Java

This section describes the code changes you need to make in your tests to integrate it with Perfecto Smart Reporting. As a prerequisite, you need to download the Reporting SDK client for your programming languages and framework.

On this page:

Mandatory changes

Mandatory changes include adding import statements and creating an instance of the reporting client.

Imports

Add the following import statements to the base test class:

```
import com.perfecto.reportium.client.ReportiumClient;
import com.perfecto.reportium.client.ReportiumClientFactory;
import com.perfecto.reportium.exception.ReportiumException;
import com.perfecto.reportium.model.PerfectoExecutionContext;
import com.perfecto.reportium.model.Project;
import com.perfecto.reportium.test.TestContext;
import com.perfecto.reportium.test.result.TestResultFactory;
```

Create an instance of the reporting client

Use the ***ReportiumClientFactory*** class' *createPerfectoReportiumClient()* method to create the Smart Reporting ***ReportiumClient*** instance. The reporting client is responsible to gather basic information about the test and transmit it to the Smart Reporting system

Before creating the ReportiumClient, you should create the instance of the automation driver (either SeleniumDriver or one of the Appium drivers), and use a ***PerfectoExecutionContext*** to supply the link to the factory class creating the client instance. Use the *withWebDriver()* method to supply the link of the driver instance. The context supports all of the optional settings described below, in addition to supplying the driver link. Use the build() method to create the context object's instance and supply this to the *createPerfectoReportiumClient()* method when creating the ***ReportiumClient*** instance.

```
PerfectoExecutionContext perfectoExecutionContext = new PerfectoExecutionContext.
PerfectoExecutionContextBuilder()
    .withProject(new Project("Sample Reportium project", "1.0"))
        .withJob(new Job("IOS tests", 45).withBranch("branch-name"))
    .withCustomFields(new CustomField("programmer", "Samson"))
    .withContextTags("Regression")
    .withWebDriver(driver)
    .build();
ReportiumClient reportiumClient = new ReportiumClientFactory().createPerfectoReportiumClient
(perfectoExecutionContext);
```

Note: Reporting client should be created in proximity of the RemoteWebDriver creation.

Optional changes (highly recommended)

Add reporting tags

Tags are used as a freestyle text for filtering the reports in the Reporting app.
The tags are added when creating the Reportium client, for example: `.withContextTags("Regression")`

Add CI job information

Job information is used to add your test runs to the CI Dashboard. Use the *withJob()* method of the ***PerfectoExecutionContext*** instance , supplying the Job Name and Job Number, when creating the **ReportiumClient** instance.

Add custom fields

Create new CustomField instances with the **CustomField** class constructor. Use the *withCustomFields()* method to add a collection of Custom Fields to either the *PerfectoExecutionContext* instance, to the specific *TestContext* instance (either at start or end of test) as shown below.

Start a new test

```
@Test
public void myTest() {
reportiumClient.testStart("myTest", new TestContext.Builder()
                                                                    .withTestExecutionTags
("Sanity", "Nightly")
                                                                    .withCustomFields(new
CustomField("version", "OS11"))
                                                                    .build());
...
}
```

Add test steps

Separate your test into logical groupings of actions as logical steps. Each step is labeled and appears in the Execution Report together with the component actions. When implementing the test, indicate the beginning of the logical step with the *stepStart()* method, providing the label of the step, and (optionally) the *stepEnd()* method to indicate the end of the logical step (for the report). The *stepEnd()* method supports an optional message parameter that would be included in the execution report.

```
@Test
public void myTest() {
reportiumClient.testStart("myTest", new TestContext("Sanity"));

//test step - login to app
reportiumClient.stepStart("Login to application");
WebElement username = driver.findElement(By.id("username"));
Username.sendText("myUser");
Driver.findElement(By.name("submit")).click();
reportiumClient.stepEnd();  // the message parameter is optional

//test step - open a premium acct
reportiumClient.stepStart("Open a premium account");
WebElement premiumAccount = driver.findElement(By.id("premium-account"));
assertTrue(premiumAccount.getText(), "PREMIUM");
premiumAccount.click();
reportiumClient.stepEnd("Finished step");

 reportiumClient.stepStart("Transfer funds");
...
}
```

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 *reportiumAssert()* 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.

```
//test step - open a premium acct
reportiumClient.stepStart("Open a premium account");
WebElement premiumAccount = driver.findElement(By.id("premium-account"));
if (premiumAccount.getText().compareToIgnoreCase("premium") {
        reportiumClient.reportiumAssert("Check for Premium account", true);
        premiumAccount.click();
} else {
        reportiumClient.reportiumAssert("Check for Premium account", false);
}
```

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.

```
@Test
public void myTest() {
	String failR = "Element not found";

	reportiumClient.testStart("myTest", new TestContext("Sanity"));

	try {
		reportiumClient.stepStart("Login to application");
		WebElement username = driver.findElement(By.id("username"));
		Username.sendText("myUser");
		Driver.findElement(By.name("submit")).click();
			reportiumClient.stepEnd();

			reportiumClient.stepStart("Open a premium account");
		WebElement premiumAccount = driver.findElement(By.id("premium-account"));
		assertTrue(premiumAccount.getText(), "PREMIUM");
		premiumAccount.click();
			reportiumClient.stepEnd("Finished step");

			reportiumClient.stepStart("Transfer funds");
		...
		//stopping the test - success
		reportiumClient.testStop(TestResultFactory.createSuccess());
	} catch (Throwable t) {
		//stopping the test - failure
		reportiumClient.testStop(TestResultFactory.createFailure(t.getMessage(), t, failR));
	}
}
```

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*:

```
String testStopTag = "Failure reason test";
CustomField testStopCustomField = new CustomField("source", "sdk");
//Code for the test end checking
//stopping the test - failure
TestContext testContextEnd = new TestContext.Builder()
            .withTestExecutionTags(testStopTag)
            .withCustomFields(testStopCustomField)
            .build();
reportiumClient.testStop(TestResultFactory.createFailure(expectedFailureMessage, new RuntimeException
(eMessage), failReason), testContextEnd);
```

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

Get the report URL

```
String reportURL = reportiumClient.getReportUrl();
System.out.println("Report URL - " + reportURL);
```

Single report for multiple executions

There is an option to create a reporting client for multiple drivers. This will result in a single execution report that combines all drivers executions and will result in unified report that will contain all executions as a single merged execution.

This type of flow is for scenarios where you want to create a unified execution to multiple drivers flows - For example: One driver that corresponds to a taxi driver that receives requests from passengers and one driver that corresponds to the passenger that asks for a taxi.

Use the ***ReportiumClientFactory*** class *createPerfectoReportiumClient()* method to create the Smart Reporting ***ReportiumClient*** instance. The reporting client is responsible to gather basic information about the test and transmit it to the Smart Reporting system

Before creating the ReportiumClient, you should create the instance of the automation driver (either SeleniumDriver or one of the Appium drivers), and use a ***PerfectoExecutionContext*** to supply the link to the factory class creating the client instance. Use the *withWebDriver()* method multiple times to supply the link of the drivers instances and their aliases (Optional). The context supports all of the optional settings described below, in addition to supplying the drivers links. Use the build() method to create the context object's instance and supply this to the *createPerfectoReportiumClient()* metho d when creating the ***ReportiumClient*** instance.

```
PerfectoExecutionContext perfectoExecutionContext = new PerfectoExecutionContext.
PerfectoExecutionContextBuilder()
    .withProject(new Project("Sample Reportium project", "1.0"))
        .withJob(new Job("IOS tests", 45).withBranch("branch-name"))
    .withCustomFields(new CustomField("programmer", "Samson"))
    .withContextTags("Regression")
    .withWebDriver(driver1, "Alias1")
    .withWebDriver(driver2, "Alias2")
    .withWebDriver(driver3, "Alias3")
    .build();
ReportiumClient reportiumClient = new ReportiumClientFactory().createPerfectoReportiumClient
(perfectoExecutionContext);
```

GitHub samples

- JBehave
- JUnit
- TestNG
- Main

Browse the Perfecto GitHub repo for complete Java Reporting samples.

JavaDoc

Click here for the complete JavaDoc reporting documentation.