Write Editor Tests with Python

Learn how to create Editor Tests with Python.

The **PythonAutomationTest** plugin is required. To enable it, follow these steps:

1. Select **Edit > Plugins** to open the **Plugin** panel.
2. Use the search bar to find the plugin.
3. Enable the corresponding checkbox.
4. Restart Unreal Editor.

PythonAutomationTest Plugin

The PythonAutomationTest plugin automatically discovers Python scripts from the current project and adds them as tests in the **Test Automation** window. The plugin wraps the Python script so the automation test system traps failures and provides ways to cope with latent commands.

Once enabled, the plugin includes Python scripts named with the pattern `test_*.py` and located inside the project and plugin `/Content/Python` folder. The sub-folder structure is parsed and reflected in the test name, which will result in placing each test under:

`Editor > Python > Project/Plugin Name > Sub-folder Structure > Test Name`

In the Python script itself, anything goes. You can execute it through **File > Execute Python Script**. The test results contain all exceptions, log errors, and warnings.

Handling Latent Commands

If you call a non-blocking function, the Editor needs to tick for it to complete.

The Editor does not tick during Python script execution, so you need to release the global interpreter lock (GIL), schedule a post-Slate callback, and call `python` again. One way to do this is to use the Python automation scheduler in the following way:

```
1 import unreal
2 @unreal.AutomationScheduler.add_latent_command
3 def load_some_stuff():
4 pass
```

Copy full snippet

The instructions execute in the order you register them. You can schedule dynamically by registering a generator or returning a function or generator from the callback.

```
@unreal.AutomationScheduler.add_latent_command
def do_some_stuff():
print('initiate')

yield
print('start loop')

for i in xrange(10):
print('loop %d'% i)
task = get_a_task_somehow()
while not task.is_task_done():
yield

print('task %d done'% i)
```

Copy full snippet

Reporting Errors

There are three ways you can report errors during test execution.

- Using Python `assert` - Blocks execution of any further code from the current test and includes the Python call stack in the test report.
- `raise Exception("<error>")` - Blocks execution of any further code from the current test and reports the exception.
- Log the error (with Editor command or `unreal.log_error("<error>")`) - Includes the message in the test report and continues execution.

Expected Log Errors

You can leverage expected C++ framework errors through the following Python code:

```
unreal.AutomationLibrary.add_expected_log_error(expected_pattern_string, occurrences=1, exact_match=False)
```

Copy full snippet

This function matches errors and warnings. Successive calls with the same pattern do not accumulate count. The number of occurrences needs to be accurate the first time, as a different number of occurrences will report a failure.

Screenshot Support

You can take screenshots through a Python test with the automation scheduler.

The `take_high_res_screenshot` function from the automation library requests a high-resolution screenshot. However, an additional Editor tick is required to complete the screenshot, so you must use the scheduler to pause the test before proceeding to the next step.

Example

The following example takes multiple screenshots from different cameras after loading a level.

```
1 import unreal
2
3 @unreal.AutomationScheduler.add_latent_command
4 def setup_level():
```

```
unreal.EditorLevelLibrary.load_level("/Game/mymap")

@unreal.AutomationScheduler.add_latent_command
def take_all_cam_screenshots():
level_actors = unreal.EditorLevelLibrary.get_all_level_actors()
all_cameras = unreal.EditorFilterLibrary.by_class(
level_actors,
unreal.CameraActor
)

for cam in all_cameras:
camera_name = cam.get_actor_label()
task = unreal.AutomationLibrary.take_high_res_screenshot(1280, 720, camera_name, camera=cam,)
if not task.is_valid_task():
continue

print ('Requested screenshot for '+ camera_name )
while not task.is_task_done():
yield
```

Copy full snippet

Image Comparison Support

You can compare image files to a test reference by using the following command:

```
unreal.AutomationLibrary.compare_image_against_reference(image_file_path, comparison_name, comparison_tolerance)
```

Copy full snippet

The function returns true if the image is read and queues for comparison successfully. Queued comparisons are evaluated at the end of the test.

Telemetry Support

You can store telemetry data through a Python test with the following function:

```
unreal.AutomationLibrary.add_test_telemetry_data(datapoint, measurement, context)
```

Copy full snippet

> The `measurement` parameter is a float.

You can change the storage name for the test with the following function:

```
unreal.AutomationLibrary.set_test_telemetry_storage(name)
```

Copy full snippet