Skip to content

Application Templates

Usage

Application templates are a feature that allows you to define a re-usable configuration that can be selected to replace the default configuration, without requiring a separate Blender installation or overwriting your personal settings.

Application templates can be selected from the splash screen or File ‣ New submenu. When there are no templates found the menu will not be displayed on the splash screen.

New application templates can be installed from the Blender Menu. If you would like to keep the current application template active on restarting Blende save your preferences.

Motivation

In some cases it's not enough to write a single script or add-on, and expect someone to replace their preferences and startup file, install scripts and chan their keymap.

The goal of application templates is to support switching to a customized configuration without disrupting your existing settings and installation. This mean people can build their own *applications* on top of Blender that can be easily distributed.

Details

An application template may define its own:

Startup File
: The default file to load with this template.

Preferences
: Only certain preferences from a template are used:

- Themes.
- Add-ons.
- Keymaps.
- Viewport lighting.

Splash Screen
: Templates may provide their own splash screen image.

Python Scripts
: While templates have access to the same functionality as any other scripts, typical operations include:

- Modifying and replacing parts of the user interface.
- Defining new menus, keymaps and tools.
- Defining a custom add-on path for template specific add-ons.

Templates also have their own user configuration, so saving a startup file while using a template won't overwrite your default startup file.

Directory Layout

Templates may be located in one of two locations within the `scripts` directory.

Template locations:

```
{BLENDER_USER_SCRIPTS}/startup/bl_app_templates_user
{BLENDER_SYSTEM_SCRIPTS}/startup/bl_app_templates_system
```

User configuration is stored in a subdirectory:

Without a template:

```
./config/startup.blend
./config/userpref.blend
```

With a template:

```
./config/{APP_TEMPLATE_ID}/startup.blend
./config/{APP_TEMPLATE_ID}/userpref.blend
```

See Blender's Directory Layout for details on script and configuration locations.

Hint

Troubleshooting Paths

When creating an application template, you may run into issues where paths are not being found. To investigate this you can log output of all of Blender's path look-ups.

Example command line arguments that load Blender with a custom application template (replace `my_app_template` with the name of your own template):

```
blender --log "bke.appdir.*" --log-level -1 --app-template my_app_template
```

You can then check the paths where attempts to access `my_app_template` are made.

Command Line Access

Using the command-line arguments you can setup a launcher that opens Blender with a specific app template:

```
blender --app-template my_template
```

Template Contents

Each of the following files can be used for application templates but are optional.

`startup.blend`
: Factory startup file to use for this template.

`userpref.blend`
: Factory preferences file to use for this template. When omitted preferences are shared with the default Blender configuration.

(As noted previously, this is only used for a subset of preferences).

`splash.png`
: Splash screen to override Blender's default artwork (not including header text). Note, this image must be a `1000x500` image.

`__init__.py`
: A Python script which must contain `register` and `unregister` functions.

Note

Bundled blend-files `startup.blend` and `userpref.blend` are considered *Factory Settings* and are never overwritten.

The user may save their own startup/preferences while using this template which will be stored in their user configuration, but only when the template includes its own `userpref.blend` file.

The original template settings can be loaded using: *Load Template Factory Settings* from the file menu in much the same way *Load Factory Settings* works.

Template Scripts

While app templates can use Python scripts, they simply have access to the same APIs available for add-ons and any other scripts.

As noted above, you may optionally have an `__init__.py` in your app template. This has the following advantages:

- Changes can be made to the startup or preferences, without having to distribute a blend-file.
- Changes can be made dynamically.

  You could for example – configure the template to check the number of processors, operating system and memory, then set values based on this.
- You may enable add-ons associated with your template.

On activation a `register` function is called, `unregister` is called when another template is selected.

As these only run once, any changes to defaults must be made via handler. Two handlers you are likely to use are:

- `bpy.app.handlers.load_factory_preferences_post`
- `bpy.app.handlers.load_factory_startup_post`

These allow you to define your own "factory settings", which the user may change, just as Blender has it's own defaults when first launched.

This is an example `__init__.py` file which defines defaults for an app template to use.

```
import bpy
from bpy.app.handlers import persistent

@persistent
def load_handler_for_preferences(_):
    print("Changing Preference Defaults!")
    from bpy import context

    prefs = context.preferences
    prefs.use_preferences_save = False

    kc = context.window_manager.keyconfigs["blender"]
    kc_prefs = kc.preferences
    if kc_prefs is not None:
        kc_prefs.select_mouse = 'RIGHT'
        kc_prefs.spacebar_action = 'SEARCH'
        kc_prefs.use_pie_click_drag = True

    view = prefs.view
    view.header_align = 'BOTTOM'


@persistent
def load_handler_for_startup(_):
    print("Changing Startup Defaults!")

    # Use smooth faces.
    for mesh in bpy.data.meshes:
        for poly in mesh.polygons:
            poly.use_smooth = True

    # Use material preview shading.
    for screen in bpy.data.screens:
        for area in screen.areas:
            for space in area.spaces:
                if space.type == 'VIEW_3D':
                    space.shading.type = 'MATERIAL'
                    space.shading.use_scene_lights = True
```

```python
def register():
    print("Registering to Change Defaults")
    bpy.app.handlers.load_factory_preferences_post.append(load_handler_for_preferences)
    bpy.app.handlers.load_factory_startup_post.append(load_handler_for_startup)

def unregister():
    print("Unregistering to Change Defaults")
    bpy.app.handlers.load_factory_preferences_post.remove(load_handler_for_preferences)
    bpy.app.handlers.load_factory_startup_post.remove(load_handler_for_startup)
```

Previous
Creating a Dynamic Extensions Repository

N
Keymap Customizati

Copyright © : This page is licensed under a CC-BY-SA 4.0 Int. License
Made with Furo
Last updated on 2025-05-10

View Source
View Translation
Report issue on this page