Jasmine

Jasmine is a popular behavior-driven development (BDD) testing framework. Using Jasmine with Perfecto is as easy as setting up a regular Selenium test script.

There are different libs you can use to interact with Perfecto via Jasmine. This section walks you through implementations with:

- Protractor. This is probably the easiest to work with. It supports both synchronized run mode and parallel executions. For a sample project on GitHub, see https://github.com/PerfectoMobileSA/jasmine-protractor-sample.
- Selenium-Webdriver. For a sample project on GitHub, see https://github.com/PerfectoMobileSA/jasmine-selenium-sample.

On this page:

Prerequisites

To create a Jasmine project, you need to have `node.js` and `npm` installed.

1 | Initialize a Jasmine project from scratch

1. Create a project folder.
2. Create a spec folder in the project folder by running the following commands in the project folder, in this order:

```
npm init
```

```
npm install --save-dev jasmine selenium-webdriver protractor perfecto-reporting
```

```
./node_modules/.bin/jasmine init
```

3. In the spec folder, write test scripts with extension `.spec.js` so it will be picked up by Jasmine as test scripts by default.
4. (Optional) To change the Jasmine settings, edit `spec/support/jasmine.json`.

2 | Set up Jasmine to connect to Perfecto

Now that you have set up the empty Jasmine project, you can configure your framework to connect to Perfecto. There are different ways to achieve this. This section focuses on Selenium-Webdriver and Protractor.

Set up with Selenium-Webdriver

You can integrate Jasmine with Perfecto using the Selenium-Webdriver by adding a `helper.js` file. With this solution, you do not need to add code to the test specs, which would impact the readability of your scripts. For example, you can add the following to get the Perfecto remoteWebDriver:

Get Perfecto remoteWebdriver sample

```
var sel = require('selenium-webdriver');
var capabilities = {
'platformName' : 'iOS',
'deviceName' : '<your_device_id>',
'bundleId' : 'com.apple.calculator', //your app bundle id
'browserName' : 'mobileOS',
'securityToken' : '<your_security_token>'
}
var REMOTE_URL = 'https://<your_cloud_name>.perfectomobile.com/nexperience/perfectomobile/wd/hub/fast';
var drv;

exports.launch = async () => {
try{
if (typeof drv == 'undefined') {
//launch the remoteWebDriver
drv = await new sel.Builder().withCapabilities(capabilities).usingServer(REMOTE_URL).build();
}

await drv.manage().setTimeouts({implicit:20000});

});

}catch(e)
{
console.log(e);
}

return drv;
}
```

With the `drv` object, you compose your Jasmine tests. For example:

Sample Jasmine test

```
describe('Test apple calculator', ()=> {

var drv;

beforeAll(async ()=>{
try{
drv = await helper.launch();

}catch(e)
{
console.error(e.message);
throw e;
}
});
var singleRun = ()=>{
it(' test and verify 1 + 2 = 3', async () => {

await drv.findElement(sel.By.xpath('//*[@label="1"]')).click();
await drv.findElement(sel.By.xpath('//*[@label="add"]')).click();
await drv.findElement(sel.By.xpath('//*[@label="2"]')).click();
await drv.findElement(sel.By.xpath('//*[@label="equals"]')).click();
expect(await drv.findElement(sel.By.xpath('//*[@label="Result"]')).getText()).toEqual('3');

});
}
afterAll(async ()=>{
if (typeof drv != 'undefined')
await drv.quit();
});
});
```

Set up with Protractor

Compared to Selenium-Webdriver, it is easier to integrate Jasmine with Perforce using Protractor, regarding both the configuration of parallel execution and the ease of coding. Because Protractor has a built-in synchronized execution mechanism, you do not need to put `async`/`await` ahead of driver commands. You can set the launch and capability settings inside the `config.js` file. For example:

Launch and capabilities settings

```
exports.config = {
seleniumAddress: 'https://<your_cloud_name>.perfectomobile.com/nexperience/perfectomobile/wd/hub/fast',
specs: ['spec/*.spec.js'],
//Use multiCapabilities to achieve parallel execution
multiCapabilities: [
{
'platformName' : 'iOS',
'deviceName' : '<device_id>',
'bundleId' : 'com.apple.calculator',
'browserName' : 'mobileOS',
'securityToken' : '<your_security_token>'
},
{
'platformName' : 'iOS',
'deviceName' : '<device_id>',
'bundleId' : 'com.apple.calculator',
'browserName' : 'mobileOS',
'securityToken' : '<your_security_token>'
}
],
//default page loading timeout in ms
getPageTimeout: 10000,

//set perfecto reporter
onPrepare: async () => {
jasmine.getEnv().addReporter(perfectoReporter);
browser.ignoreSynchronization=true;
var perfectoExecutionContext = await new perfectoReporting.Perfecto.PerfectoExecutionContext({
webdriver: browser,
job: {jobName: "JasmineProtractorPerfectoCI",buildNumber:3},
tags: ['jasmine protractor tests']
});
reportingClient = await new perfectoReporting.Perfecto.PerfectoReportingClient(perfectoExecutionContext);

},
//set jasmine options
jasmineNodeOpts: {
showColors: true,
defaultTimeoutInterval: 1000000,
}
}
```

Following is a sample test case:

Sample Protractor test case

```
describe('Test apple calculator', ()=> {
let EC = protractor.ExpectedConditions;
it(' 1 + 2 = 3', () => {

browser.wait(EC.presenceOf(element(by.xpath('//*[@label="1"]'))),10000);

element(by.xpath('//*[@label="1"]')).click();
element(by.xpath('//*[@label="add"]')).click();
element(by.xpath('//*[@label="2"]')).click();
element(by.xpath('//*[@label="equals"]')).click();
expect(element(by.xpath('//*[@label="Result"]')).getText()).toEqual("3");

});
//teardown on finish all
afterAll(()=>{
browser.close();
});
});
```

3 | Set up Perfecto Smart Reporting for Jasmine

Jasmine has a reporter interface. You can register a customized Reporter by using the following statement:

```
jasmine.getEnv().addReporter(perfectoReporter);
```

The following code block is an example for a customized `perfectoReporter`:

Customized perfectoReporter

```
var perfectoReporter =
{
jasmineStarted: function(suiteInfo) {
// put insome info on jasmine started
},

suiteStarted: (result) => {
// here you can add some custom code to execute when each suite is started
},
specStarted: (result) => {
// each spec will be a test in Perfecto Reporting
reportingClient.testStart(result.fullName);
},
specDone: (result) => {
// ending the test
// here we report about test end event


if (result.status === 'failed') {
// on a failure we report the failure message and stack trace

console.log('Test status is: ' + result.status);
const failure = result.failedExpectations[result.failedExpectations.length - 1];

reportingClient.testStop({
status: perfectoReporting.Constants.results.failed,
message: `${failure.message} ${failure.stack}`
});

} else {
// on success we report that the test has passed
console.log('Test status is: ' + result.status);
reportingClient.testStop({
status: perfectoReporting.Constants.results.passed
});
}
},
suiteDone: (result) => {
// when the suite is done we print in the console its description and status
console.log('Suite done: ' + result.description + ' was ' + result.status);
}

};
```

Note: The `reportingClient` is initialized as follows:

```
var perfectoExecutionContext = await new perfectoReporting.Perfecto.PerfectoExecutionContext({
webdriver: browser,
job: {jobName: "<yourCI job name>",buildNumber:<your CI build number>},
tags: [<your tags>']
});
reportingClient = await new perfectoReporting.Perfecto.PerfectoReportingClient(perfectoExecutionContext);
```