Operators (bpy.ops)

Calling Operators

Provides python access to calling operators, this includes operators written in C, Python or macros.

Only keyword arguments can be used to pass operator properties.

Operators don't have return values as you might expect, instead they return a set() which is made up of: `{'RUNNING_MODAL', 'CANCELLED' 'FINISHED', 'PASS_THROUGH'}`. Common return values are `{'FINISHED'}` and `{'CANCELLED'}`, the latter meaning that the operator execution was aborted without making any changes or saving an undo history entry.

Calling an operator in the wrong context will raise a `RuntimeError`, there is a poll() method to avoid this problem.

Note that the operator ID (bl_idname) in this example is `mesh.subdivide`, `bpy.ops` is just the access path for python.

Keywords and Positional Arguments

For calling operators keywords are used for operator properties and positional arguments are used to define how the operator is called.

There are 2 optional positional arguments (documented in detail below).

```
bpy.ops.test.operator(execution_context, undo)
```

- execution_context - `str` (enum).
- undo - `bool` type.

Each of these arguments is optional, but must be given in the order above.

```
import bpy

# calling an operator
bpy.ops.mesh.subdivide(number_cuts=3, smoothness=0.5)


# check poll() to avoid exception.
if bpy.ops.object.mode_set.poll():
    bpy.ops.object.mode_set(mode='EDIT')
```

Overriding Context

It is possible to override context members that the operator sees, so that they act on specified rather than the selected or active data, or to execute an operator in the different part of the user interface.

The context overrides are passed in as keyword arguments, with keywords matching the context member names in `bpy.context`. For example to override `bpy.context.active_object`, you would pass `active_object=object` to `bpy.types.Context.temp_override`.

> Note
>
> You will nearly always want to use a copy of the actual current context as basis (otherwise, you'll have to find and gather all needed data yourself).

> Note
>
> Context members are names which Blender uses for data access, overrides do not extend to overriding methods or any Python specific functionality.

```
# Remove all objects in scene rather than the selected ones.
```

```
import bpy
from bpy import context
context_override = context.copy()
context_override["selected_objects"] = list(context.scene.objects)
with context.temp_override(**context_override):
    bpy.ops.object.delete()
```

Execution Context

When calling an operator you may want to pass the execution context.

This determines the context that is given for the operator to run in, and whether invoke() is called or only execute().

`EXEC_DEFAULT` is used by default, running only the `execute()` method, but you may want the operator to take user interaction with `INVOKE_DEFAULT` which will also call invoke() if existing.

The execution context is one of:

- `INVOKE_DEFAULT`
- `INVOKE_REGION_WIN`
- `INVOKE_REGION_CHANNELS`
- `INVOKE_REGION_PREVIEW`
- `INVOKE_AREA`
- `INVOKE_SCREEN`
- `EXEC_DEFAULT`
- `EXEC_REGION_WIN`
- `EXEC_REGION_CHANNELS`
- `EXEC_REGION_PREVIEW`
- `EXEC_AREA`
- `EXEC_SCREEN`

```
# collection add popup
import bpy
bpy.ops.object.collection_instance_add('INVOKE_DEFAULT')
```

It is also possible to run an operator in a particular part of the user interface. For this we need to pass the window, area and sometimes a region.

```
# Maximize 3d view in all windows.
import bpy
from bpy import context

for window in context.window_manager.windows:
    screen = window.screen
    for area in screen.areas:
        if area.type == 'VIEW_3D':
            with context.temp_override(window=window, area=area):
                bpy.ops.screen.screen_full_area()
            break
```

SUBMODULES

Action Operators

Anim Operators

Armature Operators

Asset Operators

Previous
Message Bus (bpy.msgbus)
Report issue on this page

N
Action Operat