BlendDataLibraries(bpy_struct)

base class — bpy_struct

class bpy.types.**BlendDataLibraries(bpy_struct)**

Collection of libraries

tag(value)

tag

PARAMETERS:

value (*boolean*) – Value

remove(library, *, do_unlink=True, do_id_user=True, do_ui_user=True)

Remove a library from the current blendfile

PARAMETERS:

- **library** (Library , (never None)) – Library to remove
- **do_unlink** (*boolean, (optional)*) – Unlink all usages of this library before deleting it
- **do_id_user** (*boolean, (optional)*) – Decrement user counter of all datablocks used by this library
- **do_ui_user** (*boolean, (optional)*) – Make sure interface does not reference this library

classmethod bl_rna_get_subclass(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The RNA type or default when not found.

RETURN TYPE:

bpy.types.Struct subclass

classmethod bl_rna_get_subclass_py(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The class or default when not found.

RETURN TYPE:

type

load(filepath, link=False, relative=False, assets_only=False, create_liboverrides=False, reuse_liboverrides=False, create_liboverrides_runtime=False)

Returns a context manager which exposes 2 library objects on entering. Each object has attributes matching bpy.data which are lists of strings be linked.

PARAMETERS:

- **filepath** (*str* | *bytes*) – The path to a blend file.
- **link** (*bool*) – When False reference to the original file is lost.
- **relative** (*bool*) – When True the path is stored relative to the open blend file.
- **assets_only** (*bool*) – If True, only list data-blocks marked as assets.
- **create_liboverrides** (*bool*) – If True and link is True, liboverrides will be created for linked data.
- **reuse_liboverrides** (*bool*) – If True and create_liboverride is True, search for existing liboverride first.
- **create_liboverrides_runtime** (*bool*) – If True and create liboverride is True, create (or search for existing) runtime

liboverride.

```
import bpy

filepath = "//link_library.blend"

# load a single scene we know the name of.
with bpy.data.libraries.load(filepath) as (data_from, data_to):
    data_to.scenes = ["Scene"]


# load all meshes
with bpy.data.libraries.load(filepath) as (data_from, data_to):
    data_to.meshes = data_from.meshes


# link all objects starting with 'A'
with bpy.data.libraries.load(filepath, link=True) as (data_from, data_to):
    data_to.objects = [name for name in data_from.objects if name.startswith("A")]


# append everything
with bpy.data.libraries.load(filepath) as (data_from, data_to):
    for attr in dir(data_to):
        setattr(data_to, attr, getattr(data_from, attr))


# the loaded objects can be accessed from 'data_to' outside of the context
# since loading the data replaces the strings for the datablocks or None
# if the datablock could not be loaded.
with bpy.data.libraries.load(filepath) as (data_from, data_to):
    data_to.meshes = data_from.meshes
# now operate directly on the loaded data
for mesh in data_to.meshes:
    if mesh is not None:
        print(mesh.name)
```

write(filepath, datablocks, path_remap=False, fake_user=False, compress=False)

Write data-blocks into a blend file.

Note

Indirectly referenced data-blocks will be expanded and written too.

PARAMETERS:

- **filepath** (*str* | *bytes*) – The path to write the blend-file.
- **datablocks** (set[`bpy.types.ID`]) – set of data-blocks.
- **path_remap** (*str*) –

  Optionally remap paths when writing the file:

  - `NONE` No path manipulation (default).
  - `RELATIVE` Remap paths that are already relative to the new location.
  - `RELATIVE_ALL` Remap all paths to be relative to the new location.
  - `ABSOLUTE` Make all paths absolute on writing.

- **fake_user** (*bool*) – When True, data-blocks will be written with fake-user flag enabled.
- **compress** (*bool*) – When True, write a compressed blend file.

```
import bpy

filepath = "//new_library.blend"

# write selected objects and their data to a blend file
data_blocks = set(bpy.context.selected_objects)
bpy.data.libraries.write(filepath, data_blocks)


# write all meshes starting with a capital letter and
# set them with fake-user enabled so they aren't lost on re-saving
data_blocks = {mesh for mesh in bpy.data.meshes if mesh.name[:1].isupper()}
bpy.data.libraries.write(filepath, data_blocks, fake_user=True)


# write all materials, textures and node groups to a library
data_blocks = {*bpy.data.materials, *bpy.data.textures, *bpy.data.node_groups}
bpy.data.libraries.write(filepath, data_blocks)
```

Inherited Properties

- bpy_struct.id_data

Inherited Functions

- bpy_struct.as_pointer
- bpy_struct.driver_add
- bpy_struct.driver_remove
- bpy_struct.get
- bpy_struct.id_properties_clear
- bpy_struct.id_properties_ensure
- bpy_struct.id_properties_ui
- bpy_struct.is_property_hidden
- bpy_struct.is_property_overridable_library
- bpy_struct.is_property_readonly
- bpy_struct.is_property_set
- bpy_struct.items
- bpy_struct.keyframe_delete
- bpy_struct.keyframe_insert
- bpy_struct.keys
- bpy_struct.path_from_id
- bpy_struct.path_resolve
- bpy_struct.pop
- bpy_struct.property_overridable_library_set
- bpy_struct.property_unset
- bpy_struct.type_recast
- bpy_struct.values

References

- BlendData.libraries