Python

Download

Download Reporting SDK client for your programming languages and framework.

Note: The Reporting SDK requires installation of Python 2.7.x, and its definition as a dependency.

On this page:

Smart Reporting setup

The following code changes are required to include reporting in your test:

Note: The below snippets are a standalone script that does not use any execution testing framework.

Mandatory

Imports

Add the following import statement to the test script:

```
from perfecto import PerfectoExecutionContext, TestResultFactory,
TestContext, PerfectoReportiumClient
from perfecto import model
```

Create an instance of the reporting client

Create an Execution Context (Optional)

The ***PerfectoExecutionContext*** instance for your test suite, allows the tester to add some global information that will identify the test suite and allow the user to quickly focus on the test reports for this suite. The Execution Context supports:

- **Tags** - used as a freestyle text for filtering the reports in the Report Library grid.
- **CI Job** and **Project** information - job information is used to add your test runs to the CI Dashboard. The job information includes <Job Name, Branch Name, Job Number> and project information includes <Project Name, Project Version>
- **Custom Fields** - tags that are <name, value> pairs and can be used to identify and filter your different test runs.

Create an instance of the ReportiumClient used to log test start/step/end events:

```
# By default context_tags, job, project will be set to None and custom fields is an empty list
def create_reporting_client(self):
# In this example the information is taken from a global set of data
        cf1 = model.CustomField(cFields[0]['name'], cFields[0]['value'])
        cf2 = model.CustomField(cFields[1]['name'], cFields[1]['value'])
    perfecto_execution_context = PerfectoExecutionContext(webdriver=self.driver,
                                                              tags=[tags[0], tags[1], tags[2], tags[3]],
                                                              job=model.Job(job['name'], job['number'], job
['branch']),
                                                              project=model.Project(project['name'], project
['version']),
                                                                                            customFields=[cf1,
cf2])
    self.reporting_client = PerfectoReportiumClient(perfecto_execution_context)
```

Add test information and steps

Start a new test

Use the **ReportiumClient**'s ***test_start***() method to start a specific test, and optionally supply a **TestContext** to add *custom fields* and *tags* that will be applied to this particular test, within the test suite.
For example:

```
cf1 = model.CustomField(key1, val1)
cf2 = model.CustomField(key2, val2)
self.reporting_client.test_start(test_data['name'], TestContext(customFields=[cf1, cf2], tags=[in_tags[0],
in_tags[1]]))
```

Add test steps

Separate your test into actions using test steps.

```
self.reporting_client.step_start('Navigate to Perfecto dev portal')
self.driver.get('developers.perfectomobile.com')
self.reporting_client.step_end()
```

Add assertions to the execution report

At various points within a test execution, the script may perform verification of different test conditions. The result of these verification may be added to the Test Report by using the reportiumAssertion*()* method of the ReportiumClient instance. When using this method, the script includes two parameters:

- A message string - that will be used to label the assertion.
- A Boolean - indicates the result of the verification operation.

```
self.reporting_client.step_start('Search: PerfectoCode GitHub repo - Should fail')
self.driver.find_element(By.NAME, 'qq').send_keys('PerfectoCode GitHub')
self.reporting_client.reportium_assert('assert we are in the expected url', self.driver.current_url == self.
expected_url)
self.reporting_client.step_end()
```

Stop the test

When the test is completed, supply an indication of the final outcome of the test by generating a ***TestResult*** instance. The *TestResultFactory* class supports:

- **createSuccess** method - that notifies the reporting server that the test resulted in a successful status.
- **createFailure** method - that notifies the reporting server that the test resulted in a unsuccessful status and supports adding a notification message that is displayed in the test report.
    - You can also provide a *failure reason,* or depend on the Smart Reporting analysis to identify the failure reason.

```
try:
    ...//selenium code

    if self.currentResult.wasSuccessful():
        self.reporting_client.test_stop(TestResultFactory.create_success())
    else:
        self.reporting_client.test_stop(TestResultFactory.create_failure(self.currentResult.message,
                                                                         self.currentResult.exception,
                                                                         self.currentResult.failReason))
        # Print report's url
    print 'Report-Url: ' + self.reporting_client.report_url() + '\n'
except Exception as e:
    self.reporting_client.test_stop(TestResultFactory.create_failure(e.message,
                                                                     e, self.exceptionResult.failReason))

self.driver.quit()
```

In addition to providing the status of the test result, it is possible to provide additional *tags* and *custom fields* to the test - this may be used, for example, to add indications of the paths that caused the result or the reason for stopping the test. Use the ***TestContext*** to add additional *tags* and *cust om fields*:

```
cf1 = model.CustomField(key1, val1)
cf2 = model.CustomField(key2, val2)
tec = TestContext(customFields=[cf1, cf2], tags=[success_tags[0], success_tags[1]]))
self.reporting_client.test_stop(TestResultFactory.create_success(), tec)
```

Note: Adding the ***TestContext*** to the *test_stop* is optional.

Get the report URL

```
//retrieve the report URL
print 'Report-Url: ' + self.reporting_client.report_url() + '\n'
```

GitHub samples

unittest

Browse the Perfecto GitHub repo for complete Python Reporting samples.