

# Epiphan WUI test automation (WIP)

This document can be found in PDF form [HERE](#)

but the best place to read it is [HERE](#)

Sections marked with a \* may contain information that is old, or incorrect.

## [1] Table of Contents

In this document, the following sections should be present.

- [1] Table of Contents
- [2] About
  - [2.1] Building Blocks\*
  - [2.2] General Architecture\*
  - [2.3] Grid\*
  - [2.4] Tests\*
- [3] Setup
  - [3.1] Git and Maven
    - \* [3.1.1] Steps
    - \* [3.1.2] Notes
  - [3.2] Setting up the Grid
    - \* [3.2.1] Steps
    - \* [3.2.2] Notes
- [4] Use
  - [4.1] Grid Options
  - [4.2] Running Tests\*
  - [4.3] Writing Tests\*

## [2] About

This repository contains the building blocks of a basic web UI test automation framework for use by Epiphan QA.

This framework is meant to provide a simple base upon which to build automated tests for the WUI shared by the VGA Grid, Recorder Pro, and Pearl. Additionally, this framework is meant to overcome some problems with the default Selenium Grid2 system.

This is **not** a framework for testing **anything other than WUI** (but it maybe could be later).

### [2.1] Building Blocks

The Cornerstones of the UI automation framework are:

- [Selenium](#) - A *Web User Interface* (WUI) automation framework which allows Java programs to mimic the interactions of a human with a web page. Selenium implements the W3C [WebDriver](#) spec as it's API. Selenium also provides functionality called Selenium Grid2, which allows multiple machines to be networked together in order to run tests on multiple OS/browser combinations.
- [TestNG](#) - A Java framework for unit and integration testing similar to JUnit. TestNG allows the configuration of tests into 'suites' and 'groups' which provides a great deal of flexibility with regards to which tests to run, and when. TestNG also supports the the passing of parameters to tests at runtime through an xml configuration file.
- [Apache Maven](#) - A flexible Java build system with configurable dependency management.
- [Jenkins CI](#) - A popular *Continuous Integration* (CI) system used to schedule automatic builds and to aggregate build and test reports.

Other components include:

- [ReportNG](#) - Improves on TestNG's default HTML and XML reports.
- [Apache Commons Exec](#) - An Apache Commons library for process execution from within Java programs. Commons Exec allows Java programs to spawn processes, and to continuously monitor the output of those processes while performing other tasks.
- [GitHub](#) - A popular Git repository host.

### [2.2] General Architecture

The Maven project for the automated test system has the following structure:

```
fuzzy-shame
|-grid
|  |-pom.xml
|  |-src
|  |  |-main
|  |      |-java/
|  |-target/
|  |-test-output/
|-tests
|  |-pom.xml
|  |-testng.xml
|  |-src
|  |  |-test
|  |      |-java/
|  |-target/
|  |-test-output/
|-target/
|-test-output/
```

## [2.3] Grid

The actual automated test system (**grid**) is made up of a ‘hub’ and some number of ‘nodes’.

Nodes can be distributed over many machines on the network. This allows the testing of many OS/browser combinations simultaneously. Each node is capable of controlling a configurable number of browser instances. For example, one node could be in control of 10 Firefox browsers, 6 instances of Chrome, and 1 of Internet Explorer. Each of these browser instances is a ‘test slot’ which can accommodate 1 test at a time. The hub is the central point to which the tests and the nodes connect. All test data within the grid flows through the hub. When a test program connects to the hub, it is assigned to a node with an open test slot matching it’s requirements.

Unfortunately, vanilla Selenium nodes are prone to getting stuck under some circumstances. Whenever some event causes a Selenium-controlled browser instance to spawn a new OS-level window (e.g. a Firefox download menu), the WebDriver controlling that browser is unable to close or otherwise eliminate the window. This results in the browser hanging indefinitely (becoming dead).

In order to prevent all test slots from eventually filling with dead browsers, each node can be injected with a custom servlet, and made to communicate with the hub via a custom proxy. The proxy counts the number of tests sent to each node, and once a node has been sent as many tests as it has test slots, the proxy sends a message to the servlet causing the node to terminate.

Every node (including the hub) can also be run through a watchdog process. This watchdog simply checks the node every few seconds, and starts a fresh node

if it finds that the node has terminated. The watchdog does not care if the node was terminated by a crash or by deliberate action.

## [2.4] Tests

The tests to be automated are found in the `tests` module.

Unlike `grid` this module does not produce any permanent artifacts when built. This means that the module must be built from scratch each time tests are run.

The testing framework is based on the open source unit/functional/integration testing library, TestNG.

Tests are stored only in uncompiled source form in order to make the addition, removal, and editing of tests as simple as possible.

Tests are run by invoking the Maven build system with the `test` target. any test output is directed to `/fuzzy-shame/tests/test-output/${timestamp}/` or to some subdirectory thereof.

## [3] Setup

### [3.1] Git and Maven

To be able to write tests, or to make changes to the test system, it is necessary to set up git and maven. Both the build system, as well as the tests rely on maven to run. Git allows the system to be kept in a consistent state even with multiple parties contributing new code, or alterations.

#### [3.1.1] Steps

1. Install the latest version of Maven [version 3.3.3 at 2015-05-13]
2. Install the latest version of Git [version 1.9.1 at 2015-05-13]
3. checkout a copy of the [repository](#).
4. Create a personal branch off the `master` branch of the repository. This will allow you to work without stepping on anybody's toes.  
The idea is that the master branch should always be in a functional state, and developments from testers personal branches are merged in only when they are ready.
5. The repository contains a ready-to-go maven project with everything you need to begin writing and executing tests.
6. If you do not already have a copy of `grid.jar` or `spawn.jar`, build the `grid` module to obtain both.

### [3.1.2] Notes

- The full list of command line options for `grid.jar` can be found in [4.1], or by running `java -jar grid.jar -h`

## [3.2] Setting up the Grid

Before using the automated test system, it is necessary to provide an infrastructure upon which tests can be run. This entails setting up a modified Selenium grid as described in [2.2]

### [3.2.1] Steps

1. Obtain a working copy of the grid package. This can be done by running `mvn clean install` on the `grid` module. The file `grid-<version>.zip` will be created in `/fuzzy-shame/grid/target/`. Pay no attention to `grid-<version>.jar`. It contains all the classes in the `grid` module, and will be installed to your local Maven repository for use in other projects. You may extract `grid-<version>.zip` to a convenient location.
2. Open a shell and enter the directory where you extracted the contents of `grid-<version>.zip`
3. Create a Selenium hub. To do this, run `java -jar spawn.jar grid.jar -role hub`. You may optionally specify the `-port <port>` parameter to set the port on which the hub will listen for incoming connections from nodes (default 4444) and/or the `-hubConfig hubconfig.json` parameter to specify a configuration file for the hub.
4. Create Selenium nodes. The hub cannot run tests on its own, it simply delegates work to nodes which control browser instances. To create a node, run `java -jar spawn.jar grid.jar -role node -hub http://<hub IP address>/grid/register`. You may set the `-nodeConfig nodeconfig.json` parameter to specify a configuration file for the node.

### [3.2.2] Notes

- The reason we run `grid.jar` through `spawn.jar` rather than straight through `java -jar` is that `spawn.jar` acts as a watchdog, restarting dead nodes as described in [2.2]
- When running through `spawn.jar`, provide program arguments just as you normally would. The full list of command line arguments for `grid.jar` can be found by running `java -jar grid.jar -h`

- If you wish to test against browsers other than firefox, you will need a WebDriver for those browsers. Drivers for both Chrome and IE are bundled with the grid package. In order to invoke them, use the command line option `-Dwebdriver.chrome.driver=<path-to-chromedriver>` or `-Dwebdriver.ie.driver=<path-to-iedriver>`. Depending on your OS, the chromedriver to use will be `chromedriver.exe` for windows, `chromedriver_mac` for mac, `chromedriver` for 32bit linux, or `chromedriver_64` for 64bit linux.
- Browsers other than FireFox and Chrome are unsupported until BASIC authentication can be made to work seamlessly across them.

## [4] Use

### [4.1] Grid Options

#### `-hubConfig:`

(hub) a JSON file following grid2 format that defines the hub properties.

#### `-throwOnCapabilityNotPresent:`

(hub) <true | false> default to true. If true, the hub will reject test requests right away if no proxy is currently registered that can host that capability. Set it to false to have the request queued until a node supporting the capability is added to the grid.

#### `-maxSession:`

(node) max number of tests that can run at the same time on the node, independently of the browser used.

#### `-hub:`

(node) <http://localhost:4444/grid/register> : the url that will be used to post the registration request. This option takes precedence over `-hubHost` and `-hubPort` options.

#### `-hubPort:`

(node) <xxxx> : the port listened by a hub the registration request should be sent to. Default to 4444. Option `-hub` takes precedence over this option.

#### `-registerCycle:`

(node) how often in ms the node will try to register itself again. Allow to restart the hub without having to restart the nodes.

**-capabilityMatcher:**  
 (hub) a class implementing the CapabilityMatcher interface.  
 Defaults to  
 org.openqa.grid.internal.utils.DefaultCapabilityMatcher. Specify  
 the logic the hub will follow to define if a request can be  
 assigned to a node. Change this class if you want to have the  
 matching process use regular expression instead of exact match  
 for the version of the browser for instance. All the nodes of a  
 grid instance will use the same matcher, defined by the registry.

**-nodeStatusCheckTimeout:**  
 (node) in ms. Connection and socket timeout which is used for  
 node alive check.

**-timeout:**  
 (node) <XXXX> the timeout in seconds before the hub  
 automatically ends a test that hasn't had any activity in the  
 last X seconds. The browser will be released for another test to  
 use. This typically takes care of the client crashes.

**-port:**  
 (hub & node) <xxxx> : the port the remote/hub will listen on.  
 Default to 4444.

**-hubHost:**  
 (node) <IP | hostname> : the host address of a hub the  
 registration request should be sent to. Default to localhost.  
 Option -hub takes precedence over this option.

**-newSessionWaitTimeout:**  
 (hub) <XXXX>. Default to no timeout ( -1 ) the time in ms after  
 which a new test waiting for a node to become available will time  
 out. When that happens, the test will throw an exception before  
 starting a browser.

**-nodePolling:**  
 (node) in ms. Interval between alive checks of node how often the  
 hub checks if the node is still alive.

**-host:**  
 (hub & node) <IP | hostname> : usually not needed and determined  
 automatically. For exotic network configuration, network with  
 VPN, specifying the host might be necessary.

**-downPollingLimit:**  
 (node) node is marked as down after downPollingLimit alive

checks.

-unregisterIfStillDownAfter:

(node) in ms. If the node remains down for more than  
unregisterIfStillDownAfter millisec, it will disappear from the  
hub.Default is 1min.

-cleanupCycle:

(node) <XXXX> in ms. How often a proxy will check for timed out  
thread.

-nodeConfig:

(node) a JSON file following grid2 format that defines the node  
properties.

-prioritizer:

(hub) a class implementing the Prioritizer interface. Default to  
null ( no priority = FIFO ).Specify a custom prioritizer if you  
need the grid to process the tests from the CI, or the IE tests  
first for instance.

-servlets:

(hub & node) <com.mycompany.MyServlet,com.mycompany.MyServlet2>  
to register a new servlet on the hub/node. The servlet will  
accessible under the path /grid/admin/MyServlet  
/grid/admin/MyServlet2

-proxy:

(node) the class that will be used to represent the node. By  
default org.openqa.grid.selenium.proxy.DefaultRemoteProxy.

-browserTimeout:

(hub/node) The timeout in seconds a browser can hang

-grid1Yml:

(hub) a YML file following grid1 format.

-role:

<hub|node> (default is no grid, just run an RC/webdriver server).  
When launching a node, the parameters will be forwarded to the  
server on the node, so you can use something like -role node  
-trustAllSSLCertificates. In that case, the SeleniumServer will  
be launch with the trustallSSLCertificates option.



## [4.2] Running Tests

The code for the test cases resides in `/fuzzy-shame/tests/src/tests/java/`. Each test can be a member of any number of 'groups', or 'suites', and can receive parameters at runtime that need not be hardcoded into the test case itself.

When running tests through `mvn test`, it is possible to specify parameters to change the test behavior. For example, `mvn test -DsuiteFiles=suite.xml -DtargetIP=192.168.114.117`

The full list of parameters is:

### `suiteFiles`

The xml file defining, containing, or pointing to the tests, or suites to be run.

The default is `"testng.xml"`

### `hubIP`

The IP where the Selenium Grid hub is located.

The default is `"192.168.114.106:4444"`

### `targetIP`

A comma separated list of IP addresses against which to run the test(s). presumably these will be the IP addresses of Pearls, Grids, etc..

The default is `"192.168.114.23"`

### `targetBrowser`

A comma separated list of browsers against which to run the test(s).

The default is `"firefox"`

### `targetOS`

A comma separated list of operating systems against which to run the test(s)

The default is `"linux"`

### `targetUser`

The username to log in to the devices at targetIP.

The default is `"admin"`

### `targetPass`

The password to log in to the devices at targetIP.

The default is `""`

It is important to note that the test(s) will be run on every combination of the targetIP and browser parameters. Thus, to run tests on a target in firefox, chrome and IE, you should execute:

`mvn test -DtargetIP=192.168.114.117 -Dbrowser=firefox,chrome,internetexplorer`  
rather than:

```
mvn test -DtargetIP=192.168.114.117,192.168.114.117,192.168.114.117
-Dbrowser=firefox,chrome,internetexplorer
```

Test reports will be stored in a timestamped subdirectory of `tests/test-output`

### [4.3] Writing Tests

The Java source files containing the automated tests are stored in `/fuzy-shame/tests/src/test/java/com/epiphan/qa/tests` and are members of the package `com.epiphan.qa.tests`

The package `com.epiphan.qa.tests.util` provides some basic helper classes for helping to run tests, and dealing with the basic Selenium initialization common to all of the tests.

For consistency, each test class should be named based on that tests ID in JIRA. For example, a class implementing the test case MD-3993, would be found in `MD3993.java`

Each method which is to be executed as a test, must be marked with the `@Test` annotation. Further, each test method must take one parameter of type `TestEnvironment` and specify the data provider `env`.

Each test method would thus appear like this:

```
@Test(dataProviderClass=DataProviders.class,dataProvider=env)
public void doTest (TestEnvironment e) {
    ...
}
```

The reason that the test methods are required to be this way is to ensure that each one is called exactly once for each OS/browser “environment” specified when the tests are run.

The `TestEnvironment` class is meant to store important information about the current test. It also provides a function to automatically create a `WebDriver` with appropriate settings. To make use of this in your test methods, do something like this:

```
@Test(dataProviderClass=DataProviders.class,dataProvider=env)
public void doTest (TestEnvironment e) {
    WebDriver driver = e.startDriver();
    ...
}
```

Any messages which you wish to appear in the test reports can be logged using the logging functionality built into TestNG’s Reporter class. For example:

```
import org.testng.Reporter;
...
String mesg = "message";
Reporter.log(mesg);
...
```

Any questions you may have about Selenium itself may be answered in the Selenium documentation. Do not count on this. Selenium is very poorly documented. For questions not answered there, check the w3c WebDriver spec.

See `/fuzzy-shame/tests/src/test/java/com/epiphan/qa/tests/MD3993.java` for an example test case based on the MD-3993 test in JIRA.

When defining test suites, please create a new suite xml file for each suite. The name of the file should follow the form: `suitename.xml`. For example, a suite file defining a set of high priority wui tests for 3.15.1 might be named `3-15-1-high-priority.xml`

Ensure that the default suite file, `testng.xml` includes an entry to the suite file you create. Do not edit `testng.xml` beyond this. This will ensure that the default `mvn test` command will run all the test suites.