Menu(bpy_struct)

Basic Menu Example

Here is an example of a simple menu. Menus differ from panels in that they must reference from a header, panel or another menu.

Notice the 'CATEGORY_MT_name' in `Menu.bl_idname`, this is a naming convention for menus.

> Note
>
> Menu subclasses must be registered before referencing them from blender.

> Note
>
> Menus have their `Layout.operator_context` initialized as 'EXEC_REGION_WIN' rather than 'INVOKE_DEFAULT' (see Execution Context). If the operator context needs to initialize inputs from the `Operator.invoke` function, then this needs to be explicitly set.

```
import bpy


class BasicMenu(bpy.types.Menu):
    bl_idname = "OBJECT_MT_select_test"
    bl_label = "Select"

    def draw(self, context):
        layout = self.layout

        layout.operator("object.select_all", text="Select/Deselect All").action = 'TOGGLE'
        layout.operator("object.select_all", text="Inverse").action = 'INVERT'
        layout.operator("object.select_random", text="Random")


bpy.utils.register_class(BasicMenu)

# test call to display immediately.
bpy.ops.wm.call_menu(name="OBJECT_MT_select_test")
```

Submenus

This menu demonstrates some different functions.

```
import bpy


class SubMenu(bpy.types.Menu):
    bl_idname = "OBJECT_MT_select_submenu"
    bl_label = "Select"

    def draw(self, context):
        layout = self.layout

        layout.operator("object.select_all", text="Select/Deselect All").action = 'TOGGLE'
        layout.operator("object.select_all", text="Inverse").action = 'INVERT'
        layout.operator("object.select_random", text="Random")
```

```
        # Access this operator as a sub-menu.
        layout.operator_menu_enum("object.select_by_type", "type", text="Select All by Typ

        layout.separator()

        # Expand each operator option into this menu.
        layout.operator_enum("object.light_add", "type")

        layout.separator()

        # Use existing menu.
        layout.menu("VIEW3D_MT_transform")


bpy.utils.register_class(SubMenu)

# test call to display immediately.
bpy.ops.wm.call_menu(name="OBJECT_MT_select_submenu")
```

Extending Menus

When creating menus for add-ons you can't reference menus in Blender's default scripts. Instead, the add-on can add menu items to existing menus.

The function menu_draw acts like `Menu.draw`.

```
import bpy


def menu_draw(self, context):
    self.layout.operator("wm.save_homefile")


bpy.types.TOPBAR_MT_file.append(menu_draw)
```

Preset Menus

Preset menus are simply a convention that uses a menu sub-class to perform the common task of managing presets.

This example shows how you can add a preset menu.

This example uses the object display options, however you can use properties defined by your own scripts too.

```
import bpy
from bpy.types import Operator, Menu
from bl_operators.presets import AddPresetBase


class OBJECT_MT_display_presets(Menu):
    bl_label = "Object Display Presets"
    preset_subdir = "object/display"
    preset_operator = "script.execute_preset"
    draw = Menu.draw_preset


class AddPresetObjectDisplay(AddPresetBase, Operator):
```

```
class AddPresetObjectDisplay(AddPresetBase, Operator):
    '''Add a Object Display Preset'''
    bl_idname = "camera.object_display_preset_add"
    bl_label = "Add Object Display Preset"
    preset_menu = "OBJECT_MT_display_presets"

    # variable used for all preset values
    preset_defines = [
        "obj = bpy.context.object"
    ]

    # properties to store in the preset
    preset_values = [
        "obj.display_type",
        "obj.show_bounds",
        "obj.display_bounds_type",
        "obj.show_name",
        "obj.show_axis",
        "obj.show_wire",
    ]

    # where to store the preset
    preset_subdir = "object/display"


# Display into an existing panel
def panel_func(self, context):
    layout = self.layout

    row = layout.row(align=True)
    row.menu(OBJECT_MT_display_presets.__name__, text=OBJECT_MT_display_presets.bl_label)
    row.operator(AddPresetObjectDisplay.bl_idname, text="", icon='ZOOM_IN')
    row.operator(AddPresetObjectDisplay.bl_idname, text="", icon='ZOOM_OUT').remove_active


classes = (
    OBJECT_MT_display_presets,
    AddPresetObjectDisplay,
)


def register():
    for cls in classes:
        bpy.utils.register_class(cls)
    bpy.types.OBJECT_PT_display.prepend(panel_func)


def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)
    bpy.types.OBJECT_PT_display.remove(panel_func)


if __name__ == "__main__":
    register()
```

Extending the Button Context Menu

This example enables you to insert your own menu entry into the common right click menu that you get while hovering over a UI button (e.g. operator, value field, color, string, etc.)

To make the example work, you have to first select an object then right click on an user interface element (maybe a color in the material properties) and choose *Execute Custom Action*.

Executing the operator will then print all values.

```
import bpy


def dump(obj, text):
    for attr in dir(obj):
        print("{!r}.{:s} = {:s}".format(obj, attr, getattr(obj, attr)))


class WM_OT_button_context_test(bpy.types.Operator):
    """Right click entry test"""
    bl_idname = "wm.button_context_test"
    bl_label = "Run Context Test"

    @classmethod
    def poll(cls, context):
        return context.active_object is not None

    def execute(self, context):
        value = getattr(context, "button_pointer", None)
        if value is not None:
            dump(value, "button_pointer")

        value = getattr(context, "button_prop", None)
        if value is not None:
            dump(value, "button_prop")

        value = getattr(context, "button_operator", None)
        if value is not None:
            dump(value, "button_operator")

        return {'FINISHED'}


def draw_menu(self, context):
    layout = self.layout
    layout.separator()
    layout.operator(WM_OT_button_context_test.bl_idname)


def register():
    bpy.utils.register_class(WM_OT_button_context_test)
    bpy.types.UI_MT_button_context_menu.append(draw_menu)


def unregister():
```

```
    bpy.types.UI_MT_button_context_menu.remove(draw_menu)
    bpy.utils.unregister_class(WM_OT_button_context_test)


if __name__ == "__main__":
    register()
```

base class — bpy_struct

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

Editor menu containing buttons

bl_description

TYPE:

string, default ‘’

bl_idname

If this is set, the menu gets a custom ID, otherwise it takes the name of the class used to define the menu (for example, if the class name is “OBJECT_MT_hello”, and bl_idname is not set by the script, then bl_idname = “OBJECT_MT_hello”)

TYPE:

string, default ‘’, (never None)

bl_label

The menu label

TYPE:

string, default ‘’, (never None)

bl_options

Options for this menu type

- SEARCH_ON_KEY_PRESS Search on Key Press – Open a menu search when a key pressed while the menu is open.

TYPE:

enum set in {‘SEARCH_ON_KEY_PRESS’}, default {‘SEARCH_ON_KEY_PRESS’}

bl_owner_id

TYPE:

string, default ‘’, (never None)

bl_translation_context

TYPE:

string, default ‘*’, (never None)

layout

Defines the structure of the menu in the UI

TYPE:

UILayout, (readonly)

classmethod poll(context)

If this method returns a non-null output, then the menu can be drawn

RETURN TYPE:

boolean

draw(context)

Draw UI elements into the menu UI layout

classmethod append(draw_func)

Append a draw function to this menu, takes the same arguments as the menus draw function

classmethod draw_collapsible(context, layout)

draw_preset(_context)

Define these on the subclass: - preset_operator (string) - preset_subdir (string)

Optionally: - preset_add_operator (string) - preset_extensions (set of strings) - preset_operator_defaults (dict of keyword args)

classmethod is_extended()

path_menu(searchpaths, operator, *, props_default=None, prop_filepath='filepath', filter_ext=None, filter_path=None, display_name=None, add_operator=None, add_operator_props=None)

Populate a menu from a list of paths.

PARAMETERS:

- **searchpaths** (*Sequence[str]*) – Paths to scan.
- **operator** (*str*) – The operator id to use with each file.
- **prop_filepath** (*str*) – Optional operator filepath property (defaults to “filepath”).
- **props_default** (*dict[str, Any]*) – Properties to assign to each operator.
- **filter_ext** (*Callable[[str], bool] | None*) –

  Optional callback that takes the file extensions.

  Returning false excludes the file from the list.
- **display_name** (*Callable[[str], str]*) – Optional callback that takes the full path, returns the name to display.

classmethod prepend(draw_func)

Prepend a draw function to this menu, takes the same arguments as the menus draw function

classmethod remove(draw_func)

Remove a draw function that has been added to this menu

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

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