max_col),
(12)                max_row=self.max_row
(8)             )
(4)         @property
(4)         def rows(self):
(8)             """
(8)             Return cell coordinates as rows
(8)             """
(8)             for row in range(self.min_row, self.max_row+1):
(12)                yield [(row, col) for col in range(self.min_col, self.max_col+1)]
(4)         @property
(4)         def cols(self):
(8)             """
(8)             Return cell coordinates as columns
(8)             """
(8)             for col in range(self.min_col, self.max_col+1):
(12)                yield [(row, col) for row in range(self.min_row, self.max_row+1)]
(4)         @property
(4)         def cells(self):
(8)             from itertools import product
(8)             return product(range(self.min_row, self.max_row+1),
range(self.min_col, self.max_col+1))
(4)         def _check_title(self, other):
(8)             """
(8)             Check whether comparisons between ranges are possible.
(8)             Cannot compare ranges from different worksheets
(8)             Skip if the range passed in has no title.
(8)             """
(8)             if not isinstance(other, CellRange):
(12)                raise TypeError(repr(type(other)))
(8)             if other.title and self.title != other.title:
(12)                raise ValueError("Cannot work with ranges from different
worksheets")
(4)         def __repr__(self):
(8)             fmt = u"<{cls} {coord}>"
(8)             if self.title:
(12)                fmt = u"<{cls} {title!r}!{coord}>"
(8)             return fmt.format(cls=self.__class__.__name__, title=self.title,
coord=self.coord)
(4)         def __hash__(self):
(8)             return hash((self.min_row, self.min_col, self.max_row, self.max_col))
(4)         def __str__(self):
(8)             fmt = "{coord}"
(8)             title = self.title
(8)             if title:
(12)                fmt = u"{title}!{coord}"
(12)                title = quote_sheetname(title)
(8)             return fmt.format(title=title, coord=self.coord)
(4)         def __copy__(self):
(8)             return self.__class__(min_col=self.min_col, min_row=self.min_row,
(30)                                  max_col=self.max_col, max_row=self.max_row,
(30)                                  title=self.title)
(4)         def shift(self, col_shift=0, row_shift=0):
(8)             """
(8)             Shift the focus of the range according to the shift values
(*col_shift*, *row_shift*).
(8)             :type col_shift: int
(8)             :param col_shift: number of columns to be moved by, can be negative
(8)             :type row_shift: int
(8)             :param row_shift: number of rows to be moved by, can be negative
(8)             :raise: :class:`ValueError` if any row or column index < 1
(8)             """
(8)             if (self.min_col + col_shift <= 0
(12)                         or self.min_row + row_shift <= 0):
(12)                         raise ValueError("Invalid shift value: col_shift={0}, row_shift={1}".format(col_shift, row_shift))
(8)                      self.min_col += col_shift
(8)                      self.min_row += row_shift
(8)                      self.max_col += col_shift
(8)                      self.max_row += row_shift
(4)                  def __ne__(self, other):
(8)                      """
(8)                      Test whether the ranges are not equal.
(8)                      :type other: openpyxl.worksheet.cell_range.CellRange
(8)                      :param other: Other sheet range
(8)                      :return: ``True`` if *range* != *other*.
(8)                      """
(8)                      try:
(12)                         self._check_title(other)
(8)                      except ValueError:
(12)                         return True
(8)                      return (
(12)                         other.min_row != self.min_row
(12)                         or self.max_row != other.max_row
(12)                         or other.min_col != self.min_col
(12)                         or self.max_col != other.max_col
(8)                      )
(4)                  def __eq__(self, other):
(8)                      """
(8)                      Test whether the ranges are equal.
(8)                      :type other: openpyxl.worksheet.cell_range.CellRange
(8)                      :param other: Other sheet range
(8)                      :return: ``True`` if *range* == *other*.
(8)                      """
(8)                      return not self.__ne__(other)
(4)                  def issubset(self, other):
(8)                      """
(8)                      Test whether every cell in this range is also in *other*.
(8)                      :type other: openpyxl.worksheet.cell_range.CellRange
(8)                      :param other: Other sheet range
(8)                      :return: ``True`` if *range* <= *other*.
(8)                      """
(8)                      self._check_title(other)
(8)                      return other.__superset(self)
(4)                  __le__ = issubset
(4)                  def __lt__(self, other):
(8)                      """
(8)                      Test whether *other* contains every cell of this range, and more.
(8)                      :type other: openpyxl.worksheet.cell_range.CellRange
(8)                      :param other: Other sheet range
(8)                      :return: ``True`` if *range* < *other*.
(8)                      """
(8)                      return self.__le__(other) and self.__ne__(other)
(4)                  def __superset(self, other):
(8)                      return (
(12)                         (self.min_row <= other.min_row <= other.max_row <= self.max_row)
(12)                         and
(12)                         (self.min_col <= other.min_col <= other.max_col <= self.max_col)
(8)                      )
(4)                  def issuperset(self, other):
(8)                      """
(8)                      Test whether every cell in *other* is in this range.
(8)                      :type other: openpyxl.worksheet.cell_range.CellRange
(8)                      :param other: Other sheet range
(8)                      :return: ``True`` if *range* >= *other* (or *other* in *range*).
(8)                      """
(8)                      self._check_title(other)
(8)                      return self.__superset(other)
(4)                  __ge__ = issuperset
(4)                  def __contains__(self, coord):
(8)                      """
(8)                      Check whether the range contains a particular cell coordinate
```

```
(8)                 """
(8)                 cr = self.__class__(coord)
(8)                 return self.__superset(cr)
(4)             def __gt__(self, other):
(8)                 """
(8)                 Test whether this range contains every cell in *other*, and more.
(8)                 :type other: openpyxl.worksheet.cell_range.CellRange
(8)                 :param other: Other sheet range
(8)                 :return: ``True`` if *range* > *other*.
(8)                 """
(8)                 return self.__ge__(other) and self.__ne__(other)
(4)             def isdisjoint(self, other):
(8)                 """
(8)                 Return ``True`` if this range has no cell in common with *other*.
(8)                 Ranges are disjoint if and only if their intersection is the empty range.
(8)                 :type other: openpyxl.worksheet.cell_range.CellRange
(8)                 :param other: Other sheet range.
(8)                 :return: ``True`` if the range has no cells in common with other.
(8)                 """
(8)                 self._check_title(other)
(8)                 if self.bounds > other.bounds:
(12)                    self, other = other, self
(8)                 return (self.max_col < other.min_col
(16)                        or self.max_row < other.min_row
(16)                        or other.max_row < self.min_row)
(4)             def intersection(self, other):
(8)                 """
(8)                 Return a new range with cells common to this range and *other*
(8)                 :type other: openpyxl.worksheet.cell_range.CellRange
(8)                 :param other: Other sheet range.
(8)                 :return: the intersecting sheet range.
(8)                 :raise: :class:`ValueError` if the *other* range doesn't intersect
(12)                    with this range.
(8)                 """
(8)                 if self.isdisjoint(other):
(12)                    raise ValueError("Range {0} doesn't intersect {0}".format(self, other))
(8)                 min_row = max(self.min_row, other.min_row)
(8)                 max_row = min(self.max_row, other.max_row)
(8)                 min_col = max(self.min_col, other.min_col)
(8)                 max_col = min(self.max_col, other.max_col)
(8)                 return CellRange(min_col=min_col, min_row=min_row, max_col=max_col,
(25)                                 max_row=max_row)
(4)             __and__ = intersection
(4)             def union(self, other):
(8)                 """
(8)                 Return the minimal superset of this range and *other*. This new range
(8)                 will contain all cells from this range, *other*, and any additional
(8)                 cells required to form a rectangular ``CellRange``.
(8)                 :type other: openpyxl.worksheet.cell_range.CellRange
(8)                 :param other: Other sheet range.
(8)                 :return: a ``CellRange`` that is a superset of this and *other*.
(8)                 """
(8)                 self._check_title(other)
(8)                 min_row = min(self.min_row, other.min_row)
(8)                 max_row = max(self.max_row, other.max_row)
(8)                 min_col = min(self.min_col, other.min_col)
(8)                 max_col = max(self.max_col, other.max_col)
(8)                 return CellRange(min_col=min_col, min_row=min_row, max_col=max_col,
(25)                                 max_row=max_row, title=self.title)
(4)             __or__ = union
(4)             def __iter__(self):
(8)                 """
(8)                 For use as a dictionary elsewhere in the library.
(8)                 """
(8)                 for x in self.__attrs__:
(12)                    if x == "title":
(16)                        continue
```

```
(12)                v = getattr(self, x)
(12)                yield x, v
(4)         def expand(self, right=0, down=0, left=0, up=0):
(8)             """
(8)             Expand the range by the dimensions provided.
(8)             :type right: int
(8)             :param right: expand range to the right by this number of cells
(8)             :type down: int
(8)             :param down: expand range down by this number of cells
(8)             :type left: int
(8)             :param left: expand range to the left by this number of cells
(8)             :type up: int
(8)             :param up: expand range up by this number of cells
(8)             """
(8)             self.min_col -= left
(8)             self.min_row -= up
(8)             self.max_col += right
(8)             self.max_row += down
(4)         def shrink(self, right=0, bottom=0, left=0, top=0):
(8)             """
(8)             Shrink the range by the dimensions provided.
(8)             :type right: int
(8)             :param right: shrink range from the right by this number of cells
(8)             :type down: int
(8)             :param down: shrink range from the top by this number of cells
(8)             :type left: int
(8)             :param left: shrink range from the left by this number of cells
(8)             :type up: int
(8)             :param up: shrink range from the bottom by this number of cells
(8)             """
(8)             self.min_col += left
(8)             self.min_row += top
(8)             self.max_col -= right
(8)             self.max_row -= bottom
(4)         @property
(4)         def size(self):
(8)             """ Return the size of the range as a dictionary of rows and columns. """
(8)             cols = self.max_col + 1 - self.min_col
(8)             rows = self.max_row + 1 - self.min_row
(8)             return {'columns':cols, 'rows':rows}
(4)         @property
(4)         def top(self):
(8)             """A list of cell coordinates that comprise the top of the range"""
(8)             return [(self.min_row, col) for col in range(self.min_col, self.max_col+1)]
(4)         @property
(4)         def bottom(self):
(8)             """A list of cell coordinates that comprise the bottom of the range"""
(8)             return [(self.max_row, col) for col in range(self.min_col, self.max_col+1)]
(4)         @property
(4)         def left(self):
(8)             """A list of cell coordinates that comprise the left-side of the range"""
(8)             return [(row, self.min_col) for row in range(self.min_row, self.max_row+1)]
(4)         @property
(4)         def right(self):
(8)             """A list of cell coordinates that comprise the right-side of the range"""
(8)             return [(row, self.max_col) for row in range(self.min_row, self.max_row+1)]
(0)     class MultiCellRange(Strict):
(4)         ranges = UniqueSequence(expected_type=CellRange)
(4)         def __init__(self, ranges=set()):
(8)             if isinstance(ranges, str):
(12)                ranges = [CellRange(r) for r in ranges.split()]
(8)             self.ranges = set(ranges)
```

```
325: (4)                def __contains__(self, coord):
326: (8)                    if isinstance(coord, str):
327: (12)                       coord = CellRange(coord)
328: (8)                    for r in self.ranges:
329: (12)                       if coord <= r:
330: (16)                           return True
331: (8)                    return False
332: (4)                def __repr__(self):
333: (8)                    ranges = " ".join([str(r) for r in self.sorted()])
334: (8)                    return f"<{self.__class__.__name__} [{ranges}]>"
335: (4)                def __str__(self):
336: (8)                    ranges = u" ".join([str(r) for r in self.sorted()])
337: (8)                    return ranges
338: (4)                def __hash__(self):
339: (8)                    return hash(str(self))
340: (4)                def sorted(self):
341: (8)                    """
342: (8)                    Return a sorted list of items
343: (8)                    """
344: (8)                    return sorted(self.ranges, key=attrgetter('min_col', 'min_row',
'max_col', 'max_row'))
345: (4)                def add(self, coord):
346: (8)                    """
347: (8)                    Add a cell coordinate or CellRange
348: (8)                    """
349: (8)                    cr = coord
350: (8)                    if isinstance(coord, str):
351: (12)                       cr = CellRange(coord)
352: (8)                    elif not isinstance(coord, CellRange):
353: (12)                       raise ValueError("You can only add CellRanges")
354: (8)                    if cr not in self:
355: (12)                       self.ranges.add(cr)
356: (4)                def __iadd__(self, coord):
357: (8)                    self.add(coord)
358: (8)                    return self
359: (4)                def __eq__(self, other):
360: (8)                    if  isinstance(other, str):
361: (12)                       other = self.__class__(other)
362: (8)                    return self.ranges == other.ranges
363: (4)                def __ne__(self, other):
364: (8)                    return not self == other
365: (4)                def __bool__(self):
366: (8)                    return bool(self.ranges)
367: (4)                def remove(self, coord):
368: (8)                    if not isinstance(coord, CellRange):
369: (12)                       coord = CellRange(coord)
370: (8)                    self.ranges.remove(coord)
371: (4)                def __iter__(self):
372: (8)                    for cr in self.ranges:
373: (12)                       yield cr
374: (4)                def __copy__(self):
375: (8)                    ranges = {copy(r) for r in self.ranges}
376: (8)                    return MultiCellRange(ranges)

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

File 172 - cell_watch.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Sequence,
4: (4)                  String,
5: (0)              )
6: (0)              class CellWatch(Serialisable):
7: (4)                  tagname = "cellWatch"
8: (4)                  r = String()
9: (4)                  def __init__(self,
10: (17)                             r=None,
11: (16)                            ):
```

```
12: (8)                 self.r = r
13: (0)             class CellWatches(Serialisable):
14: (4)                 tagname = "cellWatches"
15: (4)                 cellWatch = Sequence(expected_type=CellWatch)
16: (4)                 __elements__ = ('cellWatch',)
17: (4)                 def __init__(self,
18: (17)                             cellWatch=(),
19: (16)                            ):
20: (8)                     self.cellWatch = cellWatch
```

File 173 - dimensions.py:

```
1: (0)              from copy import copy
2: (0)              from openpyxl.compat import safe_string
3: (0)              from openpyxl.utils import (
4: (4)                  get_column_letter,
5: (4)                  get_column_interval,
6: (4)                  column_index_from_string,
7: (4)                  range_boundaries,
8: (0)              )
9: (0)              from openpyxl.utils.units import DEFAULT_COLUMN_WIDTH
10: (0)             from openpyxl.descriptors import (
11: (4)                 Integer,
12: (4)                 Float,
13: (4)                 Bool,
14: (4)                 Strict,
15: (4)                 String,
16: (4)                 Alias,
17: (0)             )
18: (0)             from openpyxl.descriptors.serialisable import Serialisable
19: (0)             from openpyxl.styles.styleable import StyleableObject
20: (0)             from openpyxl.utils.bound_dictionary import BoundDictionary
21: (0)             from openpyxl.xml.functions import Element
22: (0)             class Dimension(Strict, StyleableObject):
23: (4)                 """Information about the display properties of a row or column."""
24: (4)                 __fields__ = ('hidden',
25: (17)                             'outlineLevel',
26: (17)                             'collapsed',)
27: (4)                 index = Integer()
28: (4)                 hidden = Bool()
29: (4)                 outlineLevel = Integer(allow_none=True)
30: (4)                 outline_level = Alias('outlineLevel')
31: (4)                 collapsed = Bool()
32: (4)                 style = Alias('style_id')
33: (4)                 def __init__(self, index, hidden, outlineLevel,
34: (17)                             collapsed, worksheet, visible=True, style=None):
35: (8)                     super().__init__(sheet=worksheet, style_array=style)
36: (8)                     self.index = index
37: (8)                     self.hidden = hidden
38: (8)                     self.outlineLevel = outlineLevel
39: (8)                     self.collapsed = collapsed
40: (4)                 def __iter__(self):
41: (8)                     for key in self.__fields__:
42: (12)                        value = getattr(self, key, None)
43: (12)                        if value:
44: (16)                            yield key, safe_string(value)
45: (4)                 def __copy__(self):
46: (8)                     cp = self.__new__(self.__class__)
47: (8)                     attrib = self.__dict__
48: (8)                     attrib['worksheet'] = self.parent
49: (8)                     cp.__init__(**attrib)
50: (8)                     cp._style = copy(self._style)
51: (8)                     return cp
52: (4)                 def __repr__(self):
53: (8)                     return f"<{self.__class__.__name__} Instance, Attributes=
{dict(self)}>"
54: (0)             class RowDimension(Dimension):
```

```
55: (4)             """Information about the display properties of a row."""
56: (4)             __fields__ = Dimension.__fields__ + ('ht', 'customFormat', 'customHeight',
's',
57: (41)                                                 'thickBot', 'thickTop')
58: (4)             r = Alias('index')
59: (4)             s = Alias('style_id')
60: (4)             ht = Float(allow_none=True)
61: (4)             height = Alias('ht')
62: (4)             thickBot = Bool()
63: (4)             thickTop = Bool()
64: (4)             def __init__(self,
65: (17)                         worksheet,
66: (17)                         index=0,
67: (17)                         ht=None,
68: (17)                         customHeight=None, # do not write
69: (17)                         s=None,
70: (17)                         customFormat=None, # do not write
71: (17)                         hidden=False,
72: (17)                         outlineLevel=0,
73: (17)                         outline_level=None,
74: (17)                         collapsed=False,
75: (17)                         visible=None,
76: (17)                         height=None,
77: (17)                         r=None,
78: (17)                         spans=None,
79: (17)                         thickBot=None,
80: (17)                         thickTop=None,
81: (17)                         **kw
82: (17)                         ):
83: (8)                 if r is not None:
84: (12)                    index = r
85: (8)                 if height is not None:
86: (12)                    ht = height
87: (8)                 self.ht = ht
88: (8)                 if visible is not None:
89: (12)                    hidden = not visible
90: (8)                 if outline_level is not None:
91: (12)                    outlineLevel = outline_level
92: (8)                 self.thickBot = thickBot
93: (8)                 self.thickTop = thickTop
94: (8)                 super().__init__(index, hidden, outlineLevel,
95: (43)                                                   collapsed, worksheet, style=s)
96: (4)             @property
97: (4)             def customFormat(self):
98: (8)                 """Always true if there is a style for the row"""
99: (8)                 return self.has_style
100: (4)            @property
101: (4)            def customHeight(self):
102: (8)                """Always true if there is a height for the row"""
103: (8)                return self.ht is not None
104: (0)        class ColumnDimension(Dimension):
105: (4)            """Information about the display properties of a column."""
106: (4)            width = Float()
107: (4)            bestFit = Bool()
108: (4)            auto_size = Alias('bestFit')
109: (4)            index = String()
110: (4)            min = Integer(allow_none=True)
111: (4)            max = Integer(allow_none=True)
112: (4)            collapsed = Bool()
113: (4)            __fields__ = Dimension.__fields__ + ('width', 'bestFit', 'customWidth',
'style',
114: (41)                                                'min', 'max')
115: (4)            def __init__(self,
116: (17)                        worksheet,
117: (17)                        index='A',
118: (17)                        width=DEFAULT_COLUMN_WIDTH,
119: (17)                        bestFit=False,
120: (17)                        hidden=False,
121: (17)                        outlineLevel=0,
122: (17)                    outline_level=None,
123: (17)                    collapsed=False,
124: (17)                    style=None,
125: (17)                    min=None,
126: (17)                    max=None,
127: (17)                    customWidth=False, # do not write
128: (17)                    visible=None,
129: (17)                    auto_size=None,):
130: (8)            self.width = width
131: (8)            self.min = min
132: (8)            self.max = max
133: (8)            if visible is not None:
134: (12)               hidden = not visible
135: (8)            if auto_size is not None:
136: (12)               bestFit = auto_size
137: (8)            self.bestFit = bestFit
138: (8)            if outline_level is not None:
139: (12)               outlineLevel = outline_level
140: (8)            self.collapsed = collapsed
141: (8)            super().__init__(index, hidden, outlineLevel,
142: (46)                                                 collapsed, worksheet,
style=style)
143: (4)        @property
144: (4)        def customWidth(self):
145: (8)            """Always true if there is a width for the column"""
146: (8)            return bool(self.width)
147: (4)        def reindex(self):
148: (8)            """
149: (8)            Set boundaries for column definition
150: (8)            """
151: (8)            if not all([self.min, self.max]):
152: (12)               self.min = self.max = column_index_from_string(self.index)
153: (4)        @property
154: (4)        def range(self):
155: (8)            """Return the range of cells actually covered"""
156: (8)            return f"{get_column_letter(self.min)}:{get_column_letter(self.max)}"
157: (4)        def to_tree(self):
158: (8)            attrs = dict(self)
159: (8)            if attrs.keys() != {'min', 'max'}:
160: (12)               return Element("col", **attrs)
161: (0)    class DimensionHolder(BoundDictionary):
162: (4)        """
163: (4)        Allow columns to be grouped
164: (4)        """
165: (4)        def __init__(self, worksheet, reference="index", default_factory=None):
166: (8)            self.worksheet = worksheet
167: (8)            self.max_outline = None
168: (8)            self.default_factory = default_factory
169: (8)            super().__init__(reference, default_factory)
170: (4)        def group(self, start, end=None, outline_level=1, hidden=False):
171: (8)            """allow grouping a range of consecutive rows or columns together
172: (8)            :param start: first row or column to be grouped (mandatory)
173: (8)            :param end: last row or column to be grouped (optional, default to
start)
174: (8)            :param outline_level: outline level
175: (8)            :param hidden: should the group be hidden on workbook open or not
176: (8)            """
177: (8)            if end is None:
178: (12)               end = start
179: (8)            if isinstance(self.default_factory(), ColumnDimension):
180: (12)               new_dim = self[start]
181: (12)               new_dim.outline_level = outline_level
182: (12)               new_dim.hidden = hidden
183: (12)               work_sequence = get_column_interval(start, end)[1:]
184: (12)               for column_letter in work_sequence:
185: (16)                   if column_letter in self:
186: (20)                       del self[column_letter]
187: (12)               new_dim.min, new_dim.max = map(column_index_from_string, (start,
end))
```

```
188: (8)                    elif isinstance(self.default_factory(), RowDimension):
189: (12)                       for el in range(start, end + 1):
190: (16)                           new_dim = self.worksheet.row_dimensions[el]
191: (16)                           new_dim.outline_level = outline_level
192: (16)                           new_dim.hidden = hidden
193: (4)                def to_tree(self):
194: (8)                    def sorter(value):
195: (12)                       value.reindex()
196: (12)                       return value.min
197: (8)                    el = Element('cols')
198: (8)                    outlines = set()
199: (8)                    for col in sorted(self.values(), key=sorter):
200: (12)                       obj = col.to_tree()
201: (12)                       if obj is not None:
202: (16)                           outlines.add(col.outlineLevel)
203: (16)                           el.append(obj)
204: (8)                    if outlines:
205: (12)                       self.max_outline = max(outlines)
206: (8)                    if len(el):
207: (12)                       return el # must have at least one child
208: (0)            class SheetFormatProperties(Serialisable):
209: (4)                tagname = "sheetFormatPr"
210: (4)                baseColWidth = Integer(allow_none=True)
211: (4)                defaultColWidth = Float(allow_none=True)
212: (4)                defaultRowHeight = Float()
213: (4)                customHeight = Bool(allow_none=True)
214: (4)                zeroHeight = Bool(allow_none=True)
215: (4)                thickTop = Bool(allow_none=True)
216: (4)                thickBottom = Bool(allow_none=True)
217: (4)                outlineLevelRow = Integer(allow_none=True)
218: (4)                outlineLevelCol = Integer(allow_none=True)
219: (4)                def __init__(self,
220: (17)                            baseColWidth=8, #according to spec
221: (17)                            defaultColWidth=None,
222: (17)                            defaultRowHeight=15,
223: (17)                            customHeight=None,
224: (17)                            zeroHeight=None,
225: (17)                            thickTop=None,
226: (17)                            thickBottom=None,
227: (17)                            outlineLevelRow=None,
228: (17)                            outlineLevelCol=None,
229: (16)                           ):
230: (8)                    self.baseColWidth = baseColWidth
231: (8)                    self.defaultColWidth = defaultColWidth
232: (8)                    self.defaultRowHeight = defaultRowHeight
233: (8)                    self.customHeight = customHeight
234: (8)                    self.zeroHeight = zeroHeight
235: (8)                    self.thickTop = thickTop
236: (8)                    self.thickBottom = thickBottom
237: (8)                    self.outlineLevelRow = outlineLevelRow
238: (8)                    self.outlineLevelCol = outlineLevelCol
239: (0)            class SheetDimension(Serialisable):
240: (4)                tagname = "dimension"
241: (4)                ref = String()
242: (4)                def __init__(self,
243: (17)                            ref=None,
244: (16)                           ):
245: (8)                    self.ref = ref
246: (4)                @property
247: (4)                def boundaries(self):
248: (8)                    return range_boundaries(self.ref)
```

File 174 - properties.py:

```
1: (0)            """Worksheet Properties"""
2: (0)            from openpyxl.descriptors.serialisable import Serialisable
3: (0)            from openpyxl.descriptors import String, Bool, Typed
```

```
4: (0)              from openpyxl.styles.colors import ColorDescriptor
5: (0)              class Outline(Serialisable):
6: (4)                  tagname = "outlinePr"
7: (4)                  applyStyles = Bool(allow_none=True)
8: (4)                  summaryBelow = Bool(allow_none=True)
9: (4)                  summaryRight = Bool(allow_none=True)
10: (4)                 showOutlineSymbols = Bool(allow_none=True)
11: (4)                 def __init__(self,
12: (17)                             applyStyles=None,
13: (17)                             summaryBelow=None,
14: (17)                             summaryRight=None,
15: (17)                             showOutlineSymbols=None
16: (17)                             ):
17: (8)                     self.applyStyles = applyStyles
18: (8)                     self.summaryBelow = summaryBelow
19: (8)                     self.summaryRight = summaryRight
20: (8)                     self.showOutlineSymbols = showOutlineSymbols
21: (0)             class PageSetupProperties(Serialisable):
22: (4)                 tagname = "pageSetUpPr"
23: (4)                 autoPageBreaks = Bool(allow_none=True)
24: (4)                 fitToPage = Bool(allow_none=True)
25: (4)                 def __init__(self, autoPageBreaks=None, fitToPage=None):
26: (8)                     self.autoPageBreaks = autoPageBreaks
27: (8)                     self.fitToPage = fitToPage
28: (0)             class WorksheetProperties(Serialisable):
29: (4)                 tagname = "sheetPr"
30: (4)                 codeName = String(allow_none=True)
31: (4)                 enableFormatConditionsCalculation = Bool(allow_none=True)
32: (4)                 filterMode = Bool(allow_none=True)
33: (4)                 published = Bool(allow_none=True)
34: (4)                 syncHorizontal = Bool(allow_none=True)
35: (4)                 syncRef = String(allow_none=True)
36: (4)                 syncVertical = Bool(allow_none=True)
37: (4)                 transitionEvaluation = Bool(allow_none=True)
38: (4)                 transitionEntry = Bool(allow_none=True)
39: (4)                 tabColor = ColorDescriptor(allow_none=True)
40: (4)                 outlinePr = Typed(expected_type=Outline, allow_none=True)
41: (4)                 pageSetUpPr = Typed(expected_type=PageSetupProperties, allow_none=True)
42: (4)                 __elements__ = ('tabColor', 'outlinePr', 'pageSetUpPr')
43: (4)                 def __init__(self,
44: (17)                             codeName=None,
45: (17)                             enableFormatConditionsCalculation=None,
46: (17)                             filterMode=None,
47: (17)                             published=None,
48: (17)                             syncHorizontal=None,
49: (17)                             syncRef=None,
50: (17)                             syncVertical=None,
51: (17)                             transitionEvaluation=None,
52: (17)                             transitionEntry=None,
53: (17)                             tabColor=None,
54: (17)                             outlinePr=None,
55: (17)                             pageSetUpPr=None
56: (17)                             ):
57: (8)                     """ Attributes """
58: (8)                     self.codeName = codeName
59: (8)                     self.enableFormatConditionsCalculation =
enableFormatConditionsCalculation
60: (8)                     self.filterMode = filterMode
61: (8)                     self.published = published
62: (8)                     self.syncHorizontal = syncHorizontal
63: (8)                     self.syncRef = syncRef
64: (8)                     self.syncVertical = syncVertical
65: (8)                     self.transitionEvaluation = transitionEvaluation
66: (8)                     self.transitionEntry = transitionEntry
67: (8)                     """ Elements """
68: (8)                     self.tabColor = tabColor
69: (8)                     if outlinePr is None:
70: (12)                        self.outlinePr = Outline(summaryBelow=True, summaryRight=True)
71: (8)                     else:
```

```
72: (12)                    self.outlinePr = outlinePr
73: (8)                 if pageSetUpPr is None:
74: (12)                    pageSetUpPr = PageSetupProperties()
75: (8)                 self.pageSetUpPr = pageSetUpPr

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

File 175 - protection.py:

1: (0)              from openpyxl.descriptors import (
2: (4)                  Bool,
3: (4)                  String,
4: (4)                  Alias,
5: (4)                  Integer,
6: (0)              )
7: (0)              from openpyxl.descriptors.serialisable import Serialisable
8: (0)              from openpyxl.descriptors.excel import (
9: (4)                  Base64Binary,
10: (0)             )
11: (0)             from openpyxl.utils.protection import hash_password
12: (0)             class _Protected:
13: (4)                 _password = None
14: (4)                 def set_password(self, value='', already_hashed=False):
15: (8)                     """Set a password on this sheet."""
16: (8)                     if not already_hashed:
17: (12)                        value = hash_password(value)
18: (8)                     self._password = value
19: (4)                 @property
20: (4)                 def password(self):
21: (8)                     """Return the password value, regardless of hash."""
22: (8)                     return self._password
23: (4)                 @password.setter
24: (4)                 def password(self, value):
25: (8)                     """Set a password directly, forcing a hash step."""
26: (8)                     self.set_password(value)
27: (0)             class SheetProtection(Serialisable, _Protected):
28: (4)                 """
29: (4)                 Information about protection of various aspects of a sheet. True values
30: (4)                 mean that protection for the object or action is active This is the
31: (4)                 **default** when protection is active, ie. users cannot do something
32: (4)                 """
33: (4)                 tagname = "sheetProtection"
34: (4)                 sheet = Bool()
35: (4)                 enabled = Alias('sheet')
36: (4)                 objects = Bool()
37: (4)                 scenarios = Bool()
38: (4)                 formatCells = Bool()
39: (4)                 formatColumns = Bool()
40: (4)                 formatRows = Bool()
41: (4)                 insertColumns = Bool()
42: (4)                 insertRows = Bool()
43: (4)                 insertHyperlinks = Bool()
44: (4)                 deleteColumns = Bool()
45: (4)                 deleteRows = Bool()
46: (4)                 selectLockedCells = Bool()
47: (4)                 selectUnlockedCells = Bool()
48: (4)                 sort = Bool()
49: (4)                 autoFilter = Bool()
50: (4)                 pivotTables = Bool()
51: (4)                 saltValue = Base64Binary(allow_none=True)
52: (4)                 spinCount = Integer(allow_none=True)
53: (4)                 algorithmName = String(allow_none=True)
54: (4)                 hashValue = Base64Binary(allow_none=True)
55: (4)                 __attrs__ = ('selectLockedCells', 'selectUnlockedCells', 'algorithmName',
56: (14)                          'sheet', 'objects', 'insertRows', 'insertHyperlinks',
'autoFilter',
57: (14)                          'scenarios', 'formatColumns', 'deleteColumns', 'insertColumns',
58: (14)                          'pivotTables', 'deleteRows', 'formatCells', 'saltValue',
'formatRows',
59: (14)                          'sort', 'spinCount', 'password', 'hashValue')
60: (4)                 def __init__(self, sheet=False, objects=False, scenarios=False,
61: (17)                             formatCells=True, formatRows=True, formatColumns=True,
62: (17)                             insertColumns=True, insertRows=True, insertHyperlinks=True,
63: (17)                             deleteColumns=True, deleteRows=True, selectLockedCells=False,
64: (17)                             selectUnlockedCells=False, sort=True, autoFilter=True,
pivotTables=True,
65: (17)                             password=None, algorithmName=None, saltValue=None,
spinCount=None, hashValue=None):
66: (8)                     self.sheet = sheet
67: (8)                     self.objects = objects
68: (8)                     self.scenarios = scenarios
69: (8)                     self.formatCells = formatCells
70: (8)                     self.formatColumns = formatColumns
71: (8)                     self.formatRows = formatRows
72: (8)                     self.insertColumns = insertColumns
73: (8)                     self.insertRows = insertRows
74: (8)                     self.insertHyperlinks = insertHyperlinks
75: (8)                     self.deleteColumns = deleteColumns
76: (8)                     self.deleteRows = deleteRows
77: (8)                     self.selectLockedCells = selectLockedCells
78: (8)                     self.selectUnlockedCells = selectUnlockedCells
79: (8)                     self.sort = sort
80: (8)                     self.autoFilter = autoFilter
81: (8)                     self.pivotTables = pivotTables
82: (8)                     if password is not None:
83: (12)                        self.password = password
84: (8)                     self.algorithmName = algorithmName
85: (8)                     self.saltValue = saltValue
86: (8)                     self.spinCount = spinCount
87: (8)                     self.hashValue = hashValue
88: (4)                 def set_password(self, value='', already_hashed=False):
89: (8)                     super().set_password(value, already_hashed)
90: (8)                     self.enable()
91: (4)                 def enable(self):
92: (8)                     self.sheet = True
93: (4)                 def disable(self):
94: (8)                     self.sheet = False
95: (4)                 def  __bool__(self):
96: (8)                     return self.sheet

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

File 176 - header_footer.py:

1: (0)              import re
2: (0)              from warnings import warn
3: (0)              from openpyxl.descriptors import (
4: (4)                  Alias,
5: (4)                  Bool,
6: (4)                  Strict,
7: (4)                  String,
8: (4)                  Integer,
9: (4)                  MatchPattern,
10: (4)                 Typed,
11: (0)             )
12: (0)             from openpyxl.descriptors.serialisable import Serialisable
13: (0)             from openpyxl.xml.functions import Element
14: (0)             from openpyxl.utils.escape import escape, unescape
15: (0)             FONT_PATTERN = '&"(?P<font>.+)"'
16: (0)             COLOR_PATTERN  = "&K(?P<color>[A-F0-9]{6})"
17: (0)             SIZE_REGEX = r"&(?P<size>\d+\s?)"
18: (0)             FORMAT_REGEX = re.compile("{0}|{1}|{2}".format(FONT_PATTERN, COLOR_PATTERN,
19: (47)                                                           SIZE_REGEX)
20: (26)                                      )
21: (0)             def _split_string(text):
22: (4)                 """
23: (4)                 Split the combined (decoded) string into left, center and right parts
24: (4)                 """
```

```
(4)          ITEM_REGEX = re.compile("""
(4)          (&L(?P<left>.+?))?
(4)          (&C(?P<center>.+?))?
(4)          (&R(?P<right>.+?))?
(4)          $""", re.VERBOSE | re.DOTALL)
(4)          m = ITEM_REGEX.match(text)
(4)          try:
(8)              parts = m.groupdict()
(4)          except AttributeError:
(8)              warn("""Cannot parse header or footer so it will be ignored""")
(8)              parts = {'left':'', 'right':'', 'center':''}
(4)          return parts
(0)      class _HeaderFooterPart(Strict):
(4)          """
(4)          Individual left/center/right header/footer part
(4)          Do not use directly.
(4)          Header & Footer ampersand codes:
(4)          * &A   Inserts the worksheet name
(4)          * &B   Toggles bold
(4)          * &D or &[Date]   Inserts the current date
(4)          * &E   Toggles double-underline
(4)          * &F or &[File]   Inserts the workbook name
(4)          * &I   Toggles italic
(4)          * &N or &[Pages]   Inserts the total page count
(4)          * &S   Toggles strikethrough
(4)          * &T   Inserts the current time
(4)          * &[Tab]   Inserts the worksheet name
(4)          * &U   Toggles underline
(4)          * &X   Toggles superscript
(4)          * &Y   Toggles subscript
(4)          * &P or &[Page]   Inserts the current page number
(4)          * &P+n   Inserts the page number incremented by n
(4)          * &P-n   Inserts the page number decremented by n
(4)          * &[Path]   Inserts the workbook path
(4)          * &&   Escapes the ampersand character
(4)          * &"fontname"   Selects the named font
(4)          * &nn   Selects the specified 2-digit font point size
(4)          Colours are in RGB Hex
(4)          """
(4)          text = String(allow_none=True)
(4)          font = String(allow_none=True)
(4)          size = Integer(allow_none=True)
(4)          RGB = ("^[A-Fa-f0-9]{6}$")
(4)          color = MatchPattern(allow_none=True, pattern=RGB)
(4)          def __init__(self, text=None, font=None, size=None, color=None):
(8)              self.text = text
(8)              self.font = font
(8)              self.size = size
(8)              self.color = color
(4)          def __str__(self):
(8)              """
(8)              Convert to Excel HeaderFooter miniformat minus position
(8)              """
(8)              fmt = []
(8)              if self.font:
(12)                 fmt.append(u'&"{0}"'.format(self.font))
(8)              if self.size:
(12)                 fmt.append("&{0} ".format(self.size))
(8)              if self.color:
(12)                 fmt.append("&K{0}".format(self.color))
(8)              return u"".join(fmt + [self.text])
(4)          def __bool__(self):
(8)              return bool(self.text)
(4)          @classmethod
(4)          def from_str(cls, text):
(8)              """
(8)              Convert from miniformat to object
(8)              """
(8)              keys = ('font', 'color', 'size')
```

```
        kw = dict((k, v) for match in FORMAT_REGEX.findall(text)
                  for k, v in zip(keys, match) if v)
        kw['text'] = FORMAT_REGEX.sub('', text)
        return cls(**kw)
class HeaderFooterItem(Strict):
    """
    Header or footer item
    """
    left = Typed(expected_type=_HeaderFooterPart)
    center = Typed(expected_type=_HeaderFooterPart)
    centre = Alias("center")
    right = Typed(expected_type=_HeaderFooterPart)
    __keys = ('L', 'C', 'R')
    def __init__(self, left=None, right=None, center=None):
        if left is None:
            left = _HeaderFooterPart()
        self.left = left
        if center is None:
            center = _HeaderFooterPart()
        self.center = center
        if right is None:
            right = _HeaderFooterPart()
        self.right = right
    def __str__(self):
        """
        Pack parts into a single string
        """
        TRANSFORM = {'&[Tab]': '&A', '&[Pages]': '&N', '&[Date]': '&D',
                     '&[Path]': '&Z', '&[Page]': '&P', '&[Time]': '&T', '&[File]': '&F',
                     '&[Picture]': '&G'}
        SUBS_REGEX = re.compile("|".join(["({0})".format(re.escape(k))
                                          for k in TRANSFORM]))
        def replace(match):
            """
            Callback for re.sub
            Replace expanded control with mini-format equivalent
            """
            sub = match.group(0)
            return TRANSFORM[sub]
        txt = []
        for key, part in zip(
            self.__keys, [self.left, self.center, self.right]):
            if part.text is not None:
                txt.append(u"&{0}{1}".format(key, str(part)))
        txt = "".join(txt)
        txt = SUBS_REGEX.sub(replace, txt)
        return escape(txt)
    def __bool__(self):
        return any([self.left, self.center, self.right])
    def to_tree(self, tagname):
        """
        Return as XML node
        """
        el = Element(tagname)
        el.text = str(self)
        return el
    @classmethod
    def from_tree(cls, node):
        if node.text:
            text = unescape(node.text)
            parts = _split_string(text)
            for k, v in parts.items():
                if v is not None:
                    parts[k] = _HeaderFooterPart.from_str(v)
            self = cls(**parts)
            return self
class HeaderFooter(Serialisable):
    tagname = "headerFooter"
```

```
162: (4)                differentOddEven = Bool(allow_none=True)
163: (4)                differentFirst = Bool(allow_none=True)
164: (4)                scaleWithDoc = Bool(allow_none=True)
165: (4)                alignWithMargins = Bool(allow_none=True)
166: (4)                oddHeader = Typed(expected_type=HeaderFooterItem, allow_none=True)
167: (4)                oddFooter = Typed(expected_type=HeaderFooterItem, allow_none=True)
168: (4)                evenHeader = Typed(expected_type=HeaderFooterItem, allow_none=True)
169: (4)                evenFooter = Typed(expected_type=HeaderFooterItem, allow_none=True)
170: (4)                firstHeader = Typed(expected_type=HeaderFooterItem, allow_none=True)
171: (4)                firstFooter = Typed(expected_type=HeaderFooterItem, allow_none=True)
172: (4)                __elements__ = ("oddHeader", "oddFooter", "evenHeader", "evenFooter",
"firstHeader", "firstFooter")
173: (4)                def __init__(self,
174: (17)                            differentOddEven=None,
175: (17)                            differentFirst=None,
176: (17)                            scaleWithDoc=None,
177: (17)                            alignWithMargins=None,
178: (17)                            oddHeader=None,
179: (17)                            oddFooter=None,
180: (17)                            evenHeader=None,
181: (17)                            evenFooter=None,
182: (17)                            firstHeader=None,
183: (17)                            firstFooter=None,
184: (16)                           ):
185: (8)                    self.differentOddEven = differentOddEven
186: (8)                    self.differentFirst = differentFirst
187: (8)                    self.scaleWithDoc = scaleWithDoc
188: (8)                    self.alignWithMargins = alignWithMargins
189: (8)                    if oddHeader is None:
190: (12)                       oddHeader = HeaderFooterItem()
191: (8)                    self.oddHeader = oddHeader
192: (8)                    if oddFooter is None:
193: (12)                       oddFooter = HeaderFooterItem()
194: (8)                    self.oddFooter = oddFooter
195: (8)                    if evenHeader is None:
196: (12)                       evenHeader = HeaderFooterItem()
197: (8)                    self.evenHeader = evenHeader
198: (8)                    if evenFooter is None:
199: (12)                       evenFooter = HeaderFooterItem()
200: (8)                    self.evenFooter = evenFooter
201: (8)                    if firstHeader is None:
202: (12)                       firstHeader = HeaderFooterItem()
203: (8)                    self.firstHeader = firstHeader
204: (8)                    if firstFooter is None:
205: (12)                       firstFooter = HeaderFooterItem()
206: (8)                    self.firstFooter = firstFooter
207: (4)                def __bool__(self):
208: (8)                    parts = [getattr(self, attr) for attr in self.__attrs__ +
self.__elements__]
209: (8)                    return any(parts)

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

File 177 - datavalidation.py:

1: (0)              from collections import defaultdict
2: (0)              from itertools import chain
3: (0)              from operator import itemgetter
4: (0)              from openpyxl.descriptors.serialisable import Serialisable
5: (0)              from openpyxl.descriptors import (
6: (4)                  Bool,
7: (4)                  NoneSet,
8: (4)                  String,
9: (4)                  Sequence,
10: (4)                 Alias,
11: (4)                 Integer,
12: (4)                 Convertible,
13: (0)             )
14: (0)             from openpyxl.descriptors.nested import NestedText
```

```
from openpyxl.utils import (
    rows_from_range,
    coordinate_to_tuple,
    get_column_letter,
)
def collapse_cell_addresses(cells, input_ranges=()):
    """ Collapse a collection of cell co-ordinates down into an optimal
        range or collection of ranges.
        E.g. Cells A1, A2, A3, B1, B2 and B3 should have the data-validation
        object applied, attempt to collapse down to a single range, A1:B3.
        Currently only collapsing contiguous vertical ranges (i.e. above
        example results in A1:A3 B1:B3).
    """
    ranges = list(input_ranges)
    raw_coords = (coordinate_to_tuple(cell) for cell in cells)
    grouped_coords = defaultdict(list)
    for row, col in sorted(raw_coords, key=itemgetter(1)):
        grouped_coords[col].append(row)
    for col, cells in grouped_coords.items():
        col = get_column_letter(col)
        fmt = "{0}{1}:{2}{3}"
        if len(cells) == 1:
            fmt = "{0}{1}"
        r = fmt.format(col, min(cells), col, max(cells))
        ranges.append(r)
    return " ".join(ranges)
def expand_cell_ranges(range_string):
    """
    Expand cell ranges to a sequence of addresses.
    Reverse of collapse_cell_addresses
    Eg. converts "A1:A2 B1:B2" to (A1, A2, B1, B2)
    """
    rows = (rows_from_range(rs) for rs in range_string.split()) # list of rows
    cells = (chain(*row) for row in rows) # flatten rows
    return set(chain(*cells))
from .cell_range import MultiCellRange
class DataValidation(Serialisable):
    tagname = "dataValidation"
    sqref = Convertible(expected_type=MultiCellRange)
    cells = Alias("sqref")
    ranges = Alias("sqref")
    showDropDown = Bool(allow_none=True)
    hide_drop_down = Alias('showDropDown')
    showInputMessage = Bool(allow_none=True)
    showErrorMessage = Bool(allow_none=True)
    allowBlank = Bool(allow_none=True)
    allow_blank = Alias('allowBlank')
    errorTitle = String(allow_none = True)
    error = String(allow_none = True)
    promptTitle = String(allow_none = True)
    prompt = String(allow_none = True)
    formula1 = NestedText(allow_none=True, expected_type=str)
    formula2 = NestedText(allow_none=True, expected_type=str)
    type = NoneSet(values=("whole", "decimal", "list", "date", "time",
                           "textLength", "custom"))
    errorStyle = NoneSet(values=("stop", "warning", "information"))
    imeMode = NoneSet(values=("noControl", "off", "on", "disabled",
                              "hiragana", "fullKatakana", "halfKatakana",
"fullAlpha","halfAlpha",
                              "fullHangul", "halfHangul"))
    operator = NoneSet(values=("between", "notBetween", "equal", "notEqual",
                               "lessThan", "lessThanOrEqual", "greaterThan",
"greaterThanOrEqual"))
    validation_type = Alias('type')
    def __init__(self,
                 type=None,
                 formula1=None,
                 formula2=None,
                 showErrorMessage=False,
    (17)                 showInputMessage=False,
    (17)                 showDropDown=False,
    (17)                 allowBlank=False,
    (17)                 sqref=(),
    (17)                 promptTitle=None,
    (17)                 errorStyle=None,
    (17)                 error=None,
    (17)                 prompt=None,
    (17)                 errorTitle=None,
    (17)                 imeMode=None,
    (17)                 operator=None,
    (17)                 allow_blank=None,
    (17)                 ):
    (8)         self.sqref = sqref
    (8)         self.showDropDown = showDropDown
    (8)         self.imeMode = imeMode
    (8)         self.operator = operator
    (8)         self.formula1 = formula1
     (8)         self.formula2 = formula2
     (8)         if allow_blank is not None:
     (12)            allowBlank = allow_blank
     (8)         self.allowBlank = allowBlank
     (8)         self.showErrorMessage = showErrorMessage
     (8)         self.showInputMessage = showInputMessage
     (8)         self.type = type
     (8)         self.promptTitle = promptTitle
     (8)         self.errorStyle = errorStyle
     (8)         self.error = error
     (8)         self.prompt = prompt
     (8)         self.errorTitle = errorTitle
     (4)     def add(self, cell):
     (8)         """Adds a cell or cell coordinate to this validator"""
     (8)         if hasattr(cell, "coordinate"):
     (12)            cell = cell.coordinate
     (8)         self.sqref += cell
     (4)     def __contains__(self, cell):
     (8)         if hasattr(cell, "coordinate"):
     (12)            cell = cell.coordinate
     (8)         return cell in self.sqref
     (0) class DataValidationList(Serialisable):
     (4)     tagname = "dataValidations"
     (4)     disablePrompts = Bool(allow_none=True)
     (4)     xWindow = Integer(allow_none=True)
     (4)     yWindow = Integer(allow_none=True)
     (4)     dataValidation = Sequence(expected_type=DataValidation)
     (4)     __elements__ = ('dataValidation',)
     (4)     __attrs__ = ('disablePrompts', 'xWindow', 'yWindow', 'count')
     (4)     def __init__(self,
     (17)                 disablePrompts=None,
     (17)                 xWindow=None,
     (17)                 yWindow=None,
     (17)                 count=None,
     (17)                 dataValidation=(),
     (16)                ):
     (8)         self.disablePrompts = disablePrompts
     (8)         self.xWindow = xWindow
     (8)         self.yWindow = yWindow
     (8)         self.dataValidation = dataValidation
     (4)     @property
     (4)     def count(self):
     (8)         return len(self)
     (4)     def __len__(self):
     (8)         return len(self.dataValidation)
     (4)     def append(self, dv):
     (8)         self.dataValidation.append(dv)
     (4)     def to_tree(self, tagname=None):
     (8)         """
     (8)         Need to skip validations that have no cell ranges
     (8)         """
```

```
151: (8)                    ranges = self.dataValidation # copy
152: (8)                    self.dataValidation = [r for r in self.dataValidation if
bool(r.sqref)]
153: (8)                    xml = super().to_tree(tagname)
154: (8)                    self.dataValidation = ranges
155: (8)                    return xml

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

File 178 - print_settings.py:

1: (0)              import re
2: (0)              from openpyxl.descriptors import (
3: (4)                  Strict,
4: (4)                  Integer,
5: (4)                  String,
6: (4)                  Typed,
7: (0)              )
8: (0)              from openpyxl.utils import quote_sheetname, absolute_coordinate
9: (0)              from openpyxl.utils.cell import SHEET_TITLE, SHEETRANGE_RE, RANGE_EXPR
10: (0)             from .cell_range import MultiCellRange
11: (0)             COL_RANGE = r"""(?P<cols>[$]?(?P<min_col>[a-zA-Z]{1,3}):[$]?(?P<max_col>[a-zA-
Z]{1,3}))"""
12: (0)             COL_RANGE_RE = re.compile(COL_RANGE)
13: (0)             ROW_RANGE = r"""(?P<rows>[$]?(?P<min_row>\d+):[$]?(?P<max_row>\d+))"""
14: (0)             ROW_RANGE_RE = re.compile(ROW_RANGE)
15: (0)             TITLES_REGEX = re.compile("""{0}{1}?,?{2}?,?""".format(SHEET_TITLE, ROW_RANGE,
COL_RANGE),
16: (26)                                      re.VERBOSE)
17: (0)             PRINT_AREA_RE = re.compile(f"({SHEET_TITLE})?(?P<cells>{RANGE_EXPR})",
re.VERBOSE)
18: (0)             class ColRange(Strict):
19: (4)                 """
20: (4)                 Represent a range of at least one column
21: (4)                 """
22: (4)                 min_col = String()
23: (4)                 max_col = String()
24: (4)                 def __init__(self, range_string=None, min_col=None, max_col=None):
25: (8)                     if range_string is not None:
26: (12)                        match = COL_RANGE_RE.match(range_string)
27: (12)                        if not match:
28: (16)                            raise ValueError(f"{range_string} is not a valid column
range")
29: (12)                        min_col, max_col = match.groups()[1:]
30: (8)                     self.min_col = min_col
31: (8)                     self.max_col = max_col
32: (4)                 def __eq__(self, other):
33: (8)                     if isinstance(other, self.__class__):
34: (12)                        return (self.min_col == other.min_col
35: (20)                                and
36: (20)                                self.max_col == other.max_col)
37: (8)                     elif isinstance(other, str):
38: (12)                        return (str(self) == other
39: (20)                                or
40: (20)                                f"{self.min_col}:{self.max_col}")
41: (8)                     return False
42: (4)                 def __repr__(self):
43: (8)                     return f"Range of columns from '{self.min_col}' to '{self.max_col}'"
44: (4)                 def __str__(self):
45: (8)                     return f"${self.min_col}:${self.max_col}"
46: (0)             class RowRange(Strict):
47: (4)                 """
48: (4)                 Represent a range of at least one row
49: (4)                 """
50: (4)                 min_row = Integer()
51: (4)                 max_row = Integer()
52: (4)                 def __init__(self, range_string=None, min_row=None, max_row=None):
53: (8)                     if range_string is not None:
54: (12)                        match = ROW_RANGE_RE.match(range_string)
```

```
55: (12)                    if not match:
56: (16)                        raise ValueError(f"{range_string} is not a valid row range")
57: (12)                    min_row, max_row = match.groups()[1:]
58: (8)                 self.min_row = min_row
59: (8)                 self.max_row = max_row
60: (4)             def __eq__(self, other):
61: (8)                 if isinstance(other, self.__class__):
62: (12)                    return (self.min_row == other.min_row
63: (20)                            and
64: (20)                            self.max_row == other.max_row)
65: (8)                 elif isinstance(other, str):
66: (12)                    return (str(self) == other
67: (20)                            or
68: (20)                            f"{self.min_row}:{self.max_row}")
69: (8)                 return False
70: (4)             def __repr__(self):
71: (8)                 return f"Range of rows from '{self.min_row}' to '{self.max_row}'"
72: (4)             def __str__(self):
73: (8)                 return f"${self.min_row}:${self.max_row}"
74: (0)         class PrintTitles(Strict):
75: (4)             """
76: (4)             Contains at least either a range of rows or columns
77: (4)             """
78: (4)             cols = Typed(expected_type=ColRange, allow_none=True)
79: (4)             rows = Typed(expected_type=RowRange, allow_none=True)
80: (4)             title = String()
81: (4)             def __init__(self, cols=None, rows=None, title=""):
82: (8)                 self.cols = cols
83: (8)                 self.rows = rows
84: (8)                 self.title = title
85: (4)             @classmethod
86: (4)             def from_string(cls, value):
87: (8)                 kw = dict((k, v) for match in TITLES_REGEX.finditer(value)
88: (18)                          for k, v in match.groupdict().items() if v)
89: (8)                 if not kw:
90: (12)                    raise ValueError(f"{value} is not a valid print titles
definition")
91: (8)                 cols = rows = None
92: (8)                 if "cols" in kw:
93: (12)                    cols = ColRange(kw["cols"])
94: (8)                 if "rows" in kw:
95: (12)                    rows = RowRange(kw["rows"])
96: (8)                 title = kw.get("quoted") or kw.get("notquoted")
97: (8)                 return cls(cols=cols, rows=rows, title=title)
98: (4)             def __eq__(self, other):
99: (8)                 if isinstance(other, self.__class__):
100: (12)                   return (self.cols == other.cols
101: (20)                           and
102: (20)                           self.rows == other.rows
103: (20)                           and
104: (20)                           self.title == other.title)
105: (8)                elif isinstance(other, str):
106: (12)                   return str(self) == other
107: (8)                return False
108: (4)            def __repr__(self):
109: (8)                return f"Print titles for sheet {self.title} cols {self.rows}, rows
{self.cols}"
110: (4)            def __str__(self):
111: (8)                title = quote_sheetname(self.title)
112: (8)                titles = ",".join([f"{title}!{value}" for value in (self.rows,
self.cols) if value])
113: (8)                return titles or ""
114: (0)        class PrintArea(MultiCellRange):
115: (4)            @classmethod
116: (4)            def from_string(cls, value):
117: (8)                new = []
118: (8)                for m in PRINT_AREA_RE.finditer(value): # can be multiple
119: (12)                   coord = m.group("cells")
120: (12)                   if coord:
```

```
121: (16)                       new.append(coord)
122: (8)                    return cls(new)
123: (4)                def __init__(self, ranges=(), title=""):
124: (8)                    self.title = ""
125: (8)                    super().__init__(ranges)
126: (4)                def __str__(self):
127: (8)                    if self.ranges:
128: (12)                       return ",".join([f"{quote_sheetname(self.title)}!
{absolute_coordinate(str(range))}"
129: (29)                                        for range in self.sorted()])
130: (8)                    return ""
131: (4)                def __eq__(self, other):
132: (8)                    super().__eq__(other)
133: (8)                    if isinstance(other, str):
134: (12)                       return str(self) == other

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

File 179 - related.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors.excel import Relation
3: (0)              class Related(Serialisable):
4: (4)                  id = Relation()
5: (4)                  def __init__(self, id=None):
6: (8)                      self.id = id
7: (4)                  def to_tree(self, tagname, idx=None):
8: (8)                      return super().to_tree(tagname)

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

File 180 - table.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Descriptor,
4: (4)                  Alias,
5: (4)                  Typed,
6: (4)                  Bool,
7: (4)                  Integer,
8: (4)                  NoneSet,
9: (4)                  String,
10: (4)                 Sequence,
11: (0)             )
12: (0)             from openpyxl.descriptors.excel import ExtensionList, CellRange
13: (0)             from openpyxl.descriptors.sequence import NestedSequence
14: (0)             from openpyxl.xml.constants import SHEET_MAIN_NS, REL_NS
15: (0)             from openpyxl.xml.functions import tostring
16: (0)             from openpyxl.utils import range_boundaries
17: (0)             from openpyxl.utils.escape import escape, unescape
18: (0)             from .related import Related
19: (0)             from .filters import (
20: (4)                 AutoFilter,
21: (4)                 SortState,
22: (0)             )
23: (0)             TABLESTYLES = tuple(
24: (4)                 ["TableStyleMedium{0}".format(i) for i in range(1, 29)]
25: (4)                 + ["TableStyleLight{0}".format(i) for i in range(1, 22)]
26: (4)                 + ["TableStyleDark{0}".format(i) for i in range(1, 12)]
27: (0)             )
28: (0)             PIVOTSTYLES = tuple(
29: (4)                 ["PivotStyleMedium{0}".format(i) for i in range(1, 29)]
30: (4)                 + ["PivotStyleLight{0}".format(i) for i in range(1, 29)]
31: (4)                 + ["PivotStyleDark{0}".format(i) for i in range(1, 29)]
32: (0)             )
33: (0)             class TableStyleInfo(Serialisable):
34: (4)                 tagname = "tableStyleInfo"
35: (4)                 name = String(allow_none=True)
36: (4)                 showFirstColumn = Bool(allow_none=True)
```

```
37: (4)                 showLastColumn = Bool(allow_none=True)
38: (4)                 showRowStripes = Bool(allow_none=True)
39: (4)                 showColumnStripes = Bool(allow_none=True)
40: (4)                 def __init__(self,
41: (17)                             name=None,
42: (17)                             showFirstColumn=None,
43: (17)                             showLastColumn=None,
44: (17)                             showRowStripes=None,
45: (17)                             showColumnStripes=None,
46: (16)                            ):
47: (8)                     self.name = name
48: (8)                     self.showFirstColumn = showFirstColumn
49: (8)                     self.showLastColumn = showLastColumn
50: (8)                     self.showRowStripes = showRowStripes
51: (8)                     self.showColumnStripes = showColumnStripes
52: (0)             class XMLColumnProps(Serialisable):
53: (4)                 tagname = "xmlColumnPr"
54: (4)                 mapId = Integer()
55: (4)                 xpath = String()
56: (4)                 denormalized = Bool(allow_none=True)
57: (4)                 xmlDataType = String()
58: (4)                 extLst = Typed(expected_type=ExtensionList, allow_none=True)
59: (4)                 __elements__ = ()
60: (4)                 def __init__(self,
61: (17)                             mapId=None,
62: (17)                             xpath=None,
63: (17)                             denormalized=None,
64: (17)                             xmlDataType=None,
65: (17)                             extLst=None,
66: (16)                            ):
67: (8)                     self.mapId = mapId
68: (8)                     self.xpath = xpath
69: (8)                     self.denormalized = denormalized
70: (8)                     self.xmlDataType = xmlDataType
71: (0)             class TableFormula(Serialisable):
72: (4)                 tagname = "tableFormula"
73: (4)                 array = Bool(allow_none=True)
74: (4)                 attr_text = Descriptor()
75: (4)                 text = Alias('attr_text')
76: (4)                 def __init__(self,
77: (17)                             array=None,
78: (17)                             attr_text=None,
79: (16)                            ):
80: (8)                     self.array = array
81: (8)                     self.attr_text = attr_text
82: (0)             class TableColumn(Serialisable):
83: (4)                 tagname = "tableColumn"
84: (4)                 id = Integer()
85: (4)                 uniqueName = String(allow_none=True)
86: (4)                 name = String()
87: (4)                 totalsRowFunction = NoneSet(values=(['sum', 'min', 'max', 'average',
88: (41)                                                         'count', 'countNums', 'stdDev',
'var', 'custom']))
89: (4)                 totalsRowLabel = String(allow_none=True)
90: (4)                 queryTableFieldId = Integer(allow_none=True)
91: (4)                 headerRowDxfId = Integer(allow_none=True)
92: (4)                 dataDxfId = Integer(allow_none=True)
93: (4)                 totalsRowDxfId = Integer(allow_none=True)
94: (4)                 headerRowCellStyle = String(allow_none=True)
95: (4)                 dataCellStyle = String(allow_none=True)
96: (4)                 totalsRowCellStyle = String(allow_none=True)
97: (4)                 calculatedColumnFormula = Typed(expected_type=TableFormula,
allow_none=True)
98: (4)                 totalsRowFormula = Typed(expected_type=TableFormula, allow_none=True)
99: (4)                 xmlColumnPr = Typed(expected_type=XMLColumnProps, allow_none=True)
100: (4)                extLst = Typed(expected_type=ExtensionList, allow_none=True)
101: (4)                __elements__ = ('calculatedColumnFormula', 'totalsRowFormula',
102: (20)                                'xmlColumnPr', 'extLst')
103: (4)                def __init__(self,
104: (17)                            id=None,
105: (17)                            uniqueName=None,
106: (17)                            name=None,
107: (17)                            totalsRowFunction=None,
108: (17)                            totalsRowLabel=None,
109: (17)                            queryTableFieldId=None,
110: (17)                            headerRowDxfId=None,
111: (17)                            dataDxfId=None,
112: (17)                            totalsRowDxfId=None,
113: (17)                            headerRowCellStyle=None,
114: (17)                            dataCellStyle=None,
115: (17)                            totalsRowCellStyle=None,
116: (17)                            calculatedColumnFormula=None,
117: (17)                            totalsRowFormula=None,
118: (17)                            xmlColumnPr=None,
119: (17)                            extLst=None,
120: (16)                           ):
121: (8)                    self.id = id
122: (8)                    self.uniqueName = uniqueName
123: (8)                    self.name = name
124: (8)                    self.totalsRowFunction = totalsRowFunction
125: (8)                    self.totalsRowLabel = totalsRowLabel
126: (8)                    self.queryTableFieldId = queryTableFieldId
127: (8)                    self.headerRowDxfId = headerRowDxfId
128: (8)                    self.dataDxfId = dataDxfId
129: (8)                    self.totalsRowDxfId = totalsRowDxfId
130: (8)                    self.headerRowCellStyle = headerRowCellStyle
131: (8)                    self.dataCellStyle = dataCellStyle
132: (8)                    self.totalsRowCellStyle = totalsRowCellStyle
133: (8)                    self.calculatedColumnFormula = calculatedColumnFormula
134: (8)                    self.totalsRowFormula = totalsRowFormula
135: (8)                    self.xmlColumnPr = xmlColumnPr
136: (8)                    self.extLst = extLst
137: (4)                def __iter__(self):
138: (8)                    for k, v in super().__iter__():
139: (12)                       if k == 'name':
140: (16)                           v = escape(v)
141: (12)                       yield k, v
142: (4)                @classmethod
143: (4)                def from_tree(cls, node):
144: (8)                    self = super().from_tree(node)
145: (8)                    self.name = unescape(self.name)
146: (8)                    return self
147: (0)            class TableNameDescriptor(String):
148: (4)                """
149: (4)                Table names cannot have spaces in them
150: (4)                """
151: (4)                def __set__(self, instance, value):
152: (8)                    if value is not None and " " in value:
153: (12)                       raise ValueError("Table names cannot have spaces")
154: (8)                    super().__set__(instance, value)
155: (0)            class Table(Serialisable):
156: (4)                _path = "/tables/table{0}.xml"
157: (4)                mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.table+xml"
158: (4)                _rel_type = REL_NS + "/table"
159: (4)                _rel_id = None
160: (4)                tagname = "table"
161: (4)                id = Integer()
162: (4)                name = String(allow_none=True)
163: (4)                displayName = TableNameDescriptor()
164: (4)                comment = String(allow_none=True)
165: (4)                ref = CellRange()
166: (4)                tableType = NoneSet(values=(['worksheet', 'xml', 'queryTable']))
167: (4)                headerRowCount = Integer(allow_none=True)
168: (4)                insertRow = Bool(allow_none=True)
169: (4)                insertRowShift = Bool(allow_none=True)
170: (4)                totalsRowCount = Integer(allow_none=True)
171: (4)                totalsRowShown = Bool(allow_none=True)
```

```
    published = Bool(allow_none=True)
    headerRowDxfId = Integer(allow_none=True)
    dataDxfId = Integer(allow_none=True)
    totalsRowDxfId = Integer(allow_none=True)
    headerRowBorderDxfId = Integer(allow_none=True)
    tableBorderDxfId = Integer(allow_none=True)
    totalsRowBorderDxfId = Integer(allow_none=True)
    headerRowCellStyle = String(allow_none=True)
    dataCellStyle = String(allow_none=True)
    totalsRowCellStyle = String(allow_none=True)
    connectionId = Integer(allow_none=True)
    autoFilter = Typed(expected_type=AutoFilter, allow_none=True)
    sortState = Typed(expected_type=SortState, allow_none=True)
    tableColumns = NestedSequence(expected_type=TableColumn, count=True)
    tableStyleInfo = Typed(expected_type=TableStyleInfo, allow_none=True)
    extLst = Typed(expected_type=ExtensionList, allow_none=True)
    __elements__ = ('autoFilter', 'sortState', 'tableColumns',
                    'tableStyleInfo')
    def __init__(self,
                 id=1,
                 displayName=None,
                 ref=None,
                 name=None,
                 comment=None,
                 tableType=None,
                 headerRowCount=1,
                 insertRow=None,
                 insertRowShift=None,
                 totalsRowCount=None,
                 totalsRowShown=None,
                 published=None,
                 headerRowDxfId=None,
                 dataDxfId=None,
                 totalsRowDxfId=None,
                 headerRowBorderDxfId=None,
                 tableBorderDxfId=None,
                 totalsRowBorderDxfId=None,
                 headerRowCellStyle=None,
                 dataCellStyle=None,
                 totalsRowCellStyle=None,
                 connectionId=None,
                 autoFilter=None,
                 sortState=None,
                 tableColumns=(),
                 tableStyleInfo=None,
                 extLst=None,
                ):
        self.id = id
        self.displayName = displayName
        if name is None:
            name = displayName
        self.name = name
        self.comment = comment
        self.ref = ref
        self.tableType = tableType
        self.headerRowCount = headerRowCount
        self.insertRow = insertRow
        self.insertRowShift = insertRowShift
        self.totalsRowCount = totalsRowCount
        self.totalsRowShown = totalsRowShown
        self.published = published
        self.headerRowDxfId = headerRowDxfId
        self.dataDxfId = dataDxfId
        self.totalsRowDxfId = totalsRowDxfId
        self.headerRowBorderDxfId = headerRowBorderDxfId
        self.tableBorderDxfId = tableBorderDxfId
        self.totalsRowBorderDxfId = totalsRowBorderDxfId
        self.headerRowCellStyle = headerRowCellStyle
        self.dataCellStyle = dataCellStyle
        self.totalsRowCellStyle = totalsRowCellStyle
        self.connectionId = connectionId
        self.autoFilter = autoFilter
        self.sortState = sortState
        self.tableColumns = tableColumns
        self.tableStyleInfo = tableStyleInfo
    def to_tree(self):
        tree = super().to_tree()
        tree.set("xmlns", SHEET_MAIN_NS)
        return tree
    @property
    def path(self):
        """
        Return path within the archive
        """
        return "/xl" + self._path.format(self.id)
    def _write(self, archive):
        """
        Serialise to XML and write to archive
        """
        xml = self.to_tree()
        archive.writestr(self.path[1:], tostring(xml))
    def _initialise_columns(self):
        """
        Create a list of table columns from a cell range
        Always set a ref if we have headers (the default)
        Column headings must be strings and must match cells in the worksheet.
        """
        min_col, min_row, max_col, max_row = range_boundaries(self.ref)
        for idx in range(min_col, max_col+1):
            col = TableColumn(id=idx, name="Column{0}".format(idx))
            self.tableColumns.append(col)
        if self.headerRowCount and not self.autoFilter:
            self.autoFilter = AutoFilter(ref=self.ref)
    @property
    def column_names(self):
        return [column.name for column in self.tableColumns]
class TablePartList(Serialisable):
    tagname = "tableParts"
    count = Integer(allow_none=True)
    tablePart = Sequence(expected_type=Related)
    __elements__ = ('tablePart',)
    __attrs__ = ('count',)
    def __init__(self,
                 count=None,
                 tablePart=(),
                ):
        self.tablePart = tablePart
    def append(self, part):
        self.tablePart.append(part)
    @property
    def count(self):
        return len(self.tablePart)
    def __bool__(self):
        return bool(self.tablePart)
class TableList(dict):
    def add(self, table):
        if not isinstance(table, Table):
            raise TypeError("You can only add tables")
        self[table.name] = table
    def get(self, name=None, table_range=None):
        if name is not None:
            return super().get(name)
        for table in self.values():
            if table_range == table.ref:
                return table
    def items(self):
        return [(name, table.ref) for name, table in super().items()]
```

```
----------------------------------------

File 181 - views.py:

(0)             from openpyxl.descriptors import (
(4)                 Bool,
(4)                 Integer,
(4)                 String,
(4)                 Set,
(4)                 Float,
(4)                 Typed,
(4)                 NoneSet,
(4)                 Sequence,
(0)             )
(0)             from openpyxl.descriptors.excel import ExtensionList
(0)             from openpyxl.descriptors.serialisable import Serialisable
(0)             class Pane(Serialisable):
(4)                 xSplit = Float(allow_none=True)
(4)                 ySplit = Float(allow_none=True)
(4)                 topLeftCell = String(allow_none=True)
(4)                 activePane = Set(values=("bottomRight", "topRight", "bottomLeft", "topLeft"))
(4)                 state = Set(values=("split", "frozen", "frozenSplit"))
(4)                 def __init__(self,
(17)                             xSplit=None,
(17)                             ySplit=None,
(17)                             topLeftCell=None,
(17)                             activePane="topLeft",
(17)                             state="split"):
(8)                     self.xSplit = xSplit
(8)                     self.ySplit = ySplit
(8)                     self.topLeftCell = topLeftCell
(8)                     self.activePane = activePane
(8)                     self.state = state
(0)             class Selection(Serialisable):
(4)                 pane = NoneSet(values=("bottomRight", "topRight", "bottomLeft", "topLeft"))
(4)                 activeCell = String(allow_none=True)
(4)                 activeCellId = Integer(allow_none=True)
(4)                 sqref = String(allow_none=True)
(4)                 def __init__(self,
(17)                             pane=None,
(17)                             activeCell="A1",
(17)                             activeCellId=None,
(17)                             sqref="A1"):
(8)                     self.pane = pane
(8)                     self.activeCell = activeCell
(8)                     self.activeCellId = activeCellId
(8)                     self.sqref = sqref
(0)             class SheetView(Serialisable):
(4)                 """Information about the visible portions of this sheet."""
(4)                 tagname = "sheetView"
(4)                 windowProtection = Bool(allow_none=True)
(4)                 showFormulas = Bool(allow_none=True)
(4)                 showGridLines = Bool(allow_none=True)
(4)                 showRowColHeaders = Bool(allow_none=True)
(4)                 showZeros = Bool(allow_none=True)
(4)                 rightToLeft = Bool(allow_none=True)
(4)                 tabSelected = Bool(allow_none=True)
(4)                 showRuler = Bool(allow_none=True)
(4)                 showOutlineSymbols = Bool(allow_none=True)
(4)                 defaultGridColor = Bool(allow_none=True)
(4)                 showWhiteSpace = Bool(allow_none=True)
(4)                 view = NoneSet(values=("normal", "pageBreakPreview", "pageLayout"))
(4)                 topLeftCell = String(allow_none=True)
(4)                 colorId = Integer(allow_none=True)
(4)                 zoomScale = Integer(allow_none=True)
(4)                 zoomScaleNormal = Integer(allow_none=True)
(4)                 zoomScaleSheetLayoutView = Integer(allow_none=True)
```

```
(4)       zoomScalePageLayoutView = Integer(allow_none=True)
(4)       zoomToFit = Bool(allow_none=True) # Chart sheets only
(4)       workbookViewId = Integer()
(4)       selection = Sequence(expected_type=Selection)
(4)       pane = Typed(expected_type=Pane, allow_none=True)
(4)       def __init__(self,
(17)                   windowProtection=None,
(17)                   showFormulas=None,
(17)                   showGridLines=None,
(17)                   showRowColHeaders=None,
(17)                   showZeros=None,
(17)                   rightToLeft=None,
(17)                   tabSelected=None,
(17)                   showRuler=None,
(17)                   showOutlineSymbols=None,
(17)                   defaultGridColor=None,
(17)                   showWhiteSpace=None,
(17)                   view=None,
(17)                   topLeftCell=None,
(17)                   colorId=None,
(17)                   zoomScale=None,
(17)                   zoomScaleNormal=None,
(17)                   zoomScaleSheetLayoutView=None,
(17)                   zoomScalePageLayoutView=None,
(17)                   zoomToFit=None,
(17)                   workbookViewId=0,
(17)                   selection=None,
(17)                   pane=None,):
(8)           self.windowProtection = windowProtection
(8)           self.showFormulas = showFormulas
(8)           self.showGridLines = showGridLines
(8)           self.showRowColHeaders = showRowColHeaders
(8)           self.showZeros = showZeros
(8)           self.rightToLeft = rightToLeft
(8)           self.tabSelected = tabSelected
(8)           self.showRuler = showRuler
(8)           self.showOutlineSymbols = showOutlineSymbols
(8)           self.defaultGridColor = defaultGridColor
(8)           self.showWhiteSpace = showWhiteSpace
(8)           self.view = view
(8)           self.topLeftCell = topLeftCell
(8)           self.colorId = colorId
(8)           self.zoomScale = zoomScale
(8)           self.zoomScaleNormal = zoomScaleNormal
(8)           self.zoomScaleSheetLayoutView = zoomScaleSheetLayoutView
(8)           self.zoomScalePageLayoutView = zoomScalePageLayoutView
(8)           self.zoomToFit = zoomToFit
(8)           self.workbookViewId = workbookViewId
(8)           self.pane = pane
(8)           if selection is None:
(12)              selection = (Selection(), )
(8)           self.selection = selection
(0)   class SheetViewList(Serialisable):
(4)       tagname = "sheetViews"
(4)       sheetView = Sequence(expected_type=SheetView, )
(4)       extLst = Typed(expected_type=ExtensionList, allow_none=True)
(4)       __elements__ = ('sheetView',)
(4)       def __init__(self,
(17)                   sheetView=None,
(17)                   extLst=None,
(16)                  ):
(8)           if sheetView is None:
(12)              sheetView = [SheetView()]
(8)           self.sheetView = sheetView
(4)       @property
(4)       def active(self):
(8)           """
(8)           Returns the first sheet view which is assumed to be active
(8)           """
```

```
133: (8)                    return self.sheetView[0]

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

File 182 - excel.py:

1: (0)              import datetime
2: (0)              import re
3: (0)              from zipfile import ZipFile, ZIP_DEFLATED
4: (0)              from openpyxl.utils.exceptions import InvalidFileException
5: (0)              from openpyxl.xml.constants import (
6: (4)                  ARC_ROOT_RELS,
7: (4)                  ARC_WORKBOOK_RELS,
8: (4)                  ARC_APP,
9: (4)                  ARC_CORE,
10: (4)                 ARC_CUSTOM,
11: (4)                 CPROPS_TYPE,
12: (4)                 ARC_THEME,
13: (4)                 ARC_STYLE,
14: (4)                 ARC_WORKBOOK,
15: (4)                 )
16: (0)             from openpyxl.drawing.spreadsheet_drawing import SpreadsheetDrawing
17: (0)             from openpyxl.xml.functions import tostring, fromstring
18: (0)             from openpyxl.packaging.manifest import Manifest
19: (0)             from openpyxl.packaging.relationship import (
20: (4)                 get_rels_path,
21: (4)                 RelationshipList,
22: (4)                 Relationship,
23: (0)             )
24: (0)             from openpyxl.comments.comment_sheet import CommentSheet
25: (0)             from openpyxl.styles.stylesheet import write_stylesheet
26: (0)             from openpyxl.worksheet._writer import WorksheetWriter
27: (0)             from openpyxl.workbook._writer import WorkbookWriter
28: (0)             from .theme import theme_xml
29: (0)             class ExcelWriter:
30: (4)                 """Write a workbook object to an Excel file."""
31: (4)                 def __init__(self, workbook, archive):
32: (8)                     self._archive = archive
33: (8)                     self.workbook = workbook
34: (8)                     self.manifest = Manifest()
35: (8)                     self.vba_modified = set()
36: (8)                     self._tables = []
37: (8)                     self._charts = []
38: (8)                     self._images = []
39: (8)                     self._drawings = []
40: (8)                     self._comments = []
41: (8)                     self._pivots = []
42: (4)                 def write_data(self):
43: (8)                     from openpyxl.packaging.extended import ExtendedProperties
44: (8)                     """Write the various xml files into the zip archive."""
45: (8)                     archive = self._archive
46: (8)                     props = ExtendedProperties()
47: (8)                     archive.writestr(ARC_APP, tostring(props.to_tree()))
48: (8)                     archive.writestr(ARC_CORE,
tostring(self.workbook.properties.to_tree()))
49: (8)                     if self.workbook.loaded_theme:
50: (12)                        archive.writestr(ARC_THEME, self.workbook.loaded_theme)
51: (8)                     else:
52: (12)                        archive.writestr(ARC_THEME, theme_xml)
53: (8)                     if len(self.workbook.custom_doc_props) >= 1:
54: (12)                        archive.writestr(ARC_CUSTOM,
tostring(self.workbook.custom_doc_props.to_tree()))
55: (12)                        class CustomOverride():
56: (16)                            path = "/" + ARC_CUSTOM #PartName
57: (16)                            mime_type = CPROPS_TYPE #ContentType
58: (12)                        custom_override = CustomOverride()
59: (12)                        self.manifest.append(custom_override)
60: (8)                     self._write_worksheets()
61: (8)                     self._write_chartsheets()
```

```
62: (8)                     self._write_images()
63: (8)                     self._write_charts()
64: (8)                     self._write_external_links()
65: (8)                     stylesheet = write_stylesheet(self.workbook)
66: (8)                     archive.writestr(ARC_STYLE, tostring(stylesheet))
67: (8)                     writer = WorkbookWriter(self.workbook)
68: (8)                     archive.writestr(ARC_ROOT_RELS, writer.write_root_rels())
69: (8)                     archive.writestr(ARC_WORKBOOK, writer.write())
70: (8)                     archive.writestr(ARC_WORKBOOK_RELS, writer.write_rels())
71: (8)                     self._merge_vba()
72: (8)                     self.manifest._write(archive, self.workbook)
73: (4)                 def _merge_vba(self):
74: (8)                     """
75: (8)                     If workbook contains macros then extract associated files from cache
76: (8)                     of old file and add to archive
77: (8)                     """
78: (8)                     ARC_VBA = re.compile("|".join(
79: (12)                        ('xl/vba', r'xl/drawings/.*vmlDrawing\d\.vml',
80: (13)                         'xl/ctrlProps', 'customUI', 'xl/activeX', r'xl/media/.*\.emf')
81: (8)                     )
82: (29)                                         )
83: (8)                     if self.workbook.vba_archive:
84: (12)                        for name in set(self.workbook.vba_archive.namelist()) -
self.vba_modified:
85: (16)                            if ARC_VBA.match(name):
86: (20)                                self._archive.writestr(name,
self.workbook.vba_archive.read(name))
87: (4)                 def _write_images(self):
88: (8)                     for img in self._images:
89: (12)                        self._archive.writestr(img.path[1:], img._data())
90: (4)                 def _write_charts(self):
91: (8)                     if len(self._charts) != len(set(self._charts)):
92: (12)                        raise InvalidFileException("The same chart cannot be used in more
than one worksheet")
93: (8)                     for chart in self._charts:
94: (12)                        self._archive.writestr(chart.path[1:], tostring(chart._write()))
95: (12)                        self.manifest.append(chart)
96: (4)                 def _write_drawing(self, drawing):
97: (8)                     """
98: (8)                     Write a drawing
99: (8)                     """
100: (8)                    self._drawings.append(drawing)
101: (8)                    drawing._id = len(self._drawings)
102: (8)                    for chart in drawing.charts:
103: (12)                       self._charts.append(chart)
104: (12)                       chart._id = len(self._charts)
105: (8)                    for img in drawing.images:
106: (12)                       self._images.append(img)
107: (12)                       img._id = len(self._images)
108: (8)                    rels_path = get_rels_path(drawing.path)[1:]
109: (8)                    self._archive.writestr(drawing.path[1:], tostring(drawing._write()))
110: (8)                    self._archive.writestr(rels_path, tostring(drawing._write_rels()))
111: (8)                    self.manifest.append(drawing)
112: (4)                def _write_chartsheets(self):
113: (8)                    for idx, sheet in enumerate(self.workbook.chartsheets, 1):
114: (12)                       sheet._id = idx
115: (12)                       xml = tostring(sheet.to_tree())
116: (12)                       self._archive.writestr(sheet.path[1:], xml)
117: (12)                       self.manifest.append(sheet)
118: (12)                       if sheet._drawing:
119: (16)                           self._write_drawing(sheet._drawing)
120: (16)                           rel = Relationship(type="drawing", Target=sheet._drawing.path)
121: (16)                           rels = RelationshipList()
122: (16)                           rels.append(rel)
123: (16)                           tree = rels.to_tree()
124: (16)                           rels_path = get_rels_path(sheet.path[1:])
125: (16)                           self._archive.writestr(rels_path, tostring(tree))
126: (4)                def _write_comment(self, ws):
127: (8)                    cs = CommentSheet.from_comments(ws._comments)
```

```
128: (8)                    self._comments.append(cs)
129: (8)                    cs._id = len(self._comments)
130: (8)                    self._archive.writestr(cs.path[1:], tostring(cs.to_tree()))
131: (8)                    self.manifest.append(cs)
132: (8)                    if ws.legacy_drawing is None or self.workbook.vba_archive is None:
133: (12)                       ws.legacy_drawing =
'xl/drawings/commentsDrawing{0}.vml'.format(cs._id)
134: (12)                       vml = None
135: (8)                    else:
136: (12)                       vml =
fromstring(self.workbook.vba_archive.read(ws.legacy_drawing))
137: (8)                    vml = cs.write_shapes(vml)
138: (8)                    self._archive.writestr(ws.legacy_drawing, vml)
139: (8)                    self.vba_modified.add(ws.legacy_drawing)
140: (8)                    comment_rel = Relationship(Id="comments", type=cs._rel_type,
Target=cs.path)
141: (8)                    ws._rels.append(comment_rel)
142: (4)                def write_worksheet(self, ws):
143: (8)                    ws._drawing = SpreadsheetDrawing()
144: (8)                    ws._drawing.charts = ws._charts
145: (8)                    ws._drawing.images = ws._images
146: (8)                    if self.workbook.write_only:
147: (12)                       if not ws.closed:
148: (16)                           ws.close()
149: (12)                       writer = ws._writer
150: (8)                    else:
151: (12)                       writer = WorksheetWriter(ws)
152: (12)                       writer.write()
153: (8)                    ws._rels = writer._rels
154: (8)                    self._archive.write(writer.out, ws.path[1:])
155: (8)                    self.manifest.append(ws)
156: (8)                    writer.cleanup()
157: (4)                def _write_worksheets(self):
158: (8)                    pivot_caches = set()
159: (8)                    for idx, ws in enumerate(self.workbook.worksheets, 1):
160: (12)                       ws._id = idx
161: (12)                       self.write_worksheet(ws)
162: (12)                       if ws._drawing:
163: (16)                           self._write_drawing(ws._drawing)
164: (16)                           for r in ws._rels:
165: (20)                               if "drawing" in r.Type:
166: (24)                                   r.Target = ws._drawing.path
167: (12)                       if ws._comments:
168: (16)                           self._write_comment(ws)
169: (12)                       if ws.legacy_drawing is not None:
170: (16)                           shape_rel = Relationship(type="vmlDrawing", Id="anysvml",
171: (41)                                                    Target="/" + ws.legacy_drawing)
172: (16)                           ws._rels.append(shape_rel)
173: (12)                       for t in ws._tables.values():
174: (16)                           self._tables.append(t)
175: (16)                           t.id = len(self._tables)
176: (16)                           t._write(self._archive)
177: (16)                           self.manifest.append(t)
178: (16)                           ws._rels.get(t._rel_id).Target = t.path
179: (12)                       for p in ws._pivots:
180: (16)                           if p.cache not in pivot_caches:
181: (20)                               pivot_caches.add(p.cache)
182: (20)                               p.cache._id = len(pivot_caches)
183: (16)                           self._pivots.append(p)
184: (16)                           p._id = len(self._pivots)
185: (16)                           p._write(self._archive, self.manifest)
186: (16)                           self.workbook._pivots.append(p)
187: (16)                           r = Relationship(Type=p.rel_type, Target=p.path)
188: (16)                           ws._rels.append(r)
189: (12)                       if ws._rels:
190: (16)                           tree = ws._rels.to_tree()
191: (16)                           rels_path = get_rels_path(ws.path)[1:]
192: (16)                           self._archive.writestr(rels_path, tostring(tree))
193: (4)                def _write_external_links(self):
```

```
194: (8)                    """Write links to external workbooks"""
195: (8)                    wb = self.workbook
196: (8)                    for idx, link in enumerate(wb._external_links, 1):
197: (12)                       link._id = idx
198: (12)                       rels_path = get_rels_path(link.path[1:])
199: (12)                       xml = link.to_tree()
200: (12)                       self._archive.writestr(link.path[1:], tostring(xml))
201: (12)                       rels = RelationshipList()
202: (12)                       rels.append(link.file_link)
203: (12)                       self._archive.writestr(rels_path, tostring(rels.to_tree()))
204: (12)                       self.manifest.append(link)
205: (4)                def save(self):
206: (8)                    """Write data into the archive."""
207: (8)                    self.write_data()
208: (8)                    self._archive.close()
209: (0)            def save_workbook(workbook, filename):
210: (4)                """Save the given workbook on the filesystem under the name filename.
211: (4)                :param workbook: the workbook to save
212: (4)                :type workbook: :class:`openpyxl.workbook.Workbook`
213: (4)                :param filename: the path to which save the workbook
214: (4)                :type filename: string
215: (4)                :rtype: bool
216: (4)                """
217: (4)                archive = ZipFile(filename, 'w', ZIP_DEFLATED, allowZip64=True)
218: (4)                workbook.properties.modified =
datetime.datetime.now(tz=datetime.timezone.utc).replace(tzinfo=None)
219: (4)                writer = ExcelWriter(workbook, archive)
220: (4)                writer.save()
221: (4)                return True

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

File 183 - theme.py:

1: (0)              """Write the theme xml based on a fixed string."""
2: (0)              theme_xml = """<?xml version="1.0"?>
3: (0)              <a:theme xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
name="Office Theme">
4: (2)                <a:themeElements>
5: (4)                  <a:clrScheme name="Office">
6: (6)                    <a:dk1>
7: (8)                      <a:sysClr val="windowText" lastClr="000000"/>
8: (6)                    </a:dk1>
9: (6)                    <a:lt1>
10: (8)                     <a:sysClr val="window" lastClr="FFFFFF"/>
11: (6)                   </a:lt1>
12: (6)                   <a:dk2>
13: (8)                     <a:srgbClr val="1F497D"/>
14: (6)                   </a:dk2>
15: (6)                   <a:lt2>
16: (8)                     <a:srgbClr val="EEECE1"/>
17: (6)                   </a:lt2>
18: (6)                   <a:accent1>
19: (8)                     <a:srgbClr val="4F81BD"/>
20: (6)                   </a:accent1>
21: (6)                   <a:accent2>
22: (8)                     <a:srgbClr val="C0504D"/>
23: (6)                   </a:accent2>
24: (6)                   <a:accent3>
25: (8)                     <a:srgbClr val="9BBB59"/>
26: (6)                   </a:accent3>
27: (6)                   <a:accent4>
28: (8)                     <a:srgbClr val="8064A2"/>
29: (6)                   </a:accent4>
30: (6)                   <a:accent5>
31: (8)                     <a:srgbClr val="4BACC6"/>
32: (6)                   </a:accent5>
33: (6)                   <a:accent6>
34: (8)                     <a:srgbClr val="F79646"/>
```

```
35: (6)                   </a:accent6>
36: (6)                   <a:hlink>
37: (8)                     <a:srgbClr val="0000FF"/>
38: (6)                   </a:hlink>
39: (6)                   <a:folHlink>
40: (8)                     <a:srgbClr val="800080"/>
41: (6)                   </a:folHlink>
42: (4)                 </a:clrScheme>
43: (4)                 <a:fontScheme name="Office">
44: (6)                   <a:majorFont>
45: (8)                     <a:latin typeface="Cambria"/>
46: (8)                     <a:ea typeface=""/>
47: (8)                     <a:cs typeface=""/>
48: (8)                     <a:font script="Jpan" typeface="&#xFF2D;&#xFF33;
&#xFF30;&#x30B4;&#x30B7;&#x30C3;&#x30AF;"/>
49: (8)                     <a:font script="Hang" typeface="&#xB9D1;&#xC740; &#xACE0;&#xB515;"/>
50: (8)                     <a:font script="Hans" typeface="&#x5B8B;&#x4F53;"/>
51: (8)                     <a:font script="Hant" typeface="&#x65B0;&#x7D30;&#x660E;&#x9AD4;"/>
52: (8)                     <a:font script="Arab" typeface="Times New Roman"/>
53: (8)                     <a:font script="Hebr" typeface="Times New Roman"/>
54: (8)                     <a:font script="Thai" typeface="Tahoma"/>
55: (8)                     <a:font script="Ethi" typeface="Nyala"/>
56: (8)                     <a:font script="Beng" typeface="Vrinda"/>
57: (8)                     <a:font script="Gujr" typeface="Shruti"/>
58: (8)                     <a:font script="Khmr" typeface="MoolBoran"/>
59: (8)                     <a:font script="Knda" typeface="Tunga"/>
60: (8)                     <a:font script="Guru" typeface="Raavi"/>
61: (8)                     <a:font script="Cans" typeface="Euphemia"/>
62: (8)                     <a:font script="Cher" typeface="Plantagenet Cherokee"/>
63: (8)                     <a:font script="Yiii" typeface="Microsoft Yi Baiti"/>
64: (8)                     <a:font script="Tibt" typeface="Microsoft Himalaya"/>
65: (8)                     <a:font script="Thaa" typeface="MV Boli"/>
66: (8)                     <a:font script="Deva" typeface="Mangal"/>
67: (8)                     <a:font script="Telu" typeface="Gautami"/>
68: (8)                     <a:font script="Taml" typeface="Latha"/>
69: (8)                     <a:font script="Syrc" typeface="Estrangelo Edessa"/>
70: (8)                     <a:font script="Orya" typeface="Kalinga"/>
71: (8)                     <a:font script="Mlym" typeface="Kartika"/>
72: (8)                     <a:font script="Laoo" typeface="DokChampa"/>
73: (8)                     <a:font script="Sinh" typeface="Iskoola Pota"/>
74: (8)                     <a:font script="Mong" typeface="Mongolian Baiti"/>
75: (8)                     <a:font script="Viet" typeface="Times New Roman"/>
76: (8)                     <a:font script="Uigh" typeface="Microsoft Uighur"/>
77: (6)                   </a:majorFont>
78: (6)                   <a:minorFont>
79: (8)                     <a:latin typeface="Calibri"/>
80: (8)                     <a:ea typeface=""/>
81: (8)                     <a:cs typeface=""/>
82: (8)                     <a:font script="Jpan" typeface="&#xFF2D;&#xFF33;
&#xFF30;&#x30B4;&#x30B7;&#x30C3;&#x30AF;"/>
83: (8)                     <a:font script="Hang" typeface="&#xB9D1;&#xC740; &#xACE0;&#xB515;"/>
84: (8)                     <a:font script="Hans" typeface="&#x5B8B;&#x4F53;"/>
85: (8)                     <a:font script="Hant" typeface="&#x65B0;&#x7D30;&#x660E;&#x9AD4;"/>
86: (8)                     <a:font script="Arab" typeface="Arial"/>
87: (8)                     <a:font script="Hebr" typeface="Arial"/>
88: (8)                     <a:font script="Thai" typeface="Tahoma"/>
89: (8)                     <a:font script="Ethi" typeface="Nyala"/>
90: (8)                     <a:font script="Beng" typeface="Vrinda"/>
91: (8)                     <a:font script="Gujr" typeface="Shruti"/>
92: (8)                     <a:font script="Khmr" typeface="DaunPenh"/>
93: (8)                     <a:font script="Knda" typeface="Tunga"/>
94: (8)                     <a:font script="Guru" typeface="Raavi"/>
95: (8)                     <a:font script="Cans" typeface="Euphemia"/>
96: (8)                     <a:font script="Cher" typeface="Plantagenet Cherokee"/>
97: (8)                     <a:font script="Yiii" typeface="Microsoft Yi Baiti"/>
98: (8)                     <a:font script="Tibt" typeface="Microsoft Himalaya"/>
99: (8)                     <a:font script="Thaa" typeface="MV Boli"/>
100: (8)                    <a:font script="Deva" typeface="Mangal"/>
101: (8)                    <a:font script="Telu" typeface="Gautami"/>
```

```
(8)                 <a:font script="Taml" typeface="Latha"/>
(8)                 <a:font script="Syrc" typeface="Estrangelo Edessa"/>
(8)                 <a:font script="Orya" typeface="Kalinga"/>
(8)                 <a:font script="Mlym" typeface="Kartika"/>
(8)                 <a:font script="Laoo" typeface="DokChampa"/>
(8)                 <a:font script="Sinh" typeface="Iskoola Pota"/>
(8)                 <a:font script="Mong" typeface="Mongolian Baiti"/>
(8)                 <a:font script="Viet" typeface="Arial"/>
(8)                 <a:font script="Uigh" typeface="Microsoft Uighur"/>
(6)               </a:minorFont>
(4)             </a:fontScheme>
(4)             <a:fmtScheme name="Office">
(6)               <a:fillStyleLst>
(8)                 <a:solidFill>
(10)                  <a:schemeClr val="phClr"/>
(8)                 </a:solidFill>
(8)                 <a:gradFill rotWithShape="1">
(10)                  <a:gsLst>
(12)                    <a:gs pos="0">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:tint val="50000"/>
(16)                        <a:satMod val="300000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(12)                    <a:gs pos="35000">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:tint val="37000"/>
(16)                        <a:satMod val="300000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(12)                    <a:gs pos="100000">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:tint val="15000"/>
(16)                        <a:satMod val="350000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(10)                  </a:gsLst>
(10)                  <a:lin ang="16200000" scaled="1"/>
(8)                 </a:gradFill>
(8)                 <a:gradFill rotWithShape="1">
(10)                  <a:gsLst>
(12)                    <a:gs pos="0">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:shade val="51000"/>
(16)                        <a:satMod val="130000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(12)                    <a:gs pos="80000">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:shade val="93000"/>
(16)                        <a:satMod val="130000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(12)                    <a:gs pos="100000">
(14)                      <a:schemeClr val="phClr">
(16)                        <a:shade val="94000"/>
(16)                        <a:satMod val="135000"/>
(14)                      </a:schemeClr>
(12)                    </a:gs>
(10)                  </a:gsLst>
(10)                  <a:lin ang="16200000" scaled="0"/>
(8)                 </a:gradFill>
(6)               </a:fillStyleLst>
(6)               <a:lnStyleLst>
(8)                 <a:ln w="9525" cap="flat" cmpd="sng" algn="ctr">
(10)                  <a:solidFill>
(12)                    <a:schemeClr val="phClr">
(14)                      <a:shade val="95000"/>
(14)                      <a:satMod val="105000"/>
```

```
171: (12)                         </a:schemeClr>
172: (10)                       </a:solidFill>
173: (10)                       <a:prstDash val="solid"/>
174: (8)                      </a:ln>
175: (8)                      <a:ln w="25400" cap="flat" cmpd="sng" algn="ctr">
176: (10)                       <a:solidFill>
177: (12)                         <a:schemeClr val="phClr"/>
178: (10)                       </a:solidFill>
179: (10)                       <a:prstDash val="solid"/>
180: (8)                      </a:ln>
181: (8)                      <a:ln w="38100" cap="flat" cmpd="sng" algn="ctr">
182: (10)                       <a:solidFill>
183: (12)                         <a:schemeClr val="phClr"/>
184: (10)                       </a:solidFill>
185: (10)                       <a:prstDash val="solid"/>
186: (8)                      </a:ln>
187: (6)                    </a:lnStyleLst>
188: (6)                    <a:effectStyleLst>
189: (8)                      <a:effectStyle>
190: (10)                       <a:effectLst>
191: (12)                         <a:outerShdw blurRad="40000" dist="20000" dir="5400000"
rotWithShape="0">
192: (14)                           <a:srgbClr val="000000">
193: (16)                             <a:alpha val="38000"/>
194: (14)                           </a:srgbClr>
195: (12)                         </a:outerShdw>
196: (10)                       </a:effectLst>
197: (8)                      </a:effectStyle>
198: (8)                      <a:effectStyle>
199: (10)                       <a:effectLst>
200: (12)                         <a:outerShdw blurRad="40000" dist="23000" dir="5400000"
rotWithShape="0">
201: (14)                           <a:srgbClr val="000000">
202: (16)                             <a:alpha val="35000"/>
203: (14)                           </a:srgbClr>
204: (12)                         </a:outerShdw>
205: (10)                       </a:effectLst>
206: (8)                      </a:effectStyle>
207: (8)                      <a:effectStyle>
208: (10)                       <a:effectLst>
209: (12)                         <a:outerShdw blurRad="40000" dist="23000" dir="5400000"
rotWithShape="0">
210: (14)                           <a:srgbClr val="000000">
211: (16)                             <a:alpha val="35000"/>
212: (14)                           </a:srgbClr>
213: (12)                         </a:outerShdw>
214: (10)                       </a:effectLst>
215: (10)                       <a:scene3d>
216: (12)                         <a:camera prst="orthographicFront">
217: (14)                           <a:rot lat="0" lon="0" rev="0"/>
218: (12)                         </a:camera>
219: (12)                         <a:lightRig rig="threePt" dir="t">
220: (14)                           <a:rot lat="0" lon="0" rev="1200000"/>
221: (12)                         </a:lightRig>
222: (10)                       </a:scene3d>
223: (10)                       <a:sp3d>
224: (12)                         <a:bevelT w="63500" h="25400"/>
225: (10)                       </a:sp3d>
226: (8)                      </a:effectStyle>
227: (6)                    </a:effectStyleLst>
228: (6)                    <a:bgFillStyleLst>
229: (8)                      <a:solidFill>
230: (10)                       <a:schemeClr val="phClr"/>
231: (8)                      </a:solidFill>
232: (8)                      <a:gradFill rotWithShape="1">
233: (10)                       <a:gsLst>
234: (12)                         <a:gs pos="0">
235: (14)                           <a:schemeClr val="phClr">
236: (16)                             <a:tint val="40000"/>
```

```
237: (16)                          <a:satMod val="350000"/>
238: (14)                        </a:schemeClr>
239: (12)                      </a:gs>
240: (12)                      <a:gs pos="40000">
241: (14)                        <a:schemeClr val="phClr">
242: (16)                          <a:tint val="45000"/>
243: (16)                          <a:shade val="99000"/>
244: (16)                          <a:satMod val="350000"/>
245: (14)                        </a:schemeClr>
246: (12)                      </a:gs>
247: (12)                      <a:gs pos="100000">
248: (14)                        <a:schemeClr val="phClr">
249: (16)                          <a:shade val="20000"/>
250: (16)                          <a:satMod val="255000"/>
251: (14)                        </a:schemeClr>
252: (12)                      </a:gs>
253: (10)                    </a:gsLst>
254: (10)                    <a:path path="circle">
255: (12)                      <a:fillToRect l="50000" t="-80000" r="50000" b="180000"/>
256: (10)                    </a:path>
257: (8)                   </a:gradFill>
258: (8)                   <a:gradFill rotWithShape="1">
259: (10)                    <a:gsLst>
260: (12)                      <a:gs pos="0">
261: (14)                        <a:schemeClr val="phClr">
262: (16)                          <a:tint val="80000"/>
263: (16)                          <a:satMod val="300000"/>
264: (14)                        </a:schemeClr>
265: (12)                      </a:gs>
266: (12)                      <a:gs pos="100000">
267: (14)                        <a:schemeClr val="phClr">
268: (16)                          <a:shade val="30000"/>
269: (16)                          <a:satMod val="200000"/>
270: (14)                        </a:schemeClr>
271: (12)                      </a:gs>
272: (10)                    </a:gsLst>
273: (10)                    <a:path path="circle">
274: (12)                      <a:fillToRect l="50000" t="50000" r="50000" b="50000"/>
275: (10)                    </a:path>
276: (8)                   </a:gradFill>
277: (6)                 </a:bgFillStyleLst>
278: (4)               </a:fmtScheme>
279: (2)             </a:themeElements>
280: (2)             <a:objectDefaults/>
281: (2)             <a:extraClrSchemeLst/>
282: (0)           </a:theme>
283: (0)           """
284: (0)           def write_theme():
285: (4)               """Write the theme xml."""
286: (4)               return theme_xml

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

File 184 - scenario.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  String,
4: (4)                  Integer,
5: (4)                  Bool,
6: (4)                  Sequence,
7: (4)                  Convertible,
8: (0)              )
9: (0)              from .cell_range import MultiCellRange
10: (0)             class InputCells(Serialisable):
11: (4)                 tagname = "inputCells"
12: (4)                 r = String()
13: (4)                 deleted = Bool(allow_none=True)
14: (4)                 undone = Bool(allow_none=True)
```

```
15: (4)                 val = String()
16: (4)                 numFmtId = Integer(allow_none=True)
17: (4)                 def __init__(self,
18: (17)                             r=None,
19: (17)                             deleted=False,
20: (17)                             undone=False,
21: (17)                             val=None,
22: (17)                             numFmtId=None,
23: (16)                            ):
24: (8)                     self.r = r
25: (8)                     self.deleted = deleted
26: (8)                     self.undone = undone
27: (8)                     self.val = val
28: (8)                     self.numFmtId = numFmtId
29: (0)             class Scenario(Serialisable):
30: (4)                 tagname = "scenario"
31: (4)                 inputCells = Sequence(expected_type=InputCells)
32: (4)                 name = String()
33: (4)                 locked = Bool(allow_none=True)
34: (4)                 hidden = Bool(allow_none=True)
35: (4)                 user = String(allow_none=True)
36: (4)                 comment = String(allow_none=True)
37: (4)                 __elements__ = ('inputCells',)
38: (4)                 __attrs__ = ('name', 'locked', 'hidden', 'user', 'comment', 'count')
39: (4)                 def __init__(self,
40: (17)                             inputCells=(),
41: (17)                             name=None,
42: (17)                             locked=False,
43: (17)                             hidden=False,
44: (17)                             count=None,
45: (17)                             user=None,
46: (17)                             comment=None,
47: (16)                            ):
48: (8)                     self.inputCells = inputCells
49: (8)                     self.name = name
50: (8)                     self.locked = locked
51: (8)                     self.hidden = hidden
52: (8)                     self.user = user
53: (8)                     self.comment = comment
54: (4)                 @property
55: (4)                 def count(self):
56: (8)                     return len(self.inputCells)
57: (0)             class ScenarioList(Serialisable):
58: (4)                 tagname = "scenarios"
59: (4)                 scenario = Sequence(expected_type=Scenario)
60: (4)                 current = Integer(allow_none=True)
61: (4)                 show = Integer(allow_none=True)
62: (4)                 sqref = Convertible(expected_type=MultiCellRange, allow_none=True)
63: (4)                 __elements__ = ('scenario',)
64: (4)                 def __init__(self,
65: (17)                             scenario=(),
66: (17)                             current=None,
67: (17)                             show=None,
68: (17)                             sqref=None,
69: (16)                            ):
70: (8)                     self.scenario = scenario
71: (8)                     self.current = current
72: (8)                     self.show = show
73: (8)                     self.sqref = sqref
74: (4)                 def append(self, scenario):
75: (8)                     s = self.scenario
76: (8)                     s.append(scenario)
77: (8)                     self.scenario = s
78: (4)                 def __bool__(self):
79: (8)                     return bool(self.scenario)
```

File 185 - __init__.py:

```
1: (0)

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

File 186 - __init__.py:

1: (0)              """Collection of XML resources compatible across different Python versions"""
2: (0)              import os
3: (0)              def lxml_available():
4: (4)                  try:
5: (8)                      from lxml.etree import LXML_VERSION
6: (8)                      LXML = LXML_VERSION >= (3, 3, 1, 0)
7: (8)                      if not LXML:
8: (12)                         import warnings
9: (12)                         warnings.warn("The installed version of lxml is too old to be used
with openpyxl")
10: (12)                        return False  # we have it, but too old
11: (8)                     else:
12: (12)                        return True  # we have it, and recent enough
13: (4)                 except ImportError:
14: (8)                     return False  # we don't even have it
15: (0)             def lxml_env_set():
16: (4)                 return os.environ.get("OPENPYXL_LXML", "True") == "True"
17: (0)             LXML = lxml_available() and lxml_env_set()
18: (0)             def defusedxml_available():
19: (4)                 try:
20: (8)                     import defusedxml # noqa
21: (4)                 except ImportError:
22: (8)                     return False
23: (4)                 else:
24: (8)                     return True
25: (0)             def defusedxml_env_set():
26: (4)                 return os.environ.get("OPENPYXL_DEFUSEDXML", "True") == "True"
27: (0)             DEFUSEDXML = defusedxml_available() and defusedxml_env_set()

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

File 187 - smart_tag.py:

1: (0)              from openpyxl.descriptors.serialisable import Serialisable
2: (0)              from openpyxl.descriptors import (
3: (4)                  Bool,
4: (4)                  Integer,
5: (4)                  String,
6: (4)                  Sequence,
7: (0)              )
8: (0)              class CellSmartTagPr(Serialisable):
9: (4)                  tagname = "cellSmartTagPr"
10: (4)                 key = String()
11: (4)                 val = String()
12: (4)                 def __init__(self,
13: (17)                             key=None,
14: (17)                             val=None,
15: (16)                            ):
16: (8)                     self.key = key
17: (8)                     self.val = val
18: (0)             class CellSmartTag(Serialisable):
19: (4)                 tagname = "cellSmartTag"
20: (4)                 cellSmartTagPr = Sequence(expected_type=CellSmartTagPr)
21: (4)                 type = Integer()
22: (4)                 deleted = Bool(allow_none=True)
23: (4)                 xmlBased = Bool(allow_none=True)
24: (4)                 __elements__ = ('cellSmartTagPr',)
25: (4)                 def __init__(self,
26: (17)                             cellSmartTagPr=(),
27: (17)                             type=None,
28: (17)                             deleted=False,
29: (17)                             xmlBased=False,
(16)                      ):
(8)                   self.cellSmartTagPr = cellSmartTagPr
(8)                   self.type = type
(8)                   self.deleted = deleted
(8)                   self.xmlBased = xmlBased
(0)           class CellSmartTags(Serialisable):
(4)               tagname = "cellSmartTags"
(4)               cellSmartTag = Sequence(expected_type=CellSmartTag)
(4)               r = String()
(4)               __elements__ = ('cellSmartTag',)
(4)               def __init__(self,
(17)                       cellSmartTag=(),
(17)                       r=None,
(16)                      ):
(8)                   self.cellSmartTag = cellSmartTag
(8)                   self.r = r
(0)           class SmartTags(Serialisable):
(4)               tagname = "smartTags"
(4)               cellSmartTags = Sequence(expected_type=CellSmartTags)
(4)               __elements__ = ('cellSmartTags',)
(4)               def __init__(self,
(17)                       cellSmartTags=(),
(16)                      ):
(8)                   self.cellSmartTags = cellSmartTags
```

File 188 - worksheet.py:

```
(0)           """Worksheet is the 2nd-level container in Excel."""
(0)           from itertools import chain
(0)           from operator import itemgetter
(0)           from inspect import isgenerator
(0)           from warnings import warn
(0)           from openpyxl.compat import (
(4)               deprecated,
(0)           )
(0)           from openpyxl.utils import (
(4)               column_index_from_string,
(4)               get_column_letter,
(4)               range_boundaries,
(4)               coordinate_to_tuple,
(0)           )
(0)           from openpyxl.cell import Cell, MergedCell
(0)           from openpyxl.formatting.formatting import ConditionalFormattingList
(0)           from openpyxl.packaging.relationship import RelationshipList
(0)           from openpyxl.workbook.child import _WorkbookChild
(0)           from openpyxl.workbook.defined_name import (
(4)               DefinedNameDict,
(0)           )
(0)           from openpyxl.formula.translate import Translator
(0)           from .datavalidation import DataValidationList
(0)           from .page import (
(4)               PrintPageSetup,
(4)               PageMargins,
(4)               PrintOptions,
(0)           )
(0)           from .dimensions import (
(4)               ColumnDimension,
(4)               RowDimension,
(4)               DimensionHolder,
(4)               SheetFormatProperties,
(0)           )
(0)           from .protection import SheetProtection
(0)           from .filters import AutoFilter
(0)           from .views import (
(4)               Pane,
(4)               Selection,
(4)               SheetViewList,
41: (0)             )
42: (0)             from .cell_range import MultiCellRange, CellRange
43: (0)             from .merge import MergedCellRange
44: (0)             from .properties import WorksheetProperties
45: (0)             from .pagebreak import RowBreak, ColBreak
46: (0)             from .scenario import ScenarioList
47: (0)             from .table import TableList
48: (0)             from .formula import ArrayFormula
49: (0)             from .print_settings import (
50: (4)                 PrintTitles,
51: (4)                 ColRange,
52: (4)                 RowRange,
53: (4)                 PrintArea,
54: (0)             )
55: (0)             class Worksheet(_WorkbookChild):
56: (4)                 """Represents a worksheet.
57: (4)                 Do not create worksheets yourself,
58: (4)                 use :func:`openpyxl.workbook.Workbook.create_sheet` instead
59: (4)                 """
60: (4)                 _rel_type = "worksheet"
61: (4)                 _path = "/xl/worksheets/sheet{0}.xml"
62: (4)                 mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.worksheet+xml"
63: (4)                 BREAK_NONE = 0
64: (4)                 BREAK_ROW = 1
65: (4)                 BREAK_COLUMN = 2
66: (4)                 SHEETSTATE_VISIBLE = 'visible'
67: (4)                 SHEETSTATE_HIDDEN = 'hidden'
68: (4)                 SHEETSTATE_VERYHIDDEN = 'veryHidden'
69: (4)                 PAPERSIZE_LETTER = '1'
70: (4)                 PAPERSIZE_LETTER_SMALL = '2'
71: (4)                 PAPERSIZE_TABLOID = '3'
72: (4)                 PAPERSIZE_LEDGER = '4'
73: (4)                 PAPERSIZE_LEGAL = '5'
74: (4)                 PAPERSIZE_STATEMENT = '6'
75: (4)                 PAPERSIZE_EXECUTIVE = '7'
76: (4)                 PAPERSIZE_A3 = '8'
77: (4)                 PAPERSIZE_A4 = '9'
78: (4)                 PAPERSIZE_A4_SMALL = '10'
79: (4)                 PAPERSIZE_A5 = '11'
80: (4)                 ORIENTATION_PORTRAIT = 'portrait'
81: (4)                 ORIENTATION_LANDSCAPE = 'landscape'
82: (4)                 def __init__(self, parent, title=None):
83: (8)                     _WorkbookChild.__init__(self, parent, title)
84: (8)                     self._setup()
85: (4)                 def _setup(self):
86: (8)                     self.row_dimensions = DimensionHolder(worksheet=self,
87: (46)                                                          default_factory=self._add_row)
88: (8)                     self.column_dimensions = DimensionHolder(worksheet=self,
89: (49)
default_factory=self._add_column)
90: (8)                     self.row_breaks = RowBreak()
91: (8)                     self.col_breaks = ColBreak()
92: (8)                     self._cells = {}
93: (8)                     self._charts = []
94: (8)                     self._images = []
95: (8)                     self._rels = RelationshipList()
96: (8)                     self._drawing = None
97: (8)                     self._comments = []
98: (8)                     self.merged_cells = MultiCellRange()
99: (8)                     self._tables = TableList()
100: (8)                    self._pivots = []
101: (8)                    self.data_validations = DataValidationList()
102: (8)                    self._hyperlinks = []
103: (8)                    self.sheet_state = 'visible'
104: (8)                    self.page_setup = PrintPageSetup(worksheet=self)
105: (8)                    self.print_options = PrintOptions()
106: (8)                    self._print_rows = None
107: (8)                    self._print_cols = None
```

```
108: (8)                        self._print_area = PrintArea()
109: (8)                        self.page_margins = PageMargins()
110: (8)                        self.views = SheetViewList()
111: (8)                        self.protection = SheetProtection()
112: (8)                        self.defined_names = DefinedNameDict()
113: (8)                        self._current_row = 0
114: (8)                        self.auto_filter = AutoFilter()
115: (8)                        self.conditional_formatting = ConditionalFormattingList()
116: (8)                        self.legacy_drawing = None
117: (8)                        self.sheet_properties = WorksheetProperties()
118: (8)                        self.sheet_format = SheetFormatProperties()
119: (8)                        self.scenarios = ScenarioList()
120: (4)                    @property
121: (4)                    def sheet_view(self):
122: (8)                        return self.views.active
123: (4)                    @property
124: (4)                    def selected_cell(self):
125: (8)                        return self.sheet_view.selection[0].sqref
126: (4)                    @property
127: (4)                    def active_cell(self):
128: (8)                        return self.sheet_view.selection[0].activeCell
129: (4)                    @property
130: (4)                    def array_formulae(self):
131: (8)                        """Returns a dictionary of cells with array formulae and the cells in
array"""
132: (8)                        result = {}
133: (8)                        for c in self._cells.values():
134: (12)                           if c.data_type == "f":
135: (16)                               if isinstance(c.value, ArrayFormula):
136: (20)                                   result[c.coordinate] = c.value.ref
137: (8)                        return result
138: (4)                    @property
139: (4)                    def show_gridlines(self):
140: (8)                        return self.sheet_view.showGridLines
141: (4)                    @property
142: (4)                    def freeze_panes(self):
143: (8)                        if self.sheet_view.pane is not None:
144: (12)                           return self.sheet_view.pane.topLeftCell
145: (4)                    @freeze_panes.setter
146: (4)                    def freeze_panes(self, topLeftCell=None):
147: (8)                        if isinstance(topLeftCell, Cell):
148: (12)                           topLeftCell = topLeftCell.coordinate
149: (8)                        if topLeftCell == 'A1':
150: (12)                           topLeftCell = None
151: (8)                        if not topLeftCell:
152: (12)                           self.sheet_view.pane = None
153: (12)                           return
154: (8)                        row, column = coordinate_to_tuple(topLeftCell)
155: (8)                        view = self.sheet_view
156: (8)                        view.pane = Pane(topLeftCell=topLeftCell,
157: (24)                                           activePane="topRight",
158: (24)                                           state="frozen")
159: (8)                        view.selection[0].pane = "topRight"
160: (8)                        if column > 1:
161: (12)                           view.pane.xSplit = column - 1
162: (8)                        if row > 1:
163: (12)                           view.pane.ySplit = row - 1
164: (12)                           view.pane.activePane = 'bottomLeft'
165: (12)                           view.selection[0].pane = "bottomLeft"
166: (12)                           if column > 1:
167: (16)                               view.selection[0].pane = "bottomRight"
168: (16)                               view.pane.activePane = 'bottomRight'
169: (8)                        if row > 1 and column > 1:
170: (12)                           sel = list(view.selection)
171: (12)                           sel.insert(0, Selection(pane="topRight", activeCell=None,
sqref=None))
172: (12)                           sel.insert(1, Selection(pane="bottomLeft", activeCell=None,
sqref=None))
173: (12)                           view.selection = sel
```

```
(4)                 def cell(self, row, column, value=None):
(8)                     """
(8)                     Returns a cell object based on the given coordinates.
(8)                     Usage: cell(row=15, column=1, value=5)
(8)                     Calling `cell` creates cells in memory when they
(8)                     are first accessed.
(8)                     :param row: row index of the cell (e.g. 4)
(8)                     :type row: int
(8)                     :param column: column index of the cell (e.g. 3)
(8)                     :type column: int
(8)                     :param value: value of the cell (e.g. 5)
(8)                     :type value: numeric or time or string or bool or none
(8)                     :rtype: openpyxl.cell.cell.Cell
(8)                     """
(8)                     if row < 1 or column < 1:
(12)                        raise ValueError("Row or column values must be at least 1")
(8)                     cell = self._get_cell(row, column)
(8)                     if value is not None:
(12)                        cell.value = value
(8)                     return cell
(4)                 def _get_cell(self, row, column):
(8)                     """
(8)                     Internal method for getting a cell from a worksheet.
(8)                     Will create a new cell if one doesn't already exist.
(8)                     """
(8)                     if not 0 < row < 1048577:
(12)                        raise ValueError(f"Row numbers must be between 1 and 1048576. Row number supplied was {row}")
(8)                     coordinate = (row, column)
(8)                     if not coordinate in self._cells:
(12)                        cell = Cell(self, row=row, column=column)
(12)                        self._add_cell(cell)
(8)                     return self._cells[coordinate]
(4)                 def _add_cell(self, cell):
(8)                     """
(8)                     Internal method for adding cell objects.
(8)                     """
(8)                     column = cell.col_idx
(8)                     row = cell.row
(8)                     self._current_row = max(row, self._current_row)
(8)                     self._cells[(row, column)] = cell
(4)                 def __getitem__(self, key):
(8)                     """Convenience access by Excel style coordinates
(8)                     The key can be a single cell coordinate 'A1', a range of cells 'A1:D25',
(8)                     individual rows or columns 'A', 4 or ranges of rows or columns 'A:D',
(8)                     4:10.
(8)                     Single cells will always be created if they do not exist.
(8)                     Returns either a single cell or a tuple of rows or columns.
(8)                     """
(8)                     if isinstance(key, slice):
(12)                        if not all([key.start, key.stop]):
(16)                            raise IndexError("{0} is not a valid coordinate or range".format(key))
(12)                        key = "{0}:{1}".format(key.start, key.stop)
(8)                     if isinstance(key, int):
(12)                        key = str(key
(22)                                  )
(8)                     min_col, min_row, max_col, max_row = range_boundaries(key)
(8)                     if not any([min_col, min_row, max_col, max_row]):
(12)                        raise IndexError("{0} is not a valid coordinate or range".format(key))
(8)                     if min_row is None:
(12)                        cols = tuple(self.iter_cols(min_col, max_col))
(12)                        if min_col == max_col:
(16)                            cols = cols[0]
(12)                        return cols
(8)                     if min_col is None:
(12)                        rows = tuple(self.iter_rows(min_col=min_col, min_row=min_row,
(40)                                            max_col=self.max_column, max_row=max_row))
(12)                if min_row == max_row:
(16)                    rows = rows[0]
(12)                return rows
(8)             if ":" not in key:
(12)                return self._get_cell(min_row, min_col)
(8)             return tuple(self.iter_rows(min_row=min_row, min_col=min_col,
(36)                                        max_row=max_row, max_col=max_col))
(4)         def __setitem__(self, key, value):
(8)             self[key].value = value
(4)         def __iter__(self):
(8)             return self.iter_rows()
(4)         def __delitem__(self, key):
(8)             row, column = coordinate_to_tuple(key)
(8)             if (row, column) in self._cells:
(12)                del self._cells[(row, column)]
(4)         @property
(4)         def min_row(self):
(8)             """The minimum row index containing data (1-based)
(8)             :type: int
(8)             """
(8)             min_row = 1
(8)             if self._cells:
(12)                min_row = min(self._cells)[0]
(8)             return min_row
(4)         @property
(4)         def max_row(self):
(8)             """The maximum row index containing data (1-based)
(8)             :type: int
(8)             """
(8)             max_row = 1
(8)             if self._cells:
(12)                max_row = max(self._cells)[0]
(8)             return max_row
(4)         @property
(4)         def min_column(self):
(8)             """The minimum column index containing data (1-based)
(8)             :type: int
(8)             """
(8)             min_col = 1
(8)             if self._cells:
(12)                min_col = min(c[1] for c in self._cells)
(8)             return min_col
(4)         @property
(4)         def max_column(self):
(8)             """The maximum column index containing data (1-based)
(8)             :type: int
(8)             """
(8)             max_col = 1
(8)             if self._cells:
(12)                max_col = max(c[1] for c in self._cells)
(8)             return max_col
(4)         def calculate_dimension(self):
(8)             """Return the minimum bounding range for all cells containing data (ex. 'A1:M24')
(8)             :rtype: string
(8)             """
(8)             if self._cells:
(12)                rows = set()
(12)                cols = set()
(12)                for row, col in self._cells:
(16)                    rows.add(row)
(16)                    cols.add(col)
(12)                max_row = max(rows)
(12)                max_col = max(cols)
(12)                min_col = min(cols)
(12)                min_row = min(rows)
(8)             else:
```

```
306: (12)                           return "A1:A1"
307: (8)                        return f"{get_column_letter(min_col)}{min_row}:
{get_column_letter(max_col)}{max_row}"
308: (4)                    @property
309: (4)                    def dimensions(self):
310: (8)                        """Returns the result of :func:`calculate_dimension`"""
311: (8)                        return self.calculate_dimension()
312: (4)                    def iter_rows(self, min_row=None, max_row=None, min_col=None,
max_col=None, values_only=False):
313: (8)                        """
314: (8)                        Produces cells from the worksheet, by row. Specify the iteration range
315: (8)                        using indices of rows and columns.
316: (8)                        If no indices are specified the range starts at A1.
317: (8)                        If no cells are in the worksheet an empty tuple will be returned.
318: (8)                        :param min_col: smallest column index (1-based index)
319: (8)                        :type min_col: int
320: (8)                        :param min_row: smallest row index (1-based index)
321: (8)                        :type min_row: int
322: (8)                        :param max_col: largest column index (1-based index)
323: (8)                        :type max_col: int
324: (8)                        :param max_row: largest row index (1-based index)
325: (8)                        :type max_row: int
326: (8)                        :param values_only: whether only cell values should be returned
327: (8)                        :type values_only: bool
328: (8)                        :rtype: generator
329: (8)                        """
330: (8)                        if self._current_row == 0 and not any([min_col, min_row, max_col,
max_row ]):
331: (12)                           return iter(())
332: (8)                        min_col = min_col or 1
333: (8)                        min_row = min_row or 1
334: (8)                        max_col = max_col or self.max_column
335: (8)                        max_row = max_row or self.max_row
336: (8)                        return self._cells_by_row(min_col, min_row, max_col, max_row,
values_only)
337: (4)                    def _cells_by_row(self, min_col, min_row, max_col, max_row,
values_only=False):
338: (8)                        for row in range(min_row, max_row + 1):
339: (12)                           cells = (self.cell(row=row, column=column) for column in
range(min_col, max_col + 1))
340: (12)                           if values_only:
341: (16)                               yield tuple(cell.value for cell in cells)
342: (12)                           else:
343: (16)                               yield tuple(cells)
344: (4)                    @property
345: (4)                    def rows(self):
346: (8)                        """Produces all cells in the worksheet, by row (see :func:`iter_rows`)
347: (8)                        :type: generator
348: (8)                        """
349: (8)                        return self.iter_rows()
350: (4)                    @property
351: (4)                    def values(self):
352: (8)                        """Produces all cell values in the worksheet, by row
353: (8)                        :type: generator
354: (8)                        """
355: (8)                        for row in self.iter_rows(values_only=True):
356: (12)                           yield row
357: (4)                    def iter_cols(self, min_col=None, max_col=None, min_row=None,
max_row=None, values_only=False):
358: (8)                        """
359: (8)                        Produces cells from the worksheet, by column. Specify the iteration
range
360: (8)                        using indices of rows and columns.
361: (8)                        If no indices are specified the range starts at A1.
362: (8)                        If no cells are in the worksheet an empty tuple will be returned.
363: (8)                        :param min_col: smallest column index (1-based index)
364: (8)                        :type min_col: int
365: (8)                        :param min_row: smallest row index (1-based index)
366: (8)                        :type min_row: int
```

```
(8)                 :param max_col: largest column index (1-based index)
(8)                 :type max_col: int
(8)                 :param max_row: largest row index (1-based index)
(8)                 :type max_row: int
(8)                 :param values_only: whether only cell values should be returned
(8)                 :type values_only: bool
(8)                 :rtype: generator
(8)                 """
(8)                 if self._current_row == 0 and not any([min_col, min_row, max_col,
max_row]):
(12)                    return iter(())
(8)                 min_col = min_col or 1
(8)                 min_row = min_row or 1
(8)                 max_col = max_col or self.max_column
(8)                 max_row = max_row or self.max_row
(8)                 return self._cells_by_col(min_col, min_row, max_col, max_row,
values_only)
(4)             def _cells_by_col(self, min_col, min_row, max_col, max_row,
values_only=False):
(8)                 """
(8)                 Get cells by column
(8)                 """
(8)                 for column in range(min_col, max_col+1):
(12)                    cells = (self.cell(row=row, column=column)
(24)                                for row in range(min_row, max_row+1))
(12)                    if values_only:
(16)                        yield tuple(cell.value for cell in cells)
(12)                    else:
(16)                        yield tuple(cells)
(4)             @property
(4)             def columns(self):
(8)                 """Produces all cells in the worksheet, by column  (see
:func:`iter_cols`)"""
(8)                 return self.iter_cols()
(4)             @property
(4)             def column_groups(self):
(8)                 """
(8)                 Return a list of column ranges where more than one column
(8)                 """
(8)                 return [cd.range for cd in self.column_dimensions.values() if cd.min
and cd.max > cd.min]
(4)             def set_printer_settings(self, paper_size, orientation):
(8)                 """Set printer settings """
(8)                 self.page_setup.paperSize = paper_size
(8)                 self.page_setup.orientation = orientation
(4)             def add_data_validation(self, data_validation):
(8)                 """ Add a data-validation object to the sheet.  The data-validation
(12)                    object defines the type of data-validation to be applied and the
(12)                    cell or range of cells it should apply to.
(8)                 """
(8)                 self.data_validations.append(data_validation)
(4)             def add_chart(self, chart, anchor=None):
(8)                 """
(8)                 Add a chart to the sheet
(8)                 Optionally provide a cell for the top-left anchor
(8)                 """
(8)                 if anchor is not None:
(12)                    chart.anchor = anchor
(8)                 self._charts.append(chart)
(4)             def add_image(self, img, anchor=None):
(8)                 """
(8)                 Add an image to the sheet.
(8)                 Optionally provide a cell for the top-left anchor
(8)                 """
(8)                 if anchor is not None:
(12)                    img.anchor = anchor
(8)                 self._images.append(img)
(4)             def add_table(self, table):
(8)                 """
```

```
431: (8)                    Check for duplicate name in definedNames and other worksheet tables
432: (8)                    before adding table.
433: (8)                    """
434: (8)                    if self.parent._duplicate_name(table.name):
435: (12)                       raise ValueError("Table with name {0} already
exists".format(table.name))
436: (8)                    if not hasattr(self, "_get_cell"):
437: (12)                       warn("In write-only mode you must add table columns manually")
438: (8)                    self._tables.add(table)
439: (4)                @property
440: (4)                def tables(self):
441: (8)                    return self._tables
442: (4)                def add_pivot(self, pivot):
443: (8)                    self._pivots.append(pivot)
444: (4)                def merge_cells(self, range_string=None, start_row=None,
start_column=None, end_row=None, end_column=None):
445: (8)                    """ Set merge on a cell range.  Range is a cell range (e.g. A1:E1) """
446: (8)                    if range_string is None:
447: (12)                       cr = CellRange(range_string=range_string, min_col=start_column,
min_row=start_row,
448: (22)                                 max_col=end_column, max_row=end_row)
449: (12)                       range_string = cr.coord
450: (8)                    mcr = MergedCellRange(self, range_string)
451: (8)                    self.merged_cells.add(mcr)
452: (8)                    self._clean_merge_range(mcr)
453: (4)                def _clean_merge_range(self, mcr):
454: (8)                    """
455: (8)                    Remove all but the top left-cell from a range of merged cells
456: (8)                    and recreate the lost border information.
457: (8)                    Borders are then applied
458: (8)                    """
459: (8)                    cells = mcr.cells
460: (8)                    next(cells) # skip first cell
461: (8)                    for row, col in cells:
462: (12)                       self._cells[row, col] = MergedCell(self, row, col)
463: (8)                    mcr.format()
464: (4)                @property
465: (4)                @deprecated("Use ws.merged_cells.ranges")
466: (4)                def merged_cell_ranges(self):
467: (8)                    """Return a copy of cell ranges"""
468: (8)                    return self.merged_cells.ranges[:]
469: (4)                def unmerge_cells(self, range_string=None, start_row=None,
start_column=None, end_row=None, end_column=None):
470: (8)                    """ Remove merge on a cell range.  Range is a cell range (e.g. A1:E1)
"""
471: (8)                    cr = CellRange(range_string=range_string, min_col=start_column,
min_row=start_row,
472: (22)                                 max_col=end_column, max_row=end_row)
473: (8)                    if cr.coord not in self.merged_cells:
474: (12)                       raise ValueError("Cell range {0} is not merged".format(cr.coord))
475: (8)                    self.merged_cells.remove(cr)
476: (8)                    cells = cr.cells
477: (8)                    next(cells) # skip first cell
478: (8)                    for row, col in cells:
479: (12)                       del self._cells[(row, col)]
480: (4)                def append(self, iterable):
481: (8)                    """Appends a group of values at the bottom of the current sheet.
482: (8)                    * If it's a list: all values are added in order, starting from the
first column
483: (8)                    * If it's a dict: values are assigned to the columns indicated by the
keys (numbers or letters)
484: (8)                    :param iterable: list, range or generator, or dict containing values
to append
485: (8)                    :type iterable: list|tuple|range|generator or dict
486: (8)                    Usage:
487: (8)                    * append(['This is A1', 'This is B1', 'This is C1'])
488: (8)                    * **or** append({'A' : 'This is A1', 'C' : 'This is C1'})
489: (8)                    * **or** append({1 : 'This is A1', 3 : 'This is C1'})
490: (8)                    :raise: TypeError when iterable is neither a list/tuple nor a dict
```

```
(8)                 """
(8)                 row_idx = self._current_row + 1
(8)                 if (isinstance(iterable, (list, tuple, range))
(12)                    or isgenerator(iterable)):
(12)                    for col_idx, content in enumerate(iterable, 1):
(16)                        if isinstance(content, Cell):
(20)                            cell = content
(20)                            if cell.parent and cell.parent != self:
(24)                                raise ValueError("Cells cannot be copied from other worksheets")
(20)                            cell.parent = self
(20)                            cell.column = col_idx
(20)                            cell.row = row_idx
(16)                        else:
(20)                            cell = Cell(self, row=row_idx, column=col_idx, value=content)
(16)                        self._cells[(row_idx, col_idx)] = cell
(8)                 elif isinstance(iterable, dict):
(12)                    for col_idx, content in iterable.items():
(16)                        if isinstance(col_idx, str):
(20)                            col_idx = column_index_from_string(col_idx)
(16)                        cell = Cell(self, row=row_idx, column=col_idx, value=content)
(16)                        self._cells[(row_idx, col_idx)] = cell
(8)                 else:
(12)                    self._invalid_row(iterable)
(8)                 self._current_row = row_idx
(4)             def _move_cells(self, min_row=None, min_col=None, offset=0, row_or_col="row"):
(8)                 """
(8)                 Move either rows or columns around by the offset
(8)                 """
(8)                 reverse = offset > 0 # start at the end if inserting
(8)                 row_offset = 0
(8)                 col_offset = 0
(8)                 if row_or_col == 'row':
(12)                    cells = self.iter_rows(min_row=min_row)
(12)                    row_offset = offset
(12)                    key = 0
(8)                 else:
(12)                    cells = self.iter_cols(min_col=min_col)
(12)                    col_offset = offset
(12)                    key = 1
(8)                 cells = list(cells)
(8)                 for row, column in sorted(self._cells, key=itemgetter(key), reverse=reverse):
(12)                    if min_row and row < min_row:
(16)                        continue
(12)                    elif min_col and column < min_col:
(16)                        continue
(12)                    self._move_cell(row, column, row_offset, col_offset)
(4)             def insert_rows(self, idx, amount=1):
(8)                 """
(8)                 Insert row or rows before row==idx
(8)                 """
(8)                 self._move_cells(min_row=idx, offset=amount, row_or_col="row")
(8)                 self._current_row = self.max_row
(4)             def insert_cols(self, idx, amount=1):
(8)                 """
(8)                 Insert column or columns before col==idx
(8)                 """
(8)                 self._move_cells(min_col=idx, offset=amount, row_or_col="column")
(4)             def delete_rows(self, idx, amount=1):
(8)                 """
(8)                 Delete row or rows from row==idx
(8)                 """
(8)                 remainder = _gutter(idx, amount, self.max_row)
(8)                 self._move_cells(min_row=idx+amount, offset=-amount, row_or_col="row")
(8)                 min_col = self.min_column
(8)                 max_col = self.max_column + 1
```

```
556: (8)                    for row in remainder:
557: (12)                       for col in range(min_col, max_col):
558: (16)                           if (row, col) in self._cells:
559: (20)                               del self._cells[row, col]
560: (8)                    self._current_row = self.max_row
561: (8)                    if not self._cells:
562: (12)                       self._current_row = 0
563: (4)                def delete_cols(self, idx, amount=1):
564: (8)                    """
565: (8)                    Delete column or columns from col==idx
566: (8)                    """
567: (8)                    remainder = _gutter(idx, amount, self.max_column)
568: (8)                    self._move_cells(min_col=idx+amount, offset=-amount,
row_or_col="column")
569: (8)                    min_row = self.min_row
570: (8)                    max_row = self.max_row + 1
571: (8)                    for col in remainder:
572: (12)                       for row in range(min_row, max_row):
573: (16)                           if (row, col) in self._cells:
574: (20)                               del self._cells[row, col]
575: (4)                def move_range(self, cell_range, rows=0, cols=0, translate=False):
576: (8)                    """
577: (8)                    Move a cell range by the number of rows and/or columns:
578: (8)                    down if rows > 0 and up if rows < 0
579: (8)                    right if cols > 0 and left if cols < 0
580: (8)                    Existing cells will be overwritten.
581: (8)                    Formulae and references will not be updated.
582: (8)                    """
583: (8)                    if isinstance(cell_range, str):
584: (12)                       cell_range = CellRange(cell_range)
585: (8)                    if not isinstance(cell_range, CellRange):
586: (12)                       raise ValueError("Only CellRange objects can be moved")
587: (8)                    if not rows and not cols:
588: (12)                       return
589: (8)                    down = rows > 0
590: (8)                    right = cols > 0
591: (8)                    if rows:
592: (12)                       cells = sorted(cell_range.rows, reverse=down)
593: (8)                    else:
594: (12)                       cells = sorted(cell_range.cols, reverse=right)
595: (8)                    for row, col in chain.from_iterable(cells):
596: (12)                       self._move_cell(row, col, rows, cols, translate)
597: (8)                    cell_range.shift(row_shift=rows, col_shift=cols)
598: (4)                def _move_cell(self, row, column, row_offset, col_offset,
translate=False):
599: (8)                    """
600: (8)                    Move a cell from one place to another.
601: (8)                    Delete at old index
602: (8)                    Rebase coordinate
603: (8)                    """
604: (8)                    cell = self._get_cell(row, column)
605: (8)                    new_row = cell.row + row_offset
606: (8)                    new_col = cell.column + col_offset
607: (8)                    self._cells[new_row, new_col] = cell
608: (8)                    del self._cells[(cell.row, cell.column)]
609: (8)                    cell.row = new_row
610: (8)                    cell.column = new_col
611: (8)                    if translate and cell.data_type == "f":
612: (12)                       t = Translator(cell.value, cell.coordinate)
613: (12)                       cell.value = t.translate_formula(row_delta=row_offset,
col_delta=col_offset)
614: (4)                def _invalid_row(self, iterable):
615: (8)                    raise TypeError('Value must be a list, tuple, range or generator, or a
dict. Supplied value is {0}'.format(
616: (12)                       type(iterable))
617: (24)                               )
618: (4)                def _add_column(self):
619: (8)                    """Dimension factory for column information"""
620: (8)                    return ColumnDimension(self)
```

```
(4)                 def _add_row(self):
(8)                     """Dimension factory for row information"""
(8)                     return RowDimension(self)
(4)                 @property
(4)                 def print_title_rows(self):
(8)                     """Rows to be printed at the top of every page (ex: '1:3')"""
(8)                     if self._print_rows:
(12)                        return str(self._print_rows)
(4)                 @print_title_rows.setter
(4)                 def print_title_rows(self, rows):
(8)                     """
(8)                     Set rows to be printed on the top of every page
(8)                     format `1:3`
(8)                     """
(8)                     if rows is not None:
(12)                        self._print_rows = RowRange(rows)
(4)                 @property
(4)                 def print_title_cols(self):
(8)                     """Columns to be printed at the left side of every page (ex: 'A:C')"""
(8)                     if self._print_cols:
(12)                        return str(self._print_cols)
(4)                 @print_title_cols.setter
(4)                 def print_title_cols(self, cols):
(8)                     """
(8)                     Set cols to be printed on the left of every page
(8)                     format ``A:C`
(8)                     """
(8)                     if cols is not None:
(12)                        self._print_cols = ColRange(cols)
(4)                 @property
(4)                 def print_titles(self):
(8)                     titles = PrintTitles(cols=self._print_cols, rows=self._print_rows, title=self.title)
(8)                     return str(titles)
(4)                 @property
(4)                 def print_area(self):
(8)                     """
(8)                     The print area for the worksheet, or None if not set. To set, supply a range
(8)                     like 'A1:D4' or a list of ranges.
(8)                     """
(8)                     self._print_area.title = self.title
(8)                     return str(self._print_area)
(4)                 @print_area.setter
(4)                 def print_area(self, value):
(8)                     """
(8)                     Range of cells in the form A1:D4 or list of ranges. Print area can be cleared
(8)                     by passing `None` or an empty list
(8)                     """
(8)                     if not value:
(12)                        self._print_area = PrintArea()
(8)                     elif isinstance(value, str):
(12)                        self._print_area = PrintArea.from_string(value)
(8)                     elif hasattr(value, "__iter__"):
(12)                        self._print_area = PrintArea.from_string(",".join(value))
(0)             def _gutter(idx, offset, max_val):
(4)                 """
(4)                 When deleting rows and columns are deleted we rely on overwriting.
(4)                 This may not be the case for a large offset on small set of cells:
(4)                 range(cells_to_delete) > range(cell_to_be_moved)
(4)                 """
(4)                 gutter = range(max(max_val+1-offset, idx), min(idx+offset, max_val)+1)
(4)                 return gutter
```

File 189 - constants.py:

```
"""Constants for fixed paths in a file and xml namespace urls."""
MIN_ROW = 0
MIN_COLUMN = 0
MAX_COLUMN = 16384
MAX_ROW = 1048576
PACKAGE_PROPS = 'docProps'
PACKAGE_XL = 'xl'
PACKAGE_RELS = '_rels'
PACKAGE_THEME = PACKAGE_XL + '/' + 'theme'
PACKAGE_WORKSHEETS = PACKAGE_XL + '/' + 'worksheets'
PACKAGE_CHARTSHEETS = PACKAGE_XL + '/' + 'chartsheets'
PACKAGE_DRAWINGS = PACKAGE_XL + '/' + 'drawings'
PACKAGE_CHARTS = PACKAGE_XL + '/' + 'charts'
PACKAGE_IMAGES = PACKAGE_XL + '/' + 'media'
PACKAGE_WORKSHEET_RELS = PACKAGE_WORKSHEETS + '/' + '_rels'
PACKAGE_CHARTSHEETS_RELS = PACKAGE_CHARTSHEETS + '/' + '_rels'
PACKAGE_PIVOT_TABLE = PACKAGE_XL + '/' + 'pivotTables'
PACKAGE_PIVOT_CACHE = PACKAGE_XL + '/' + 'pivotCache'
ARC_CONTENT_TYPES = '[Content_Types].xml'
ARC_ROOT_RELS = PACKAGE_RELS + '/.rels'
ARC_WORKBOOK_RELS = PACKAGE_XL + '/' + PACKAGE_RELS + '/workbook.xml.rels'
ARC_CORE = PACKAGE_PROPS + '/core.xml'
ARC_APP = PACKAGE_PROPS + '/app.xml'
ARC_CUSTOM = PACKAGE_PROPS + '/custom.xml'
ARC_WORKBOOK = PACKAGE_XL + '/workbook.xml'
ARC_STYLE = PACKAGE_XL + '/styles.xml'
ARC_THEME = PACKAGE_THEME + '/theme1.xml'
ARC_SHARED_STRINGS = PACKAGE_XL + '/sharedStrings.xml'
ARC_CUSTOM_UI = 'customUI/customUI.xml'
XML_NS = "http://www.w3.org/XML/1998/namespace"
DCORE_NS = 'http://purl.org/dc/elements/1.1/'
DCTERMS_NS = 'http://purl.org/dc/terms/'
DCTERMS_PREFIX = 'dcterms'
DOC_NS = "http://schemas.openxmlformats.org/officeDocument/2006/"
REL_NS = DOC_NS + "relationships"
COMMENTS_NS = REL_NS + "/comments"
IMAGE_NS = REL_NS + "/image"
VML_NS =  REL_NS + "/vmlDrawing"
VTYPES_NS = DOC_NS + 'docPropsVTypes'
XPROPS_NS = DOC_NS + 'extended-properties'
CUSTPROPS_NS = DOC_NS + 'custom-properties'
EXTERNAL_LINK_NS = REL_NS + "/externalLink"
CPROPS_FMTID = "{D5CDD505-2E9C-101B-9397-08002B2CF9AE}"
PKG_NS = "http://schemas.openxmlformats.org/package/2006/"
PKG_REL_NS = PKG_NS + "relationships"
COREPROPS_NS = PKG_NS + 'metadata/core-properties'
CONTYPES_NS = PKG_NS + 'content-types'
XSI_NS = 'http://www.w3.org/2001/XMLSchema-instance'
XML_NS = 'http://www.w3.org/XML/1998/namespace'
SHEET_MAIN_NS = 'http://schemas.openxmlformats.org/spreadsheetml/2006/main'
CHART_NS = "http://schemas.openxmlformats.org/drawingml/2006/chart"
DRAWING_NS = "http://schemas.openxmlformats.org/drawingml/2006/main"
SHEET_DRAWING_NS = "http://schemas.openxmlformats.org/drawingml/2006/spreadsheetDrawing"
CHART_DRAWING_NS = "http://schemas.openxmlformats.org/drawingml/2006/chartDrawing"
CUSTOMUI_NS = 'http://schemas.microsoft.com/office/2006/relationships/ui/extensibility'
NAMESPACES = {
    'cp': COREPROPS_NS,
    'dc': DCORE_NS,
    DCTERMS_PREFIX: DCTERMS_NS,
    'dcmitype': 'http://purl.org/dc/dcmitype/',
    'xsi': XSI_NS,
    'vt': VTYPES_NS,
    'xml': XML_NS,
    'main': SHEET_MAIN_NS,
    'cust': CUSTPROPS_NS,
}
```

```
67: (0)             WORKBOOK_MACRO = "application/vnd.ms-excel.%s.macroEnabled.main+xml"
68: (0)             WORKBOOK = "application/vnd.openxmlformats-
officedocument.spreadsheetml.%s.main+xml"
69: (0)             SPREADSHEET = "application/vnd.openxmlformats-
officedocument.spreadsheetml.%s+xml"
70: (0)             SHARED_STRINGS = SPREADSHEET % "sharedStrings"
71: (0)             EXTERNAL_LINK = SPREADSHEET % "externalLink"
72: (0)             WORKSHEET_TYPE = SPREADSHEET % "worksheet"
73: (0)             COMMENTS_TYPE = SPREADSHEET % "comments"
74: (0)             STYLES_TYPE = SPREADSHEET % "styles"
75: (0)             CHARTSHEET_TYPE = SPREADSHEET % "chartsheet"
76: (0)             DRAWING_TYPE = "application/vnd.openxmlformats-officedocument.drawing+xml"
77: (0)             CHART_TYPE = "application/vnd.openxmlformats-
officedocument.drawingml.chart+xml"
78: (0)             CHARTSHAPE_TYPE = "application/vnd.openxmlformats-
officedocument.drawingml.chartshapes+xml"
79: (0)             THEME_TYPE = "application/vnd.openxmlformats-officedocument.theme+xml"
80: (0)             CPROPS_TYPE = "application/vnd.openxmlformats-officedocument.custom-
properties+xml"
81: (0)             XLTM = WORKBOOK_MACRO % 'template'
82: (0)             XLSM = WORKBOOK_MACRO % 'sheet'
83: (0)             XLTX = WORKBOOK % 'template'
84: (0)             XLSX = WORKBOOK % 'sheet'
85: (0)             EXT_TYPES = {
86: (4)                 '{78C0D931-6437-407D-A8EE-F0AAD7539E65}': 'Conditional Formatting',
87: (4)                 '{CCE6A557-97BC-4B89-ADB6-D9C93CAAB3DF}': 'Data Validation',
88: (4)                 '{05C60535-1F16-4FD2-B633-F4F36F0B64E0}': 'Sparkline Group',
89: (4)                 '{A8765BA9-456A-4DAB-B4F3-ACF838C121DE}': 'Slicer List',
90: (4)                 '{FC87AEE6-9EDD-4A0A-B7FB-166176984837}': 'Protected Range',
91: (4)                 '{01252117-D84E-4E92-8308-4BE1C098FCBB}': 'Ignored Error',
92: (4)                 '{F7C9EE02-42E1-4005-9D12-6889AFFD525C}': 'Web Extension',
93: (4)                 '{3A4CF648-6AED-40f4-86FF-DC5316D8AED3}': 'Slicer List',
94: (4)                 '{7E03D99C-DC04-49d9-9315-930204A7B6E9}': 'Timeline Ref',
95: (0)             }
96: (0)             CTRL = "application/vnd.ms-excel.controlproperties+xml"
97: (0)             ACTIVEX = "application/vnd.ms-office.activeX+xml"
98: (0)             VBA = "application/vnd.ms-office.vbaProject"

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

File 190 - functions.py:

1: (0)              """
2: (0)              XML compatibility functions
3: (0)              """
4: (0)              import re
5: (0)              from functools import partial
6: (0)              from openpyxl import DEFUSEDXML, LXML
7: (0)              if LXML is True:
8: (4)                  from lxml.etree import (
9: (4)                  Element,
10: (4)                 SubElement,
11: (4)                 register_namespace,
12: (4)                 QName,
13: (4)                 xmlfile,
14: (4)                 XMLParser,
15: (4)                 )
16: (4)                 from lxml.etree import fromstring, tostring
17: (4)                 safe_parser = XMLParser(resolve_entities=False)
18: (4)                 fromstring = partial(fromstring, parser=safe_parser)
19: (0)             else:
20: (4)                 from xml.etree.ElementTree import (
21: (4)                 Element,
22: (4)                 SubElement,
23: (4)                 fromstring,
24: (4)                 tostring,
25: (4)                 QName,
26: (4)                 register_namespace
27: (4)                 )
```

```
28: (4)                 from et_xmlfile import xmlfile
29: (4)                 if DEFUSEDXML is True:
30: (8)                     from defusedxml.ElementTree import fromstring
31: (0)             from xml.etree.ElementTree import iterparse
32: (0)             if DEFUSEDXML is True:
33: (4)                 from defusedxml.ElementTree import iterparse
34: (0)             from openpyxl.xml.constants import (
35: (4)                 CHART_NS,
36: (4)                 DRAWING_NS,
37: (4)                 SHEET_DRAWING_NS,
38: (4)                 CHART_DRAWING_NS,
39: (4)                 SHEET_MAIN_NS,
40: (4)                 REL_NS,
41: (4)                 VTYPES_NS,
42: (4)                 COREPROPS_NS,
43: (4)                 CUSTPROPS_NS,
44: (4)                 DCTERMS_NS,
45: (4)                 DCTERMS_PREFIX,
46: (4)                 XML_NS
47: (0)             )
48: (0)             register_namespace(DCTERMS_PREFIX, DCTERMS_NS)
49: (0)             register_namespace('dcmitype', 'http://purl.org/dc/dcmitype/')
50: (0)             register_namespace('cp', COREPROPS_NS)
51: (0)             register_namespace('c', CHART_NS)
52: (0)             register_namespace('a', DRAWING_NS)
53: (0)             register_namespace('s', SHEET_MAIN_NS)
54: (0)             register_namespace('r', REL_NS)
55: (0)             register_namespace('vt', VTYPES_NS)
56: (0)             register_namespace('xdr', SHEET_DRAWING_NS)
57: (0)             register_namespace('cdr', CHART_DRAWING_NS)
58: (0)             register_namespace('xml', XML_NS)
59: (0)             register_namespace('cust', CUSTPROPS_NS)
60: (0)             tostring = partial(tostring, encoding="utf-8")
61: (0)             NS_REGEX = re.compile("({(?P<namespace>.*)})?(?P<localname>.*)")
62: (0)             def localname(node):
63: (4)                 if callable(node.tag):
64: (8)                     return "comment"
65: (4)                 m = NS_REGEX.match(node.tag)
66: (4)                 return m.group('localname')
67: (0)             def whitespace(node):
68: (4)                 stripped = node.text.strip()
69: (4)                 if stripped and node.text != stripped:
70: (8)                     node.set("{%s}space" % XML_NS, "preserve")

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

File 191 - 
SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRYCOMBINER_aligner_20_characters_for_pythons_codes.p
y:

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 =
'SANJOYNATHQHENOMENOLOGYGEOMEETRIFYINGTRIGONOMETRY_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}")

----------------------------------------
```