Skip to content

Keymap Customization

Keys

Available Keys

When customizing keymaps it's useful to use keys which won't conflict with Blender's default keymap.

Here are keys which aren't used and aren't likely to be used in the future.

F-Keys (`F5` - `F8`)
: These F-keys (including modifier combination) have been intentionally kept free for users to bind their own keys to.

`OSKey` (also known as the `Windows-Key`, `Cmd` or `Super`)
: Blender doesn't use this key for any bindings.

  macOS is an exception, where `Cmd` replaces `Ctrl` except in cases it would conflict with the system's key bindings.

Modifier Double Click
: Binding modifier keys as primary keys is supported, to avoid conflicts with regular usage you can bind them to double click.

Multi-Action Keys

Click/Drag

It's possible to configure a single key to perform multiple operations using *Click* event instead of *Press*. Then you may bind *Drag* to a separate action.

This is useful for mixing actions where one uses a drag event, e.g: Toggle a setting using with `Tab`, drag to open a pie menu showing all options related to the setting.

This is used in the default keymap in the 3D Viewport, `Alt - MMB` dragging in different directions rotates the view.

Common Operations

This section lists useful generic operations which can be used.

Key Bindings for Pop-Ups

Menus and panels can be assigned key shortcuts, even if they're only accessible from submenus elsewhere.

Open a Pop-up Menu (`wm.call_menu`)
: Open any menu on key press.

Open a Pie Menu (`wm.call_menu_pie`)
: Open any pie menu on key press.

Open a Panel (`wm.call_panel`)
: Open a pop-up panel (also known as a pop-over).

Menu & Panel Identifiers

To find the `name` of a menu, enable the preference Interface ▸ Display ▸ Python Tooltips.

Then hover the cursor over the popover button or menu item. For submenus you will need to use the back arrow to prevent the submenu from opening and gaining focus.

Key Bindings for Properties

There are many properties you might want to bind a key with. To avoid having to define operators for each property, there are generic operators for this

purpose:

Operators for adjusting properties begin with `wm.context_`.

Some of these include:

- `wm.context_toggle` toggle a Boolean property.
- `wm.context_cycle_enum` cycle an enum property forwards or backwards.
- `wm.context_menu_enum` show a pop-up menu for an enum property.
- `wm.context_pie_enum` show a pie menu for an enum property.
- `wm.context_scale_float` scale a number (used for increasing / decreasing brush size for example).
- `wm.context_toggle_enum` toggle between two options of an enum.
- `wm.context_modal_mouse` moving the cursor to interactively change a value.

See bpy.ops.wm for a complete list.

Each of these operators has a `data_path` setting to reference the property to change.

To find the `data_path`, basic Python knowledge is needed.

For example, you can use the Python Console to access a Boolean property you wish to map to a key:

```
bpy.context.object.show_name
```

To bind this to a key, add a new keymap item using the operator `wm.context_toggle` with `data_path` set to `object.show_name` (notice the `bpy.context` prefix is implicit).

See bpy.context for other context attributes.

The Python API documentation can be used to find properties or you may use the Python Console's auto-complete to inspect available properties.