

|  |
| --- |
| Fusion Test AutomatioN Framework |

**Fusion – Test Automation Framework**

**Submitted by**

**Cybage Software Pvt. Ltd.**

Cybage Towers

Survey No 13A/1+2+3/1,

Vadgaon Sheri, Pune 411014

Phone: 91-20-66041700, Fax: 91-20-66041701

Email: [biz@cybage.com](mailto:biz@cybage.com), [info@cybage.com](mailto:info@cybage.com)

Website: [www.cybage.com](http://www.cybage.com)

# Table of Contents

[1 Introduction – Whats New 4](#_Toc531338514)

[2 What is in this Guide? 6](#_Toc531338515)

[3 Intented Audience 7](#_Toc531338516)

[4 Fusion Environment Configuration 8](#_Toc531338517)

[5 Getting Started 24](#_Toc531338518)

[6 Adding a Test Script 49](#_Toc531338519)

[7 Test Execution 70](#_Toc531338520)

[8 Plugin Library 92](#_Toc531338521)

[10 Jenkins CI 101](#_Toc531338522)

[11 Add-ONS 107](#_Toc531338523)

[12 Limitations 108](#_Toc531338524)

[13 Known issues 109](#_Toc531338525)

[14 Glossary of Terms 109](#_Toc531338526)

[15 FAQs 109](#_Toc531338527)

[16 Contact Us 110](#_Toc531338528)

# 

# Introduction – Whats New

In todays Agile world, automation testing has to cut across the various layers right from database backend data access tests to API / Web services tests to UI(Desktop and Mobile) tests.

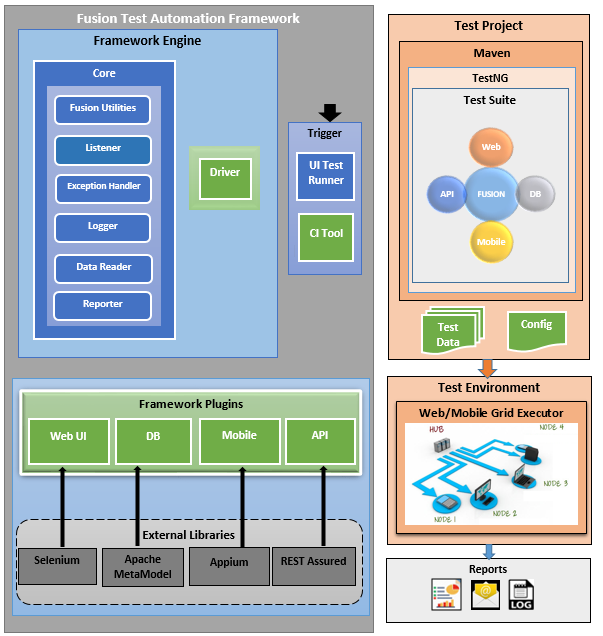
The usual approach teams use here is to create dedicated test automation frameworks that are focused on single purpose. Along with the usual benefits, this though comes with some significant issues, namely lack of integration across the layers which prevent in building end to end test cases, and redundancy of code.

The core USP of Fusion is that it not just allows silo based testing, where in you could have only a set of UI test scripts, API test scripts or database test scripts, but also fuse together instructions from each of these areas in a single script. This allows you to build extremely powerful scripts that can handle almost any tasks spanning across all the layers of your application, be it the front end, middle ware or the database layer.

**Fusion Major Features:**

* Facility to create integrated multi-layered test scripts [ UI / Mobile / API / DB / Fusion ]
* Support on different platforms Linux(Ubuntu 16.04), MacOS and Windows
* Inbuilt test environment setup for remote parallel execution
* Support parallel execution across multiple layer simultaneously
* Integration with CI tools like Jenkins
* Support for GraphQL APIs
* Headless support to run Web tests without User Interface
* User Interface to invoke test execution as per user choice
* Excellent reporting with in-built logging
* Email-able report after execution
* Automatic screenshot capture with detailed error log
* Inbuilt supportive libraries

**High Level Framework Diagram**

****

# What is in this Guide?

This guide provides an overview of the Fusion Test Automation Framework, how the framework is configured, how the test scripts are created and used, and covers all aspects of the reporting module.

# Intented Audience

This guide is primarily intended for Automation QA team who are involved in automation testing.

# Fusion Environment Configuration

## **System Requirements**

**Essential Hardware**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **OS** | **Processor** | **Memory (RAM)** | **Hard Disk Space** |
| **Server (Windows)** | Windows 7 and above | x64 Processor | 8GB (Min) | Minimum : C drive with 40 GB free space required |
| **Server (Linux)** | Ubuntu 16.04, 18.04 | 6400 CPU | 8GB (Min) | Minimum 40 GB free space |
| **Server (MAC)** | macOS Sierra 10.12.6 | [x86-64](https://en.wikipedia.org/wiki/X86-64) | 8GB (Min) | Minimum 40 GB free space |

**Essential Software**

|  |  |  |
| --- | --- | --- |
| Windows | Linux | MAC |
| Microsoft Windows 7,8,10,12 | Ubuntu 16.04, 18.04 | Recommended macOS Sierra 10.12.6 and above |
| Oracle JDK 8/ JRE 8 | Oracle JDK 8 (1.8.0\_121) | Oracle JDK 8 (1.8.0\_121) |
| Maven 3.6 and above | Maven 3.6 and above | Maven 3.6 and above |
| Recommended Eclipse IDE 2019-03 for Java Developers: (Install TestNG and Maven Plugins) | Recommended Eclipse IDE 2019-03 for Java Developers: (Install TestNG and Maven Plugins) | eclipse-jee-2018-12: (Install TestNG and Maven Plugins) |
| If using Mobile Plugin: Recommended Appium version 1.8.1 | If using Mobile Plugin: Recommended Appium version 1.8.1 | If using Mobile Plugin: Recommended Appium version 1.8.1 |
| Supported Browsers: Chrome, Firefox, IE | Supported Browsers: Chrome, Firefox | Supported Browsers: Chrome, Firefox, Safari |

**Optional Software:**

|  |  |  |
| --- | --- | --- |
| Windows | Linux | MAC |
| Optional - MongoDB for anlaytics | Optional - Mongo db for analytics | Optional - Mongo db for analytics |
| Supported Databases:MySQL, MsSQL, MongoDB, H2DB, CouchDB, PostGresDB | Supported Databases:MySQL, MsSQL, MongoDB, H2DB, CouchDB, PostGresDB | Supported Databases:MySQL, MsSQL, MongoDB, H2DB, CouchDB, PostGresDB |

**Web tests can also be run on remote machines through Selenium Grid (Hub and Node).**

**Database/API tests runs on server machine iself.**

**Essential Software & Configurations for Windows machine**

* 1. Microsoft Windows 7 or later edition
  2. Install java version 1.8.0\_121 or above subversions on the machine
  3. Add below configurations in "Environment Variables" on your machine
  4. JAVA\_HOME : C:\Program Files\Java\jdk1.8.0\_121
  5. Path:C:\ProgramData\Oracle\Java\javapath; C:\Program Files\Java\jdk1.8.0\_121\bin
  6. Firefox, Internet Explorer and Chrome (Fusion supports all browser versions which selenium 3.0 supports.)(Note:Ensure you have corresponding supporting drivers present under setupfiles folder of Fusion Framework.

**Essential Software & Configurations for Ubuntu 16.04 machine**

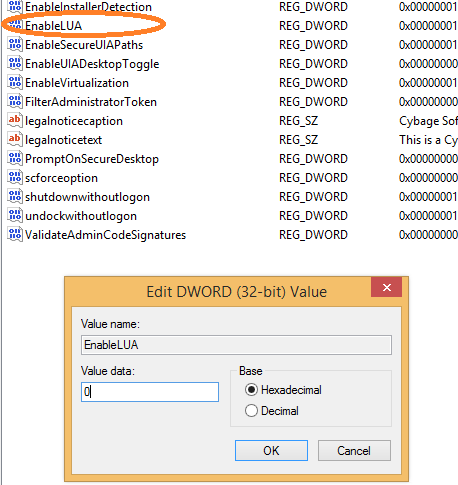
* 1. Install oracle java on your Ubuntu machine and configure JAVA\_HOME and Path in your environment file.
  2. Install and configure Open ssh. You can refer some guidelines at link <https://help.ubuntu.com/community/SSH/OpenSSH/Configuring>
  3. Firefox, Internet Explorer and Chrome (Fusion supports all browser versions which selenium 3.0 supports.)(Note:Ensure you have corresponding supporting drivers present under setupfiles folder of Fusion Framework.
  4. Install and setup maven on your Ubuntu machine and configure MAVEN\_HOME under your environment file. You can refer steps <https://www.vultr.com/docs/how-to-install-apache-maven-on-ubuntu-16-04>.
  5. How to set environment variables on Ubuntu : <https://askubuntu.com/questions/866161/setting-path-variable-in-etc-environment-vs-profile>
  6. Install Xvfb on Ubuntu 16.04 ,you can refer steps <https://www.devmanuals.net/install/ubuntu/ubuntu-16-04-LTS-Xenial-Xerus/how-to-install-xvfb.html>.
  7. For running chrome and firefox with existing hub feature , ensure you have the drivers of both the browsers under usr/bin of Hub Linux machine, in order to carry out execution.

##### **Administrator User**

1. You need to add your user account in Administrator Group
2. You need to create a service account for your project e.g. **autouser1** with admin rights.

**Following steps need to be done for Hub/Node setup on Server and client machines**

* On the machine go to registry using regedit
* Go to HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\policies\system
* Double click on EnableLUA
* Update the value data from 1 to 0



1. If your Fusion server and UI TestRunner is on **single machine**:
   * Then use your **own admin credentials** in “ExecutionUIConfig” refer to section Execute a Test from Fusion Test Runner
2. If **fusion server is different machine** **from user machine:**
   * Then add those connecting users as administrators under fusion server machine.
   * Add the service account e.g. autouser1 as administrator under Fusion server.
   * Include creditials of service account for your project e.g. **autouser1** in ExecutionUIconfig.
3. You need to create a SMTP on whose behalf email notifications will be send after test execution.

##### **Jenkins**

**Essential Software:**

|  |  |  |
| --- | --- | --- |
|  | **Windows** | **Linux** |
| **OS** | Windows 7 or 8 or above | Ubuntu 16.04 |
| JDK | Java 8 and Above | Oracle Java |
| RAM | Min 8 GB | Min 8 GB |
| Local Admin Rights | Yes | Yes |

**References for Linux prerequite Installation:**

* Installing Jenkins on Ubuntu: <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-16-04>
* Additional settings to open ports : <https://askubuntu.com/questions/8184/firewall-settings-for-personal-file-sharing>
* Limiting Logs size: <https://qamag.net/stop-that-huge-jenkins-log/>

## Installation and Registration

##### **Pre-requisites steps before using Testrunner on Linux**

1. If your workspace of eclipse on Linux machine is **"/home/<UserName>/FusionQA3.0"** then open terminal on Linux machine and go till the folder **"/home/<UserName>/FusionQA3.0/AutomationTestProject"**.
2. Execute the bash file “fusion-setup.sh” using command “**bash fusion-setup.sh”**
3. To run from **TestRunner**, Open the **"ExecutionUIConfig.ini"** file and make below changes based in your respective username and workspace folder name:

* SourceRepositoryLocalMappingPath=\\172.27.XX.XXX\FusionQA3.0\AutomationTestProject\
* SourceRepositoryPhysicalPath=/home/cybage /FusionQA3.0/AutomationTestProject/
* TestServerUsername= username
* TestServerPassword=password

1. Open the latest version of "FusionTestRunner\_v3.X.jar". Enter the required details in "Execution Parameters" and in your respective test type tab. Select your tests for respective test type and click on "Run" button to execute your tests.

##### **Pre-requisites steps before using Testrunner on Windows**

1. If your workspace of eclipse on Windos machine is for example **"D:\Workspace\FusionQA3.0\AutomationTestProject"** then go till the folder **""D:\Workspace\FusionQA3.0"**.

Right click on "FusionQA3.0" and share the folder with read and write access to everyone.

1. To run from **TestRunner**, Open the **"ExecutionUIConfig.ini"** file and make below changes based in your respective folder name:

* SourceRepositoryLocalMappingPath=\\172.27.XX.XXX\FusionQA3.0\AutomationTestProject\
* SourceRepositoryPhysicalPath=D:\Workspace\FusionQA3.0\AutomationTestProject
* TestServerUsername= cybage\username
* TestServerPassword=password

1. Open the latest version of "FusionTestRunner\_v3.X.jar". Enter the required details in "Execution Parameters" and in your respective test type tab. Select your tests for respective test type and click on "Run" button to execute your tests.

##### **Fusion TestRunner With Source Control Management (SCM)**

1. FusionTestRunner will be present under AutomationTestProject “TestRunner” folder.
2. Open the "ExecutionUIConfig.ini" file and update below settings as per your project configurations.

**→ TFS Source Control**

TFS source control support through Test Runner is currently only present for windows .



**→ TFS With Git Source Control**

****

**→ BitBucket With Git Source Control**

****

**→ GitHub Source Control** ****

Sample: "ExecutionUIConfig.ini" files

|  |
| --- |
| [AutomationRepository]  SourceRepositoryType=tfs  SourceRepositoryURL=http://tfs:8080/tfs/collectionName  SourceRepositoryUsername=cybage\autoservice  SourceRepositoryPassword=\*\*\*\*\*\*  SourceRepositoryServerMappingPath=$/TCOE/Other/Fusion Test Runner/NKSampleCode  SourceRepositoryLocalMappingPath=\\<<machineIP>>\TFSSourceCode\NKSampleCodeLocal*.(Note:Shared Path where you’re actual AutomationTestProject is present on local machine if you are using Test runner from windows machine. If you are using Fusion TestRunner on Linux machine then path will be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject) (Note :In order to access the project you need to share the root folder of your project with users who are going to access it on Windows machine)*  SourceRepositoryPhysicalPath=D:\Fusion\_Restructure\_v3.1\AutomationTestProject. *(Note: Actual Path where your AutomationTestProject is present on local Windows machine. Or if you are on Linux machine then this path would be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject)*  TestServerIP=172.27.56.xxx.  TestServerUsername=cybage\username(For Linux machine ,do not specify the domain)  TestServerPassword=\*\*\*\*\*\*\*  JenkinsServerIP=172.27.56.xxx.  JenkinsServerUsername=cybage\username(For Linux machine ,do not specify the domain)  JenkinsServerPassword=\*\*\*\*\*\*\*  JenkinsWorkspacePath=\\<<machineIP>>[\\Jenkins\workspace\Jenkinsproject](file://Jenkins/workspace/Jenkinsproject).  (Shared Path of your Jenkins workspace to copy AutomationTestProject)  *(Note :In order to copy the project you need to share the root folder of your Jenkins workspace with the users who are going to access it) (If you are using Fusion TestRunner on Linux machine then path will be for e.g. /var/lib/Jenkins/workspace)*  JenkinsWorkspacePhysicalPath=C:\Program Files (x86)\Jenkins\workspace\Jenkinsproject.(Actual Path of your Jenkins workspace to copy AutomationTestProject)  [/AutomationRepository]  [WebURL]  QA= https://www.google.co.uk  Staging=https://www.google.com  Production=https://www.google.co.in  [/WebURL]  [Browsers]  BrowsersToInclude=Firefox,Chrome,ie  [/Browsers]  [OS]  OSToRun=Windows  [/OS]  [MobileURL]  QA= https://www.google.co.uk  Testing=http://www.google.com  Cloud=http://www.google.com/in  [/MobileURL]  [MobileDevice]  MobileDeviceTypes=Android,iOS,Windows  AndroidDeviceOS=4.1,4.2  iPhoneDeviceOS=5S,7  [/MobileDevice]  [API]  APIEnvironmentToRun= Testing1,Testing2,Testing3  [/API] |

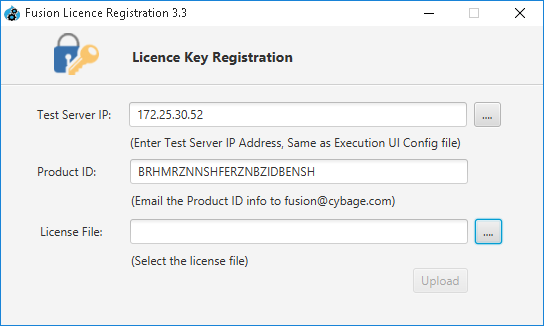
##### **Fusion TestRunner WithOUT SCM**

1. FusionTestRunner will be present under AutomationTestProject “TestRunner”folder.
2. Open the "ExecutionUIConfig.ini" file and update below settings as per your project configurations.

|  |
| --- |
| [**AutomationRepository**]  SourceRepositoryType=  SourceRepositoryURL=  SourceRepositoryUsername=  SourceRepositoryPassword=  SourceRepositoryServerMappingPath=  SourceRepositoryLocalMappingPath=\\<<machineIP>>\TFSSourceCode\NKSampleCodeLocal*.(Note:Shared Path where you’re actual AutomationTestProject is present on local machine if you are using Test runner from windows machine. If you are using Fusion TestRunner on Linux machine then path will be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject) (Note :In order to access the project you need to share the root folder of your project with users who are going to access it on Windows machine)*  SourceRepositoryPhysicalPath=D:\Fusion\_Restructure\_v3.1\AutomationTestProject. *(Note: Actual Path where your AutomationTestProject is present on local Windows machine. Or if you are on Linux machine then this path would be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject)*  TestServerIP=172.27.56.xxx  TestServerUsername=cybage\username(For Linux machine ,do not specify the domain)  TestServerPassword=\*\*\*\*\*\*\*  JenkinsServerIP=172.27.56.xxx  JenkinsServerUsername=cybage\username(For Linux machine ,do not specify the domain)  JenkinsServerPassword=\*\*\*\*\*\*\*  JenkinsWorkspacePath=\\<<machineIP>>[\\Jenkins\workspace\Jenkinsproject](file://Jenkins/workspace/Jenkinsproject).  (Shared Path of your Jenkins workspace to copy AutomationTestProject) *(Note :In order to copy the project you need to share the root folder of your Jenkins workspace with the users who are going to access it) (If you are using Fusion TestRunner on Linux machine then path will be for e.g. /var/lib/Jenkins/workspace)*  JenkinsWorkspacePhysicalPath=C:\Program Files (x86)\Jenkins\workspace\Jenkinsproject.(Actual Path of your Jenkins workspace to copy AutomationTestProject) |

##### **Getting Fusion license KEY**

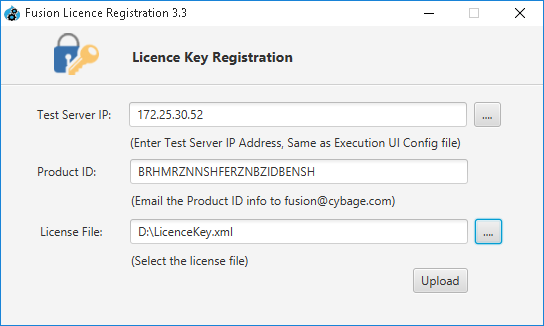
1. Double clicking “**FusionRegistration.jar**” located in “**AutomationTestProject**”
2. It will invoke below screen generating product id after entering ip address of your test server machine.



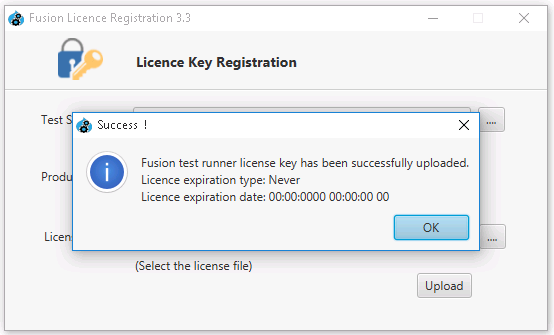
1. **Copy** the **Product ID** and mail it to Automation Team 🡪 **Fusion@cybage.com**
2. Fusion Team will provide you the License key.

##### **configuring Fusion license**

1. Copy license file xml at your local drive under **“AutomationTestProject”** location.
2. Go to the “**AutomationTestProject**” project directory location.
3. Double clicking “**FusionRegistration.jar**” located here. You can see below screen.



1. To upload your license key browse the location where license file is kept (**#5**)
2. Select license file and click upload button.
3. On successful upload, you will get confirmation message depending the type of license procured from coe team.



1. Once license on one machine configured and installed properly then “Framework.exe” file generated by the license registration process.
2. With the new version of Fusion Framework, copy “Framework.exe” file to another machine under the “AutomationTestProject”. User can start using the Fusion automation Framework without any issue. There is no need to generate licenses again for the new machines.
3. Automation COE will provide you with:
   1. Sample Automation Test project.
   2. Generic Framework Library jar
   3. Other Framework Library plugins jars with sources
   4. License as requested.

##### **Source Control Management**

* The Automation Test Project should be added in any of the below source repository
  + TFS
  + TFS with Git
  + Bit Bucket with Git
  + GitHub
* In case you do not have above source control repository then you need to raise a help desk ticket to create a source control for your project.
* Thereafter add the Automation Test Project and define your local mapping path.

**NOTE:** In case, you do not have a source control repo setup, then make following changes in the Test Runner UI "ExecutionUIConfig.ini"

* Remove the TFSSourceRepositoryServerMappingPath value and keep it as blank (empty)
* In addition, add the local directory path of the source code like

TFSSourceRepositoryLocalMappingPath=\\<<machineIP>>\c$\TFSSourceCode\NKSampleCodeLocal

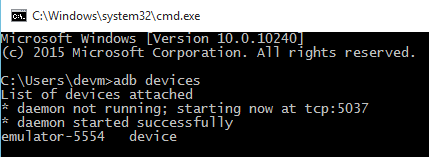
##### **Mobile Installation**

To create and execute mobile specific tests below are some pre-requisites,

* Ensure that USB ports are enabled and device specific drivers are installed.
* Android SDK installed and configured (Refer “Android SDK setup steps”).
* Appium installed and configured (Refer “Appium setup steps”).

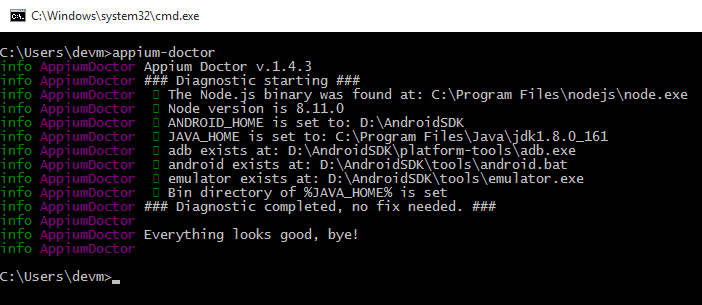
##### **Android SDK setup steps:**

1. Follow <http://www.androiddocs.com/sdk/installing/index.html> steps to install and configure Android SDK (Using Android Studio is recommended to install Android SDK).
2. Configure ANDROID\_HOME (refer <https://developer.android.com/studio/command-line/variables> ) as environment variables (If you don’t know how to add environment variables, you can refer <https://www.java.com/en/download/help/path.xml> as how to do example).
3. Add “*%ANDRIOD\_HOME%\platform-tools; %ANDRIOD\_HOME%\build-tools; %ANDRIOD\_HOME%\ tools*” into PATH.
4. To vefiry Android SDK setup, connect android device to machine (if required install device specific drivers) and run “adb devices” on command prompt or terminal. You should be able to see connected devices as below –



##### **Appium setup steps:**

1. Follow <http://appium.io/docs/en/about-appium/getting-started/#installation-via-npm> steps to install Appium using npm. Appium version 1.8.1 is recommended to use with Fusion, below are commands to refer for installing specific Appium version,
   1. To Install “ npm install -g appium@1.8.1” [one time setup]
   2. To execute “appium”
2. To verify installation use “appium-doctor” as mentioned below,
   1. Install using command “npm install -g appium-doctor” [one time setup]
   2. Execute using command “appium-doctor”, it will run some diagnostic tests and let you know the system status inorder to run appium tests



##### **Recommended macOS Sierra Version 10.12.6 and abovemac OS specific configurations:**

For connecting devices on Mac machine, configure appium, Android SDK and environment variables on Mac machine as per instructions mentioned in below link

<https://gist.github.com/hemano/351c842705ff6dbcd48ceef2dc88b8a0>

##### **Pre-Requisites for Remote MAC Execution:**

When devices are connected to your remote mac machine then before you start execution of tests on those devices, perform below steps

1. Share your windows “Fusion framework local workspace”, with mac machine.
2. Then from Mac machine access your windows workspace path.
3. Go to folder “remote-mac-devicejar” from mac machine then open terminal on mac and start appium server with command java –jar appium\_Mac\_RunnableJar-0.0.1.jar .Check whether mobile.ini file is created with device details under iniDir folder “AutomationTestProject\remote-mac-devicejar\iniDir”.
4. After that start execution of Web application on remote mac through TestRunner or Kickoff.

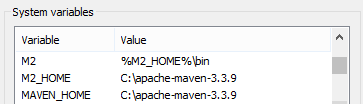
**Note:** Mobile plugin supports on Web application scripting and execution on devices connected to remote mac.

# Getting Started

## Configuring FUSION from Scratch

1. Make sure Maven is configured properly.

Add below entries in Environment System variables.



1. Make Sure JDK is properly installed and configured on the system.
2. Unzip and copy all the contents of AutomationTestProject.zip file at your local drive and place it in your workspace
3. Open Eclipse
4. Right Mouse Click (RMC) in project explorer > Click Import
5. Expand General & select Existing Project to Workspace
6. Click Next
7. Select “Select Archieve Files”
8. Click Browse to the location in **#3**.
9. Select “**AutomationTestProject**”
10. Click **Finish** button.
11. Update Build path for JDk version if needed.
12. Then follow below steps, right click “**AutomationTestProject**” Run As

a.       Maven clean

c.       clean install –Dmaven.test.skip=true

d.       update Maven project

14. Copy license file to folder “**AutomationTestProject**”

15. Register license using license registration utility provided in the package.

## **UPGRADING FUSION to NEWER Version (Only applicable for the project using older version)**

**Note:** Before migrating all the team members to new Fusion version ,ensure the team should check-in their previous work in source control repository , and then only lead should take all the latest check-in from source control repository in existing workspace.

In order to upgrade to new version

1. create a new eclipse workspace for fusion version3.X and
2. Import the new AutomationTestProject provided by fusion team in new workpace.
3. Then proceed with upgrade follow below **Step by Step Guide to Upgrade**.
4. After migrating all the previous artifacts to new workspace , “AutomationTestProject” needs to be check-in in souce control repository under new repository localtion.
5. All the other team members will then pull the New upgraded Fusion AutomationTestProject in new workspace to proceed with further scripting.

**Step by Step Guide to Upgrade**:

Identify changes you have done in your Fusion framework of older versions . It may involve below mentioned changes

1. Newly created test cases for plugins (Web,Database,API or Mobile and Fusion).
2. Extended methods created for respective plugins.
3. Changes in configuration methods in “DriverBase” if any.
4. And any other new functionality added.

**Test case level changes will be as follows for respective plugins**

1. **Web tests migration:** 
   1. Page Object classes from com.web.pages,
   2. All Tests from com.web.tests
   3. Include initialisation of web pages under new workspace UIPageWrapper.
   4. Under test cases and pages change the “import com.cybage.webframework.IUIDriver;” to import com.fusion.plugin.extension.web.IUIDriver;
2. **DB tests migration:** 
   1. All tests from package com.database.tests to new fusionversion3.X package com.database.tests.
   2. For File system test cases below mentioned changes need to done
      1. “FileDriver” need to be searched and replaced by “*dbDriver”.*
   3. Need to add argument “*ReportLogger”*  to constructor new FusionDataSetTableModel in all the test cases

For DB test cases below mentioned changes need to done

* 1. add argument “*ReportLogger”*  to constructor new FusionDataSetTableModel in all the test cases wherever applicable

1. **API tests migration:**  Copy all the api test cases from previous Fusion version to new fusion version package com.api.tests.No changes need to be done ,just create new classes or copy existing test classes.
2. **Mobile Tests migration:** Copy all the api test cases from previous Fusion version to new fusion version package com.mobile.tests. In before method signature of mobile driver needs to be changed.And mobile driver and wrapper initialization in each test cases.
3. **Fusion Tests migration:** Copy all the api test cases from previous Fusion version com.fusion to new fusion version package com.fusion.tests. The package name should be changed from com.fusion to com.fusion.tests.And all the changes mentioned above for respective plugins are applicable

All the extended methods of different plugins need to be added to respective packages under AutomationTestProject.

**Note :** In fusion Version 1.0 extended classes were present under respective plugin source codes projects . For example API ,Web, Database, mobile had different projects with Name: CybageAPILibrary , CybageWebLibrary, CybageDBLibrary, CybageMobileLibrary respectively. But in New version 3.X instead of different libraries we have extended methods for each library under “AutomationTestProject” itself and source jars for all the libraries are present under “fusionLibs” folder of new “AutomationTestProject”.So user needs to copy all the extended methods from above libraries to below mentioned packages extended classes of “AutomationTestProject” under Fusion version 3.X.

**API specific extended methods need to be added under below package from CybageAPILibrary**

1. com.fusion.plugin.extension.api

**Database specific extended methods need to be added under below package from CybageDBLibrary**

1. com.fusion.plugin.extension.db

**Mobile specific extended methods need to be added under below package from CybageMobileLibrary**

1. com.fusion.plugin.extension.mobile

**Web specific extended methods need to be added under below package from CybageWebLibrary**

1. com.fusion.plugin.extension.web

If any changes are done under DriverBase then below package of version3.X needs to be updated accordingly from previous version to this com.frameworkbase.

After all the abe mentioned changes are done, checkin the entire “AutomationTestProject” to source control repository in different branch and let other team members to use this upgraded project in new workspace for Automation.

Other changes will be for Web , if any browser are of lower versions then need to be upgraded to versions supported by Fusion version3.X.

## How to get Started

**Pre-requisites:** Ensure all the required installations are done.

1. Open the Test Automation Project
2. It will contain a sample Maven project structure
   1. Src/main/java – this will have Kickoff.java

The class is responsible for Invoking the execution

* 1. Src/main/resources – this will have log4j.properties file
  2. Src/main/java –this will contain extension packages for each different test types.
* com.fusion.plugin.extension.api
  + - * + Here you can add extended methods specific to api plugin ,for details refer section API plugin extension
* com.fusion.plugin.extension.db
  + - * + Here you can add extended methods specific to database plugin,for details refer section DB plugin Extensibility
* com.fusion.plugin.extension.mobile
  + - * + Here you can add extended methods specific to mobile plugin,for details refer section Mobile plugin extension
* com.fusion.plugin.extension.web
  + - * + Here you can add extended methods specific to Web plugin,for details refer section WebUI plugin extension
  1. Src/ test/java – this will contain various packages for each differernt test types.
* com.web.tests
* com.web.pages
  + - * + Here you need to add pageobjects for your web pages
* com.api.tests
  + - * + Here you need to add **API** Test Scripts
* com.database.tests
  + - * + Here you need to add **DB** Test Scripts
* com.fusion.tests
  + - * + Here you need to add Fusion test script that comprises of more than one test type **e.g. Web and DB**

**Note: Refer to section 6 Adding a Test Script for adding respective tests**

* 1. com.frameworkbase – this will contain DriverBase.java class.

It will be used to initialize the required drivers.

**Note: Refer to section 5.4.3Driverbase for details**

* 1. src/test/resources – this will contain the chrome driver exe
  2. Maven Dependencies – This will contain all the Maven dependencies required by the project which will be stored in your .m2/repository folder.

e.g. C:\Users\username\.m2\repository

* 1. CommonFrameworkTest\_<ExecutionRunName>>– this will contain following folders:
* **Configuration:**

This will contain 2 files

* + - 1. **appConfig.ini** – Here you need to add all the Connection Types and Data Sources information for Database and API

**Note: Refer to section 6.3.1(DB) and 6.4.2(API) for test configuration details**

* + - 1. executionRunConfig.ini – This will be autogenerated based on the configuration set through Fusion Test Runner
* **Log:**

It will contain Log files for each test type Web, Database, API, Mobile and Fusion.

**Note: Refer to section 5.4.11 for details**

* **Report:**

This will contain the Test Report for the latest execution run.

* **TestData:**

This will contain the test data for the respective test types. Here you need to add/update the test data as per the project needs.

**Note: Refer to section 5.4.5 Using dataprovider for details**

* **testNGfiles:**

This will contain the testing files that will be generated based on the testtype mentioned in the executionRunConfig.ini

It will be used for the execution of the tests.

## Framework Components

##### Framework Component Details

Fusion as a framework comprises of multiple components. The source code of the **plugin libraries** and the **sample test scripting project** will be provided to enable projects to extend/rewrite the functionality of the methods. Below table depicts the major components and the source code sharing status for them.

|  |  |  |
| --- | --- | --- |
| **Fusion Components** | **Source Code To Be Shared?** | **Details** |
| Web Plugin Library | Yes | This plugin includes ready to use Web library. Source code can be extended and new functionality can be added. This extension is available under “AutomationTestProject”. The source code of library is provided as source jars under “fusionLibs” folder present under “AutomationTestProject” |
| API Plugin Library | Yes | This plugin includes ready to use API library. Source code can be extended and new functionality can be added. This extension is available under “AutomationTestProject”. The source code of library is provided as source jars under “fusionLibs” folder present under “AutomationTestProject” |
| DB Plugin Library | Yes | This plugin includes ready to use DB library. Source code can be extended and new functionality can be added. This extension is available under “AutomationTestProject”. The source code of library is provided as source jars under “fusionLibs” folder present under “AutomationTestProject” |
| Mobile Plugin Library | Yes | This plugin includes ready to use Mobile library. Source code can be extended and new functionality can be added. This extension is available under “AutomationTestProject”. The source code of library is provided as source jars under “fusionLibs” folder present under “AutomationTestProject” |
| Automation Test Project | Yes | This is a Maven Automation test project where test scripts and project specific libraries can be added. Source code can be extended and new functionality can be added using extensibility option.  This project has included reference jars for all plugin libs along with plugin source libraries. |
| Fusion Core Framework | No | It contains the framework’s execution core. No source code will be shared, and typically no changes are required to be done here during day to day execution. |
| Fusion UI Test Runner | No | This is thick UI client for executing Fusion test scripts. . No source code will be shared, and typically no changes are required to be donein this, during day to day execution. |

##### Framework Do’s and Don’t

Do’s and Don’t while using Fusion:

**Dos**

1. It is recommended to use single domain user having administrative and registry rights on all the machines used as an automation test environment.
2. Follow the folder structure guidelines while creating test scripts.
3. Follow the test script standards for each plugin for writing.

It is recommended to make use of Sample test as a template for writing the test scripts.

1. Try,Catch Block for @Test methods is **NOT** required as exception handling for test methods are taken care internally. However if you still need to add try/catch blocks for any reason then make sure to log any exceptions in Report using method *ReportLogger*.LogFail(ExceptionType.***Application***, “ExceptionMessage”); otherwise it will not be logged. You can use finally block to add any ‘must execute’ statements where required.
2. Make sure you clear Temp file on periodic basis.
3. Report object(ReportLogger) should be used only within/through @BeforeMethod ,@Test blocks. It can also be used in the immediate first call of @AfterMethod block otherwise if used anywhere else report will fail to log that entry .
4. When Web test cases are run in hub and Node mode ,ensure to delete temporary files for service account of your project e.g. **autouser1,**on hub and nodemachines.
5. If machine on which Fusion is used is a Windows server machine then ensure that temporary files under Windows Temp folder are also deleted on regular basis after your execution.
6. For Linux (Ubuntu) machine do not change permission to 777.

**Don’ts**

1. If you are adding any configuration methods like @BeforeMethod/@BeforeTest then avoid using try/catch blocks as it’s exceptions if arise are internally handled in Fusion framework through listeners ,else report will not reflect proper output. If you are still required to use try/catch blocks for some reason then make sure to throw the same exception from catch block after your handling.
2. It is recommended not to make any change in the plugin methods.

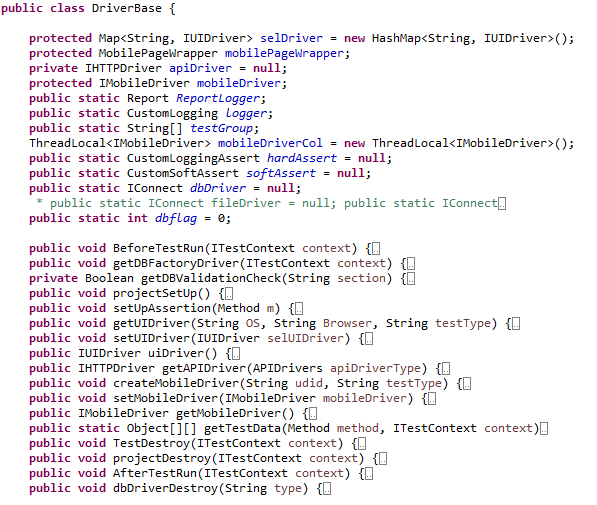
Any customizations required in the plugin can be done by overriding the existing functionality in it’s extended class.

1. It is recommended not to make any changes in DriverBase class and methods unless you need some customization or want to use your extended plugin drivers.

##### **Driverbase**

This class is used to define and initialize drivers such as selDriver, mobileDriver, apiDriver, and dbDriver etc. And it contains required before and after setup methods for project-level, suite-level, test-level and method-level as mentioned below:

* Initializes test report object i.e. extent reports
* Does the required setup before exectuting a database test.
* Initializes custom assertion class
* Reads test data from data driver and saves in object array
* Quits mobile and ui drivers after executing methods as part of AfterMethod
* Quits db and file drivers after test execution as part of AfterTest

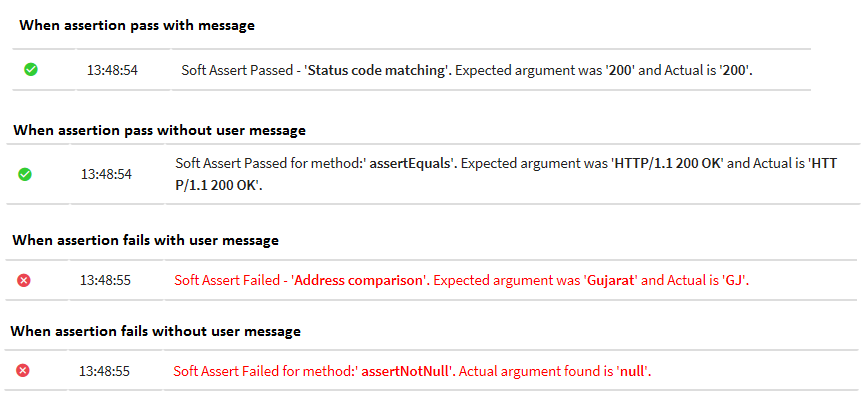


##### **Asserts**

Fusion framework provides two assertion objects –softAssert , hardAssert which are available to use directly within Test Method (@Test). These are instances of class extended from TestNG class SoftAssert and LoggingAssert. These objects provide all assertion methods which TestNG assertion provides. Behavior of these assertion methods are same as that of TestNG assertion. However there are certain customization done to ensure assertion entries are always logged into test logs and Extent reports. For best use of these assertion follow below given rules-

* + Use softAssert and hardAssert objects directly as required in your test method (@Test).
  + For softAssert, use assertAll() at the end of test method or in finally block.

* + 
  + Use assertion method where assertion message can be passed so that report looks more meaningful and consistent. This message will be printed in report and logs whenever corresponding assertion pass or fail. So make sure this message is neutral like ‘Address comparison’/’Verification of status code’. e.g for assertEquals(String actual, String expected), instead use assertEquals(String actual, String expected, String message).
  + Refer sample report entries when assertion pass and fails with and without user message.



##### **USING Data Provider**

Fusion framework provides data driven test approach where all your test data will be required to be maintained separately in Excel (.xlsx) files . Data provider library will look for relevant files in corresponding test type (Web /API/Database/Fusion/ Mobile) folder and will pass required set of data to each test(@Test).

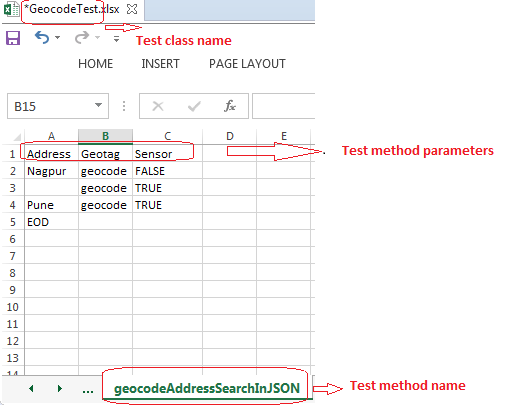
**Mandatory Usage and naming conventions:**

* Do not make any changes to the definition of existing data provider in DriverBase class .
* Data provider currently doesn’t support parallel execution at data iteration level though test can be executed parallel. So make sure ‘parallel’ identifier is not provided or always set to ‘false’ for given DataProvider. (refer code snippet below)



* To provide test data in excel file, provide .xlsx file name same as of test class name and sheet name as test method (@Test) name. Every test parameter in @Test should be provided in excel file sheet as its columns with same header names. (Case is not sensitive, column order also doesn’t matter)





* Header names should always start from first row , and entries should start from first column.
* For non-default sheets ‘EOD’ (End of Data) should be specified at the end of entries in the first column to mark the end of data set. Rest all entries if exists will be ignored. For Default sheets all columns needs to provide ‘EOD’ at the end of required data set.
* Format of all entries including ‘blank’ entries should always be ‘Text’ even if it contains non text values (like Boolean, integer, double etc.). Conversion of these values into respective type is specific to project and needs to be handled at test method level as of now.
* All blank entries in specific test sheet will be considered as empty string values and will be passed accordingly to test method.

**Optional Usage and implementations:**

* If any test parameter is common for all test methods in a test class then all such data columns can be placed in "Default" sheet of specific test data file .
* If any test parameter is common across multiple test class then all such data columns can be placed in "Default" sheet in "Default.xlsx" file (like any login credentials or session token could be placed at one place instead of repeating it at multiple sheets).
* If same parameter column exist in specific test sheet as well as in Default sheet or file then it will be fetched based on internal preference set as given below.
* Preference of fetching any test parameter for a given method among files and sheets is as given below-

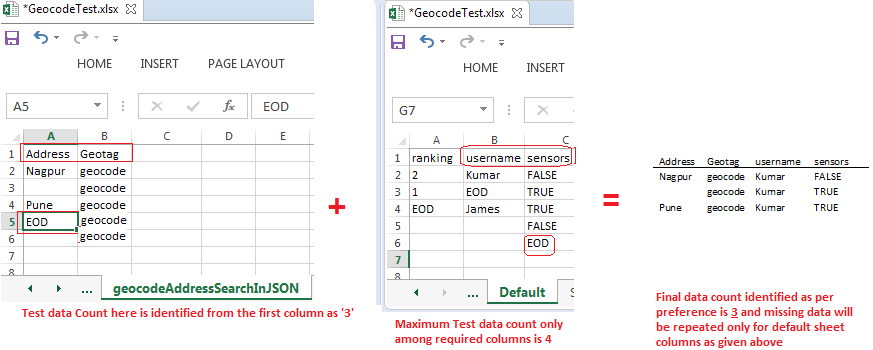
1.Specific test sheet in specific test data file

2.'Default' sheet in specific test data file

3.’Default’ sheet in Default.xlsx file

* If count of entries are different for required columns as per where 'EOD' is defined (e.g when column are divided across specific and default sheets)then first maximum count is identified from the default sheet among required parameters, then similarly max count is derived from specific test sheet among remaining required parameters .Later as per preference of the sheets given above final count is considered. If any parameters doesn't have required data entries then existing entries will be repeated in same order for Default sheets else blank entries will be passed for specific sheets. E.g refer below code and corresponding test data identified.





**Exception Details :**

Data provider throws exception for following scenarios and needs to be handled as given below-

|  |  |
| --- | --- |
| Exception Type : Message | Resolution |
| ReflectiveOperationException : Test method parameter names are not identified | Build the project by considering method parameters e.g For Java build/TestNG execution , check 'store information about method parameters' under Java compiler setting of editor OR For Maven build , add following lines under maven-compiler-plugin configuration "<compilerArgument>-parameters</compilerArgument>” |
| NullPointerException : Test type not identified based on test package | Make sure your target test methods are defined in packages under specific test types (api/web/database/mobile/fusion) like if it’s a fusion test then it’s package name should start with ‘com.fusion.tests’ |
| NullPointerException: Test data file missing including default file | Make sure test data file is uploaded for all required test class having same file name as that of test class. Or Upload Default.xlsx file having common columns with same name as that of test parameters. |
| NullPointerException : Test data sheet <xyz> missing including default sheet | Make sure test data sheet exists in uploaded test data files for all required test methods having same sheet name as that of test method(@Test). Or add Default sheet having common columns with same name as that of test parameters. |
| NullPointerException : Parameter <param> is not found in either sheets | Make sure required test parameter exists in specific sheet with same column name as that of parameter name, check spellings and unwanted space if added. |
| NullPointerException: Test data row count is not identified for test | Make sure required test files are uploaded with corresponding sheets in it with required naming conventions. |
| NullPointerException: Column <columnName > is not found in any given sheet | Make sure required test parameter exists as a column in test data sheet of required test data file with required naming conventions. |
| NullPointerException : Row index '0' doesn't exist for sheet: | Make sure header names of test data starts from first row of the sheet. Don’t leave first row as blank. |
| IllegalArgumentException: Unsupported argument type : Found cell at row index | Make sure data type defined against all entries including blank entries in excel file sheet is of format ‘Text. Even ‘General’ is NOT allowed. |
| NullPointerException : There is no data entry record for the column | Make sure ‘EOD’ is not added as the first entry for any column after header. Minimum one entry is required against any test data column. If you have to pass null or blank entry against parameter then add a blank row entry with format as ‘Text’. |
| NullPointerException : 'Could not identify the data row count as 'EOD' is not defined for the column | Make sure ‘EOD’ is added against the first column in specific test sheet and against all columns if it’s in any ‘Default’ sheet. |

##### **Email**

Fusion send execution report by Email after execution is completed.

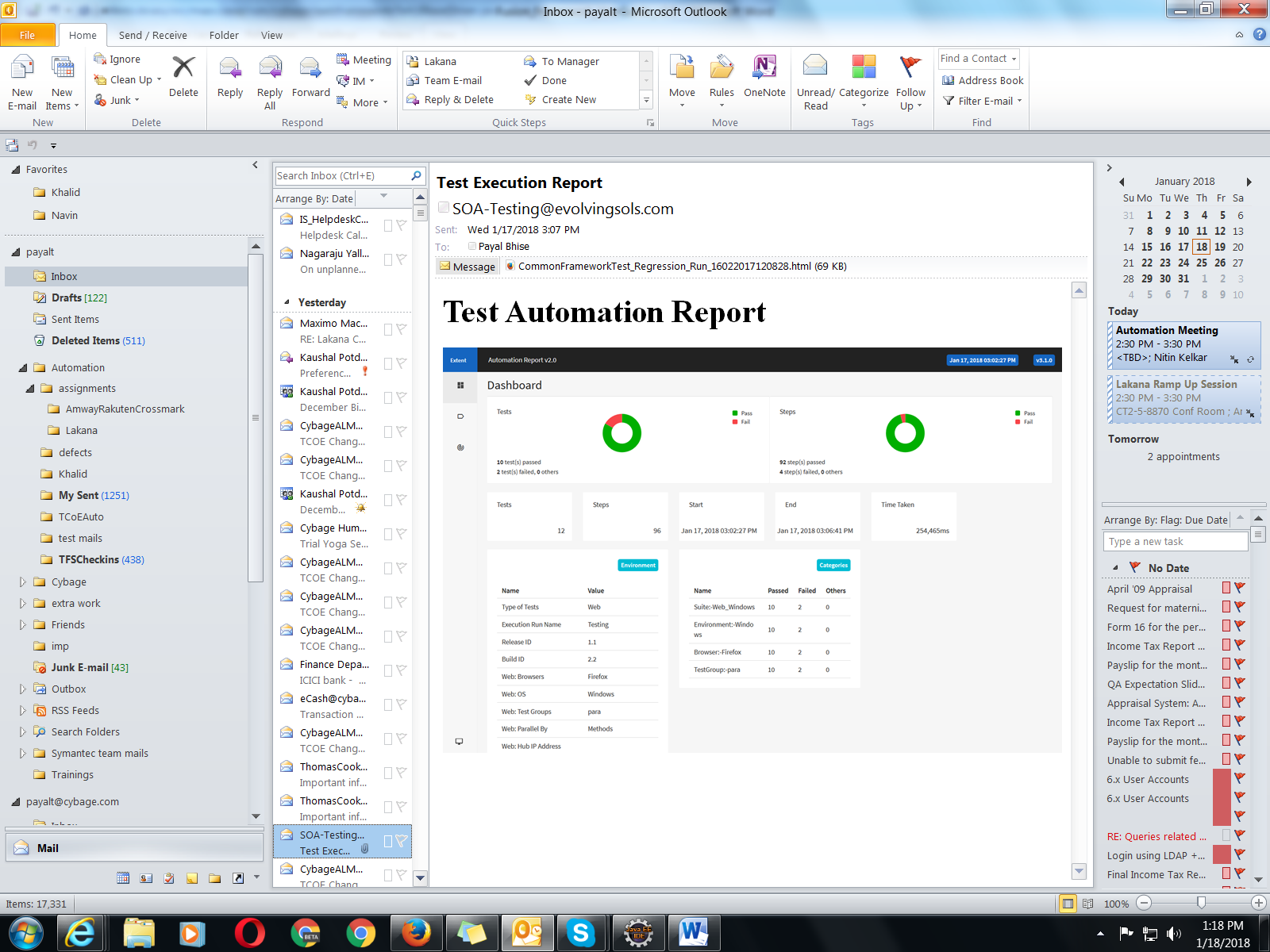
This email has html report as an attachment and an image of execution summary as an email body.

Following configuration is required for sending out the email.

1. Get authorised SMTP user credentials for email sending.
2. Get Email account by which email will be sent.
3. Go to [\\AutomationTestProject\ExternalConfig.ini](file://AutomationTestProject/ExternalConfig.ini) and provide required information there under section [Email].
4. Make sure that server machine has Firefox version 47 or above is installed.

Email will be sent to the email address/distribution list specified on the execution UI.

Sample email screenshot:



**Customization**:

* Email sending functionality is getting trigged from KickOff.java - finally block
* To Modify Report Caption or to display customized text in the email body:

emailBodyContents.append("<h1>Test Automation Report</h1>");

* Instead of default report if you want to send some other customized report as an attachment, String htmlReportLocation holds that location.
* Below method send email using above configutation by attaching the given file as an attachment and with given email body contents.

EmailSending.*composeEmail*(emailBodyContents, htmlReportLocation);

* For any other changes in the email sending behaviour, one can write new method and can call that method in place of EmailSending.*composeEmail*(..).

##### **Listeners**

Fusion framework got an internal listener (TestListener) which is used majorly for Reporting functionality.It helps in listening various TestNG events and log it into Extent Report.

It is NOT supposed to be extended by any means as it will affect Reporting.

##### **KickOff**

**KickOff** is the main program which initiates the execution.

It is responsible for–

1. Setting up the Grid environment
2. Trigger the execution according to Execution UI selected parameters using Maven POM.
3. After execution it brings down the environment and sends out the email with Execution Report.

##### **Web Test Execution Environment**

Web tests can run on standalone machine or on remote machines and in parallel.

Execution on remote machine is handled using Selenium GRID.

Hub-node setup is automatically been done during kickoff process by considering the Hub and node IP addresses provided on the Test Runner UI.

**Supported browsers**:

Web tests can run on follwing browsers with the help of respective browser drivers.

1. Firefox – version 47 and above
2. Chrome – version 53 and above
3. Internet Explorer- version 9,10,11

Tests can be run in headless mode with Firefox and Chrome browser when “Headless” is selected on the TestRunner UI. Heandless mode is not available for Internet explorer.

##### **INI File Operations**

This class exposes API’s related to INI file operations such as reading and writing INI fie, getting value from INI file, etc.

While writing test cases or utility function if there is need to read value from any (Application, Execution, Email or Mobile) of the INI file, you can simply call **getValueFromIniFile API and other API’s,** it will provide you the respective Key value.

String url = IniFileOperations.*getValueFromIniFile*(

IniFileType.***Execution***, testType + "Testing", "MobileBaseURL");

IniFileType can be Execution/Application/Email/Mobile. For other API’s from this class refer Java

IniFileType can be Execution/Application/Email/Mobile. For other API’s from this class refer Java Doc.

##### **Logger**

**Introduction:**

* Fusion **Logger** is implementing **log4j** and generating log files for each test type Web, Database, API, Mobile and Fusion.
* WEBUI.log
* DATABASE.log
* Combined.log
* API.log
* MOBILE.log
* FUSION.log
* This LOG folder will be generated under the “Execution run name” folder created for Fusion Test Runner.
* Log statements will be added in the respective log files under the LOG folder for each type of automation test.
* Above log files will be having log statements present in automation test cases + log statements present in respective plugin libraries for corresponding test type.For e.g. If you are executing automation test cases of **Web** test type then log statements present in test cases of Web + respective plugin methods called by web tests in **CybageWebLibrary** will beadded in **WEBUI.log**.Similar behavior is for DATABASE.log, API.log, MOBILE.log and FUSION.log
* In addition to this, there will be another two log files generated:
  + **FRAMEWORK.log** will be having log statements to common framework operations which are required for each test type but not directly associated with the tests.For e.g. log statements present in DriverBase (BeforeMethod/AfterMethod, BeforeTest/AfterTest, BeforeSuite/AfterSuite) and other common framework operations present in frameworkutility like TestNGOperations, GridOperations, EmailSending, IniFileOperations and Reports will be added in FRAMEWORK.log file.
  + **Combined.log** is the log file which will be combined logs present in all other log files WEBUI.log, DATABASE.log, API.log, MOBILE.log, FUSION.log and FRAMEWORK.log for one test case execution.
* Configurations related to log statements and files is mentioned in **log4j.properties** which is present under the path “\src\main\resources” of the AutomationTestProject.

**How to use Logger in Test:**

* User can add log statement in automation tests or in plugin method of respective test type using below Log method having two parameters.



***For e.g.:*********

* **Change LogType to either** DEBUG/WARN/INFO/ERROR/FATAL based on the need.

**How to Configure Log4J.properties file:**

**Note** – This would be required in case you need to change the default log level

We have defined logger for each test type with its appender as below:

#log4j.rootLogger=ERROR, console

log4j.rootLogger=INFO, Combined

log4j.logger.WEBUI=INFO, WebUI

log4j.logger.API=INFO, Api

log4j.logger.DATABASE=INFO, Database

log4j.logger.MOBILE=INFO, Mobile

log4j.logger.FUSION=INFO, Fusion

log4j.logger.FRAMEWORK=INFO, Framework

**For e.g. In “**log4j.logger.WEBUI=INFO, WebUI”, **WEBUI** is the logger, **WebUI** is the appender and INFO is set as default logging level for test type “Web”.

We have set **default log level as** “INFO” for all test types for one test case execution.

You can change the **default log level** to DEBUG/WARN/INFO/ERROR/FATAL but it is recommended to have INFO as a default log level for Fusion Framework.

Please refer below link to know more about logging levels supported by log4j:

<https://logging.apache.org/log4j/2.0/manual/customloglevels.html>

We have set **rootLogger** as “Combined” to combine logs of all test types for one test case execution.

**How to read the Log statement:**

For each log statement in respective log file, we have appended (TEST\_NAME ) and (Calling Package, Class and Method name) so that we can quickly identify for which tests corresponding log statement is appearing. Also, this will help to identify which test method, plugin method is throwing the ERROR. This will help debugging easy in case of any issue found.

**For e.g.**

ERROR 2017-08-17 17:16:14,307 - (TEST\_NAME: com.api.tests.ibm.LCOBProductsTest - LCOBProducts), com.cybage.apiframework.restbasedriver << getAPI < MESSAGE: Application Exception occurred HTTP-Get:Request sending for resource 'https://lcboapi.com/products' has been failed.

Here, the error message “Application Exception occurred HTTP-Get:Request…” is observed for “TEST\_NAME: com.api.tests.ibm.LCOBProductsTest – LCOBProducts” and failed for the plugin method **“com.cybage.apiframework.restbasedriver << getAPI”**.

##### **Exception Handling**

Fusion exception handling is categorized in the following sub categories

1. Application exception
2. Framework exception

##### **Application Exception**

Any method related to an application must throw an application specific exception, that should be self explanatory in the report.

**Application specific exception report sample:**

**catch** (Exception ex) {

ReportLogger.LogFail(ExceptionType.***Application***, ex.getMessage());

}

##### **Framework Specific Exception**

Any method related to framework or core logic of framework must throw framework specific exception, that should be self explanatory in the report.

**Framework specific exception report sample:**

**catch** (Exception ex) {

ReportLogger.LogFail(ExceptionType.***Framework***, ex.getMessage());

}

##### **Folder Structure Test Case AND EXTENDED PACKAGES**

In AutomationTestProject, test packages are maintained under *src/test/java* folder. And for each test type there can be multiple packages created based on the need or to segregate as appropriate. Test package name must be started with the prefixes as mentioned in table below:

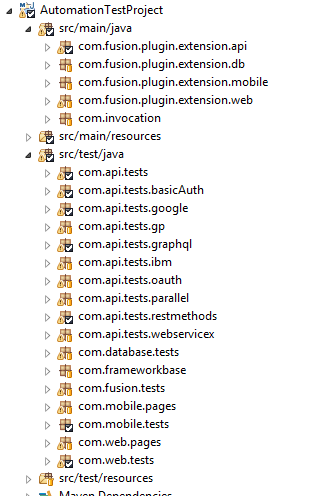
Also, given exampl

|  |  |  |
| --- | --- | --- |
| **TEST TYPE** | **TEST PACKAGES (src/test/java)** | **Example (with further segregation)** |
| API | com.api.tests | com.api.tests.webservices  com.api.tests.restservices |
| DATABASE | com.database.tests | com.database.tests.rdbms  com.database.tests.files |
| FUSION | com.fusion.tests | com.fusion.tests.acceptance |
| MOBILE | com.mobile.tests | com.mobile.tests.android  com.mobile.tests.ios |
| WEB | com.web.tests | com.web.tests.smoketest  com.web.tests.regression |

Extended packages for plugins to include new or overrride methods

|  |  |
| --- | --- |
| **TYPE** | **Extension PACKAGES (src/test/java)** |
| API | com.fusion.plugin.extension.api |
| DATABASE | com.fusion.plugin.extension.db |
| MOBILE | com.fusion.plugin.extension.mobile |
| WEB | com.fusion.plugin.extension.web |

**Sample Folder Structure:**



##### **Extent Report**

**Introduction:**

* Extent report is used here in Fusion framework with usage of libraries like TestNG v6.11 and ExtentReport v3.1.0
* Extent report is implemented to reflect testNG output/behaviour, so if extent report marks a test as 'Skip' then it’s because of underlying testNG behavior.
* Extent report creates test object for every data iteration of test method (@Test ). It will also provide an additional test entry object separately for any configuration method failures where all such failures can be seen together for every test (<test> in testNG xml) within a given suite.

**Mandatory Usage:**

To ensure reports are seen and reflected properly following points have to be taken care -

* Atleast 1 @BeforeMethod and 1 @AfterMethod has to be there for every test method (@Test) targeted either in it’s class or any of it’s parent class otherwise report will misbehave. This mandatory method is already added in DriverBase class so don’t make any changes to this class or methods and extend all your test class through DriverBase.
* Every @BeforeMethod should pass Method([java](eclipse-javadoc:%E2%98%82=AutomationTestProject/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_101%5C/jre%5C/lib%5C/rt.jar%3Cjava).[lang](eclipse-javadoc:%E2%98%82=AutomationTestProject/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_101%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang).[reflect](eclipse-javadoc:%E2%98%82=AutomationTestProject/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_101%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.reflect).Method) argument as the last argument which will be used internally within TestListener, else it will throw Exception and report will misbehave.(refer code snippet)

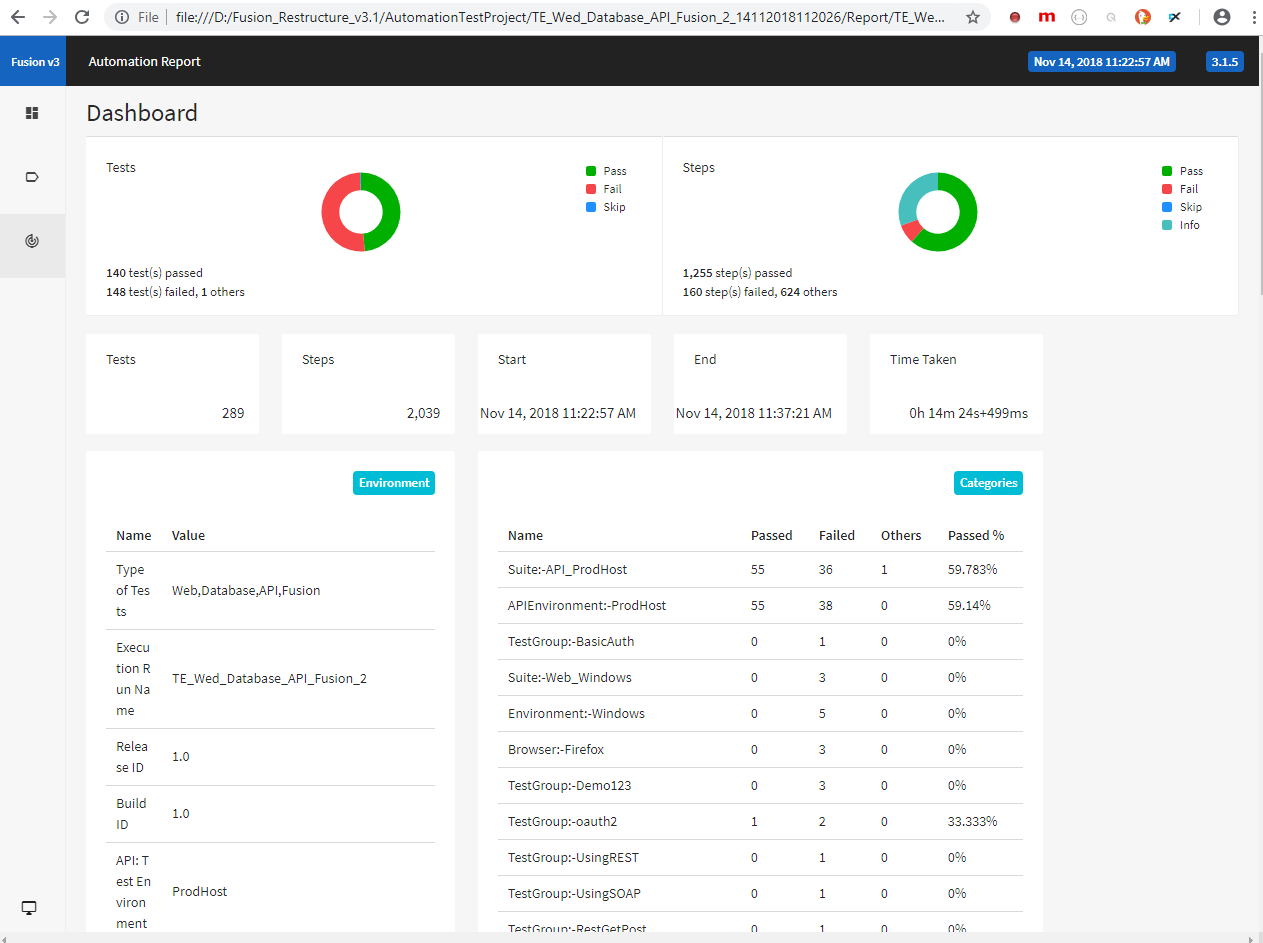


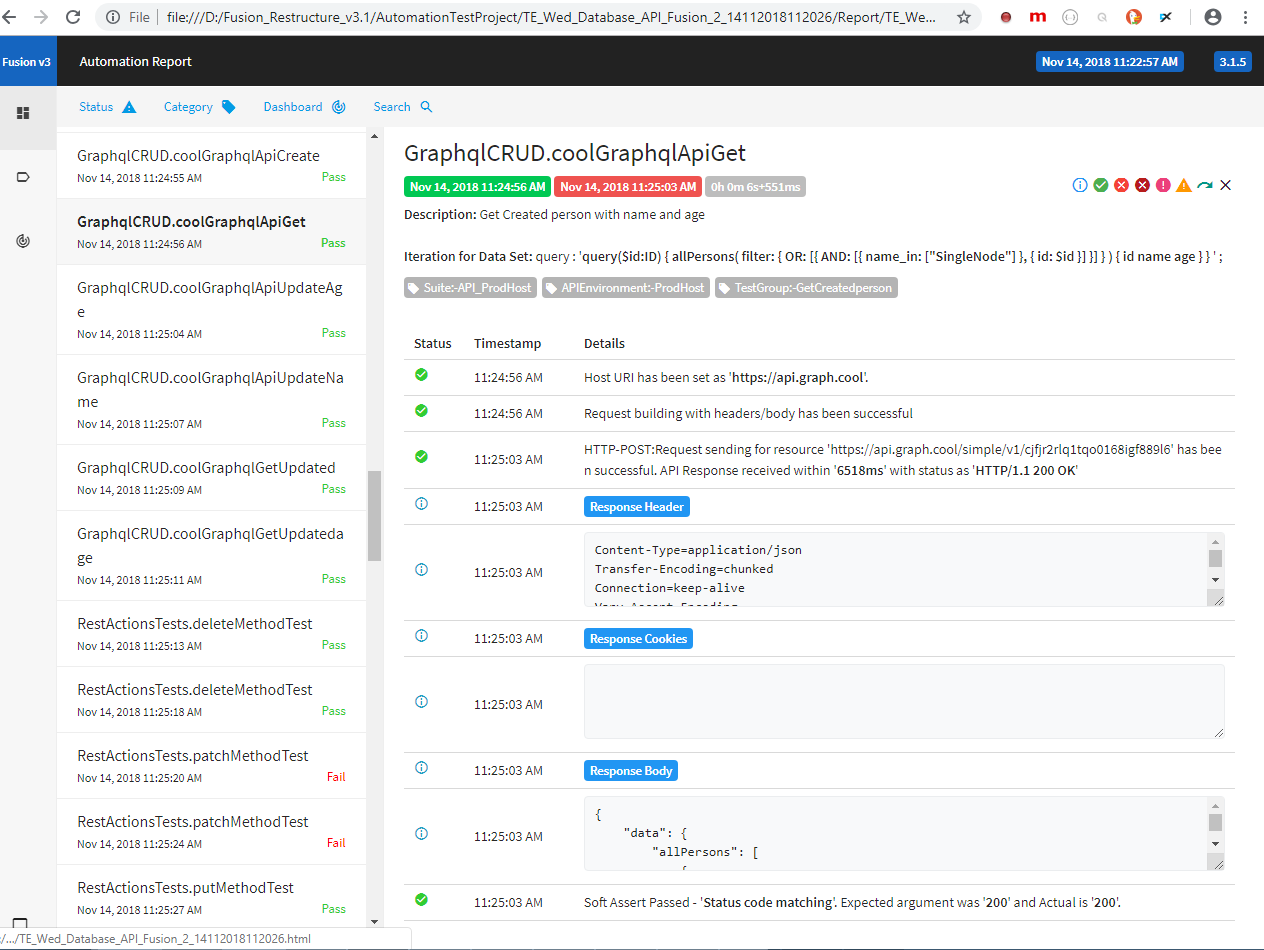
* Report object(ReportLogger) should be used only within/through @BeforeMethod ,@Test blocks. It can also be used in the immediate first call of @AfterMethod block after @Test otherwise if used anywhere else report will fail and throw NullPointerException.
* Try/catch is **NOT** required for any configuration and test methods like @BeforeMethod,@AfterTest,@Test etc. as it’s taken care in framework to handle/report any exceptions. In case if you still need to add some exception handling for some reason (like for custom messages) then make sure you throw the exception in catch block again after your handling. Refer code snippet below.

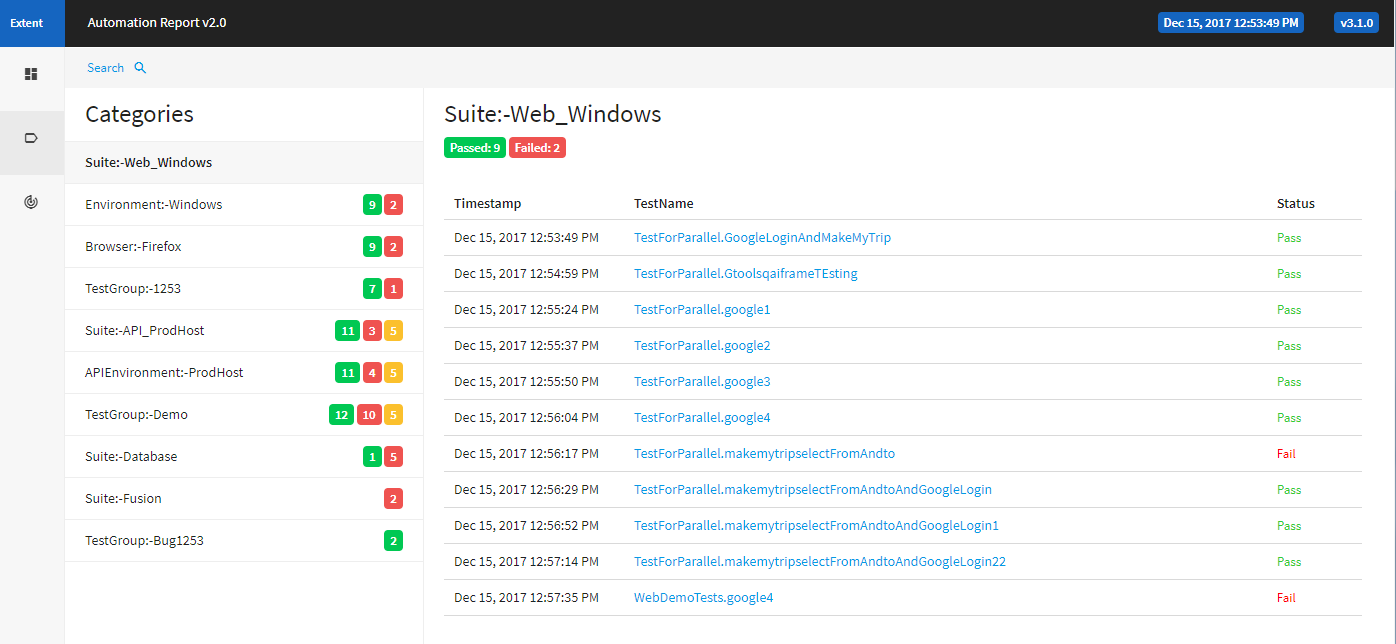


**Contact Us For:**

* Any different behavior observed on report as compared to TestNG output especially when you run TestNG xml file directly instead of UI.

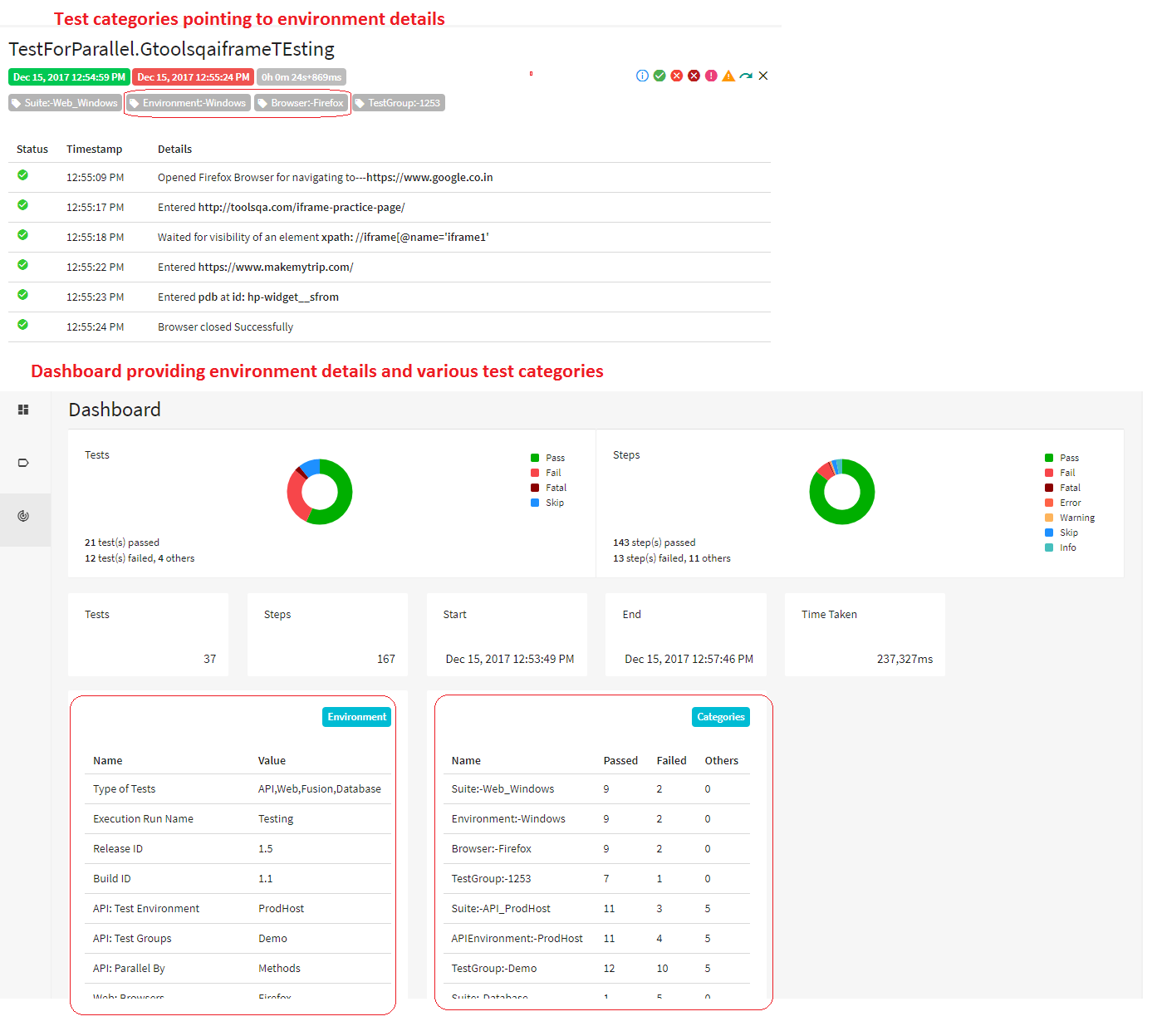






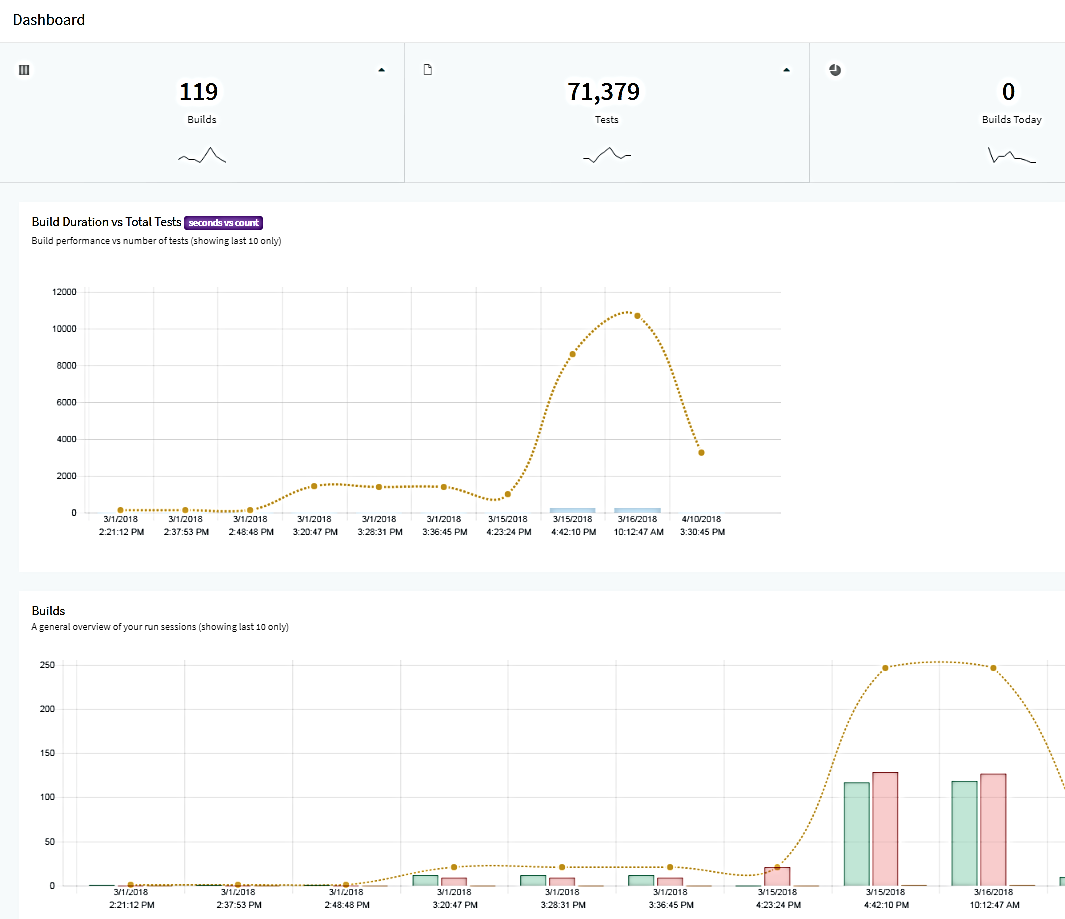
**Reports environment summary:**

In Fusion, test environment details, test groups are captured as individual categories and reflected in reports at various places under each Test case, Categories tab page, Summary page etc.. It even helps user to filter and analyze test results faster.

****

##### **KLOV Analytics**

For persistent test execution results storage and for various vital data analytics, Fusion also provides integration with Klov Analytics Server which Extent report provides. Klov internally uses MongoDB which stores all data through extent report for an execution. Using klov analytics server and its dashboard application we can do trend analysis of test execution time against results . It also provides various other features like you can compare tests from across builds for step-by-step comparison . All this informations are maintained for each project configured in Klov.



MongoDB and Klov Installation : Refer <http://extentreports.com/docs/klov/> (Follow instructions here)

Install version compatible with required Extent Report version. Once MongoDB and klov is installed , configure Klov settings (application.properties) to map with MongoDB host as given in above url page. Redis could be optional for use. Fusion internally is configured to work with Klov. Use following settings to configure running Klov and mongoDB within Fusion.

Open ExternalConfig.ini file under ‘AutomationTestProject’ folder of Fusion framework and add following lines with given section name.



If above option of “SaveReport” is set to ‘Yes’ then Fusion will look for other settings of MongoDB and Klov to map and save data. Also ensure MongoDB and Klov server are running before you execute Fusion. If data is not required to be saved in DB then set it as ‘No’ and rest of the settings will be ignored.

Make sure “ProjectName” is same for all test execution under a given project. Analysis will be provided within each project and not across project.

# Adding a Test Script

## How to add a Web UI Test Script

Web tests in a Fusion are designed considering Selenium 3.0 by following best design practices and standard PageObject pattern using PageFactory.

##### Web UI Test Creation flow

1. Add required application configuration in the **app Config.ini** file
2. Create Java class inside AutomationTestProject-> **com.web.tests**
3. Create page objct repository.
4. Create a test script using the test script format, page objects, asserts.

Sample test script from the AutomationTestProject can be used as an template for creating new test script.

##### Test Configuration

Refer appConfig section

Add Web tests related confuguration items under section **[WebTesting]**

* Mandatory:

In the appConfig.ini set the value of ImplicitWaitSec for webdriver implicit wait time.

* Optional:

If user wants to add few more configurable items for web tests, he can add those

under same section.

##### Test Script creation

* **Location of adding web tests:**

Add Web UI test scripts under package : com.web.tests There can be sub packages under com.web.tests

* **Mandatory things:**
  + Extend each test class with DriverBase
  + Add BeforeMethod to call getUIDriver method with parameters given in below code snippet:

@Parameters({ "OSToRun", "BrowsersToInclude" })

@BeforeMethod(alwaysRun = **true**)

**public** **void** googlesetupnew(String OS, String Browser, Method m) {

getUIDriver(OS, Browser, "Web");

}

These parameters are the values provided by user on the Execution UI.

* + Mark each test method with @Test and specify the group it belongs to.

@Test(groups = { "Smoke","Regression", })

* Optional::
  + Test description and Test data provider are optional attributes of @Test annotation

Example:

@Test(groups={"Smoke"}, dataProvider = "TestDataFile" ,description= "Tests for Google login functionality")

* Adding Test Steps:
* Within each testscript create an object of PageWrapper as below:

UIPageWrapper pageWrapper = **new** UIPageWrapper(uiDriver());

[Refer Page Object Repository](#_Page_Object_Repository)

* uiDriver() gives all methods available for IUIDriver.

[Refer WebUI Plugin Methods](#_Basic_Web_library)

Example:

uiDriver().clickOn(pageWrapper.pageGoogle.signin);

* One can also add asserts as an step.

[Refer Adding Asserts](#_Asserts)

##### **Page Object Repository**

* **Location of adding page object repository:**

com.web.pages

* Pages:

Fusion follows Page factory pattern.

* Create new page class
* Below is the class signature.
* Add your page objects as highlighted below:

IUIDriver driver;

**public** Page\_Google(WebDriver uidriver) {

**this**.driver = uidriver;

PageFactory.*initElements*(driver.getDriver(), **this**);

}

@FindBy(id = "gb\_70")

**public** WebElement signin;

* UIPageWrapper:

Intention of this class is to instantiate all the page classes at a one go. When a new page is introduced, its entry should be added in PageWrapper as below:

pageGoogle= **new** Page\_Google(uidriver);

Note: This is a default class hence need not to create again.

##### **Web Test Setup**

* **Location of DriverBase:**

com.frameworkbase

getUIDriver method from DriverBase.java is responsible for doing setup activity for WebUI tests.

It is recommended not to make any changes in this. But if required to make any changes below are the details:

* Class level declaration for WebUI is as follows:

This ensures parallel execution of web tests for making the test execution thread safe.

protected Map<String, IUIDriver> selDriver= new HashMap<String,IUIDriver>();

* Method for Web Tests setup: **getUIDriver(..)**

**public** **void** getUIDriver(String OS, String Browser, String testType) { *logger*.Log(LogType.***INFO***,"In Web Driver");

IUIDriver selUIDriver = **new** SelUIDriver(*ReportLogger*); setUIDriver(selUIDriver);

uiDriver().uiDriverSetup(IniFileOperations.*getValueFromIniFile*(IniFileType.

***Execution***, testType + "Testing", "WebBaseURL"),

OS,

Browser,

testType “Testing");

}

Depending on the test type as Web or Fusion, appropriate values for URL/Operating System/Browsers will be picked up(as given on the execution UI) and will be passed on to uiDriverSetup method for web driver setup.

* For parallel execution below 2 methods are used for thread safety. It is recommended not to make any changes in those.
* setUIDriver
* uiDriver
* Method for Web Tests destroy: **TestDestroy(..)**

This method closes the browsers by calling WebLibrary method **selquit()**

Any activity required to be performed after every test method of test type Web/Fusion, can be added inside the below if block of TestDestroy method:

**if**(context.getSuite().getName().contains("Web")){

**….**

**}**

## How to add a Mobile Test Script

Mobile tests in a Fusion are designed using Appium, following best design practices and standard PageObject pattern using PageFactory.

##### Mobile Test Creation flow

1. Add required application configuration in the **app Config.ini** file
2. Create Java class inside package **com.mobile.tests**
3. Create a test script using the test script format, page objects, asserts.

Sample test script from the AutomationTestProject can be used as an template for creating new test script.

##### Test Configuration

Refer appConfig section

Add Mobile tests related confuguration items under section **[MobileTesting]**

* Optional:

If you want to add application configuration items for mobile tests, you can add those

under MobileTesting section.

##### Test Script Creation

* **Location of adding Mobile tests:**

Add Web UI test scripts under package : com.mobile.tests There can be sub packages under com.Mobile.tests

* **Mandatory things:**
  + Extend each test class with DriverBase
  + Add BeforeMethod to call getMobileDriver method with parameters given in below code snippet:

IMobileDriver mobileDriver;

@Parameters({ " udid" })

@BeforeMethod(alwaysRun = **true**)

**public** **void** mobileSetUp(String udid, Method m) {

createMobileDriver(udid, "Mobile");

}

Parameters are the values provided by user on the Execution UI.

* + Mark each test method with @Test and specify the group it belongs to.

@Test(groups = { "Smoke" , "Regression" })

* Optional::
  + Test description and Test data provider are optional attributes of @Test annotation

Example:

@Test(groups={"Smoke"}, dataProvider = "TestDataFile" ,description= "Tests for Google login functionality")

* Adding Test Steps:
* mobileDriver gives all methods available for IMobileDriver.

Refer Mobile Plugin Methods

Example:

IMobileDriver driver = getMobileDriver();

MobilePageWrapper mobilePageWrapper = new MobilePageWrapper(driver);

mobileDriver.clickOn(mobilePageWrapper.pageGoogle.feelingLucky);

mobilePageWrapper gives access to all Mobile pages.

Refer Page Object Repository

* One can also add asserts as an step.

Refer Adding Asserts

##### Page Object Repository

* **Location of adding page object repository:**

com.mobile.pages

* Pages:

Fusion follows Page factory pattern.

* Create new page class
* Below is the class signature.
* Add your page objects as highlighted below:

IMobileDriver driver = **null**;

**public** Page\_Google(IMobileDriver mobileDriver) {

**this**.driver = mobileDriver;

PageFactory.*initElements*(mobileDriver.getDriver(), **this**);

}

@FindBy(name="btnG")

**public** WebElement searchButton;

* PageWrapper:

Intention of this class is to instantiate all the page classes at a one go. When a new page is introduced, its entry should be added in PageWrapper as below:

pageGoogle= **new** Page\_Google(mobileDriver);

Note: This is a default class hence need to create again.

##### Mobile Test Setup

* **Location of DriverBase:**

com.frameworkbase

getMobileDriver method from DriverBase.java is responsible for doing setup activity for Mobile tests.

It is recommended not to make any changes in this. But if required to make any changes below are the details:

* Class level declaration for Mobile is as follows:

**protected** MobilePageWrapper mobilePageWrapper;

**protected** IMobileDriver mobileDriver;

* Method for Mobile Tests setup: getMobileDriver **(..)**



* Depending on the test type as Mobile or Fusion, appropriate values for Device Name/URL will be picked up(as given on the execution UI) and will be passed on to setup method for mobile driver setup.
* Method for Mobile Tests destroy: **TestDestroy(..)**

This method closes the browsers by calling MobileLibrary method **quit()**

Any activity required to be performed after every test method of test type Mobile/Fusion, can be added inside the below if block of TestDestroy method:

**if**(context.getSuite().getName().contains("Web") || context.getSuite().getName().contains("Fusion")) **{**

**…..**

**}**

## How to add a Database Test Script

##### Test Configuration

This Section refers to the various configurations required to setup connection strings for various RDBMS and file systems. Users are required to follow the syntax and use of parameters as described in the below table.

|  |  |
| --- | --- |
| **AppConfig.ini** | In case of Database Test **appConfig.ini** is an input to Execution UI, that needs to be with certain configuration parameters defined in the respective **ConnectionStrings** for File / Database. |
| **Sample AppConfig.ini** | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **ConnecString Parameters** | **Mysql** | **MSSQL** | **XLXs** | **CSV** | **json** | | **URL** | Required e.g.  jdbc:mysql://172.27.XXX.XXX:3306 | Required e.g.  jdbc:sqlserver://172.27.XXX.XXX:1433;databaseName=TestDBNew | Required e.g.  D:\GP\Temp\emp.xlsx | Required e.g.  D:\GP\Temp\emp.csv | Required e.g.  D:\GP\Temp\emp.json | | **USERNAME** | Required | Required | NA | NA | NA | | **PASSWORD** | Required | Required | NA | NA | NA | | **TYPE** | Required e.g.  **RDBMS** | Required e.g.  **RDBMS** | Required e.g.  **FILE** | Required e.g.  **FILE** | Required e.g.  **FILE** | | **DATASOURCE** | SectionName e.g. **Staging\_Mysql** | SectionName  e.g **Staging\_Mssql** | SectionName  e.g. **Source\_xlsx** | SectionName  e.g **Source\_csv** | SectionName  e.g. **Source\_json** | | **PROVIDER** | Required  mysql | Required  mssql | Required  xls/xlsx | Required  csv | Required  json |   [**DatabaseTesting**]  [**Staging\_Mysql**]  ConnectionString="URL=jdbc:mysql://172.27.XXX.XXX:3306;USERNAME=root;PASSWORD=\*\*\*\*\*\*;TYPE=RDBMS;DATASOURCE=Staging\_Mysql;PROVIDER=mysql"  [**Staging\_Mssql**]  ConnectionString="URL=jdbc:sqlserver://172.27.XXX.XXX:1433;databaseName=TestDBNew;USERNAME=sa;PASSWORD=\*\*\*\*\*\*\*\*;TYPE=RDBMS;DATASOURCE=Staging\_Mssql;PROVIDER=mssql"  [**Source\_xlsx**]  ConnectionString="URL=D:\GP\Temp\emp.xlsx;TYPE=FILE;DATASOURCE=Source\_xlsx;PROVIDER=xlsx"  [**Source\_csv**]  ConnectionString="URL=D:\GP\Temp\emp.csv;TYPE=FILE;DELIMITER=",";COLUMNLINE=1;DATASOURCE=Source\_csv;PROVIDER=csv"  [**Source\_json**]  ConnectionString="URL=D:\GP\Temp\emp.json;TYPE=FILE;DATASOURCE=Source\_json;PROVIDER=json"  [**Target\_xlsx**]  ConnectionString="URL=D:\GP\Temp\emp.xlsx;TYPE=FILE;DATASOURCE=Source\_xlsx;PROVIDER=xlsx" |

##### Test Setup/ Tear Down

This section refers to the pre-test execution configurations.

This

|  |  |
| --- | --- |
| **Title** | **Detail Description** |
| **DriverBase Class** | This class allows user to initialize RDBMS, File Drivers, it also build connection objects based selected connections in Execution UI.  Please do refer Test Configuration Section 7.3.1 to get more information about constructing connectionstring in AppConfig.ini. |
| **@BeforeTest** | If project don’t have Database Tests as a part of there test execution then these methods can be commented out to avoid failures.  **@BeforeTest**: Since it is datacentric connection, connections must be done one time before execution of test gets started. |
| **@AfterTest** | **@AfterTest**: Conenction made in @BeforeTest method need to be closed after execution of all test cases for database and files. |
|  |  |

##### Test Script Setup

This section describe how to setup database tests in the test project.

|  |  |
| --- | --- |
| **Test class location** | Test class must reside under the package **com.database.tests** in automation test project. |
| **Parent Class** | Every test class needs to be extended by **DriverBase** by default. It is recommended not to do any modifications in the DriverBase Class. |
| **Before Method Setup** | **@BeforeMethod** must be a part of Test class. Basic code snippet of it is as below.  Before method is mandatory to be included in each test class even you are not using for any pre configuration step before executing test method.  @BeforeMethod(alwaysRun = **true**)  **public** **void** beforeMethod(Method m) {  } |
| **Test Method Template** | Below is the standard template one need to follow while constructing test scripts.  To mark test method eligible for execution @Test annotation along with appropriate group must be defined, otherwise test script won’t get referred during execution.  @Test(groups = { "GroupKey" })  **public** **void** MethodName() {  **……**  **……**  *softAssert*.assertEquals(……..);  *softAssert*.assertAll();  } |
| **Sample Test Method** | @Test(groups = { "exec" })  **public** **void** MSSQL\_CheckTableExists() {  **boolean** TableExists = **false**;  scrip  TableExists = *dbDriver*.isTableExist("Staging\_Mssql", "Sales", "Region");  **if** (TableExists) {  *softAssert*.assertEquals(TableExists, **true**,"Checking table does exist in database!!");  }  *softAssert*.assertAll();  } |
| **Mandatory** | * Annotations: @Test(groups = { "exec" })   *softAssert*.assertEquals(TableExists, **true**,"Checking table does exist in database!!");  *softAssert*.assertAll(); statement at the end to get all asserts logging. |
| **Optional** | Test description and Test data provider are optional attributes of @Test annotation  Example:  @Test(groups={" exec "}, dataProvider = "TestDataFile" ,description= "Check table exists.") |
| **After Method Setup** | There is no significant use of **@AfterMethod** as far as Database Test is concerned, so can be ignored here, unless one has done proactive configuration in @BeforeMethod which needs to wrap up at the end of @Test TestMethod execution. |

## How to add API Test Script

API tests are designed based on API adapter plugin created over REST Assured Library with its supporting features.

##### API Test Creation flow

1. Add necessary API host and resource configuration in the **appConfig.ini** file
2. Add required test configuration in the **appConfig.ini** file
3. For data driven tests ,create excel files and sheets to test each API .

4. Create test scripts using API driver plugin methods..

Sample test script from the AutomationTestProject can be used as an template for creating new test script.

##### Test Configuration

Refer appConfig section

* 1. Create API resource repository as per various environments-
* Projects may have to test API against an individual environment or multiple environments like different hosted environments or test environments - Production, Staging etc.
* For every environment where API needs to be tested , create corresponding API\_<ENV\_NAME> section in appConfig.ini like **[API\_ProdHost].** These environment names are provided to Fusion UI during test execution. (Refer Fusion UI section-API Tab)
* Add host name and all corresponding resource urls within that section wih proper identifier keys.

e.g refer section added below for a given REST API

<https://maps.googleapis.com/maps/api/geocode/json?address=pune&sensor=false>



* These identifer keys like ‘Host’ or ‘JSONGeocode’ will be used within your test script to pull corresponding values against a given environment.
* [Optional]: If users have multiple environments with different host names but got same resource URLs then common resource details can be added under a common section like **[APITESTING]** and all individual host name will be within corresponding environment section as given below.



* [OPTIONAL] : If API needs login credentials/tokens for its access then all such credentials can also be added under required API sections.



* 1. Add API tests related configuration items under section **[APITesting]**
* [Optional]: Users can add certain configurable API features here in this section as given below. These keys and values are optional 
  + ValidHTTPCodes : This enables hard assertion of all APIs against given HTTP status codes. Default values is ‘blank’ which means all HTTP status codes are considered as Valid.
  + UrlEncodingEnabled : This feature controls the encoding of special characters for API url. Default value is ‘true’.

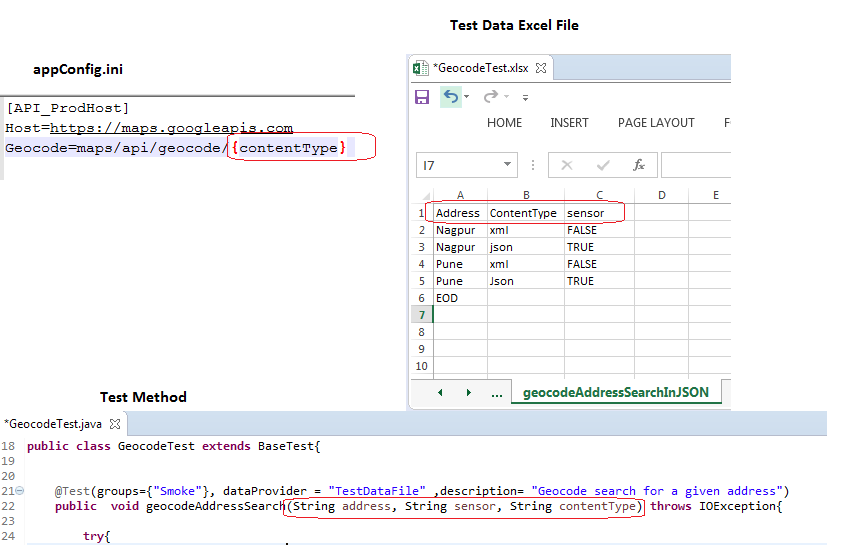
*(For more details on above features refer API plugin section)*

##### Test data Creation

* 1. Create necessary test data excel files as required test data as given below-
* **Creation of Excel files** 
  + Add corresponding excel files( .xlsx) and sheets with all parameters to support data driven test approach. Refer TestNG DataProvider for excel file conventions.
  + Parameters for API method can be query/ path/form parameters . Even entire post body and required values in it can be parameterized .

e.g for given below REST GET API , its path and query parameters can be parametrized and fetched from corresponding excel files.

Actual API : 



* **Creation of TestData**
  + Create necessary test data against identified parameters required for API and functionality to test .

(Refer sample test attached in fusion framework)

##### Test Script creation

* 1. Add API test scripts based on API resources and test data identified -
* **Location of adding API tests:**

Add API test scripts under package : com.api.tests There can be sub packages under com.api.tests specific to projects and modules.

* **Mandatory things:**
  + Extend each test class with DriverBase. You can also add an intermediate test class extending from DriverBase (to add common methods across multiple test classes) and extend it further to add target test class.
  + API driver can be called/instantiated within test method (@Test) or in @BeforeMethod through method ‘getAPIDriver’.Same driver object can be used to test multiple API endpoints in same test case.

Note: There is no need to close API driver unlike other drivers (UI/Mobile).

* + Mark each test method with @Test and specify the group it belongs to.

@Test(groups = { "Smoke" })

* **Optional:**
  + Test description and Test data provider are optional attributes of @Test annotation

Example:

@Test(groups={"Smoke"}, dataProvider = "TestDataFile" ,description= "Tests for Google geocode functionality")

* **Adding Test Steps:**

Test steps are usually added as per a test scenario. However for hitting an API request certain API driver plugin method needs to be called in following steps-

* + Create a unique driver instance for each API to hit within a test method.
  + Set Host URL for API
  + Build API with required parameters like query/path/form etc. using HashMap or a post Body as a String.
  + Hit the API with corresponding methods defined against required HTTP methods like GET/POST/PUT/DELETE etc.

(refer code snippet below)



* + Do required validations like on header or body content using API driver methods and inbuilt assertion objects like hardAssert/softAssert . Refer Assertion section



## How to add a FUSION Test Script

##### Fusion Test Creation flow

1. Add required application configuration in the **app Config.ini** file for Web, API and DB
2. Create Java class inside package **com.fusion.tests**
3. Create a test script using the test script format, page objects, asserts.

Sample test script from the AutomationTestProject can be used as an template for creating new test script.

##### Fusion Test Configuration

Refer to the Test Configuration section of API and Database for appCong.ini settings configuration.

[ **Section 6.3.1 – DB** and **Section 6.4.2 – API** ]

##### Fusion Test Script Creation

* **Location of adding Fusion tests:**

Add Fusion test scripts under package : com.fusion.tests

* **Mandatory things:**
  + Extend each test class with DriverBase

Public class TestFusion extends DriverBase

* + Add BeforeMethod to **call getUIDriver** and **MobileDriver** method with parameters given in below code snippet:
  + Also environment section to be added to get the **API environment**



* + Driver for API need to be initialized in the test method as given below in the code snippet.

public void fusionTest() throws Exception

{

IDataSet MySqlDS = null;

try {

// API tests : get API driver for rest

IHTTPDriver restDriver = getAPIDriver(APIDrivers.REST);

* + Driver for DB is already initialized in the BeforeTest method in DriverBase.

@BeforeTest(alwaysRun = true)

public void BeforeTestRun(ITestContext context) {

try {

getDBDriver(type);

}

If Assert statements are used in the test script those are required to be

Refer How to add asserts

* Optional::
  + Test description and Test data provider are optional attributes of @Test annotation

Example:

@Test(groups={"Smoke"}, dataProvider = "TestDataFile" ,description= "Tests for Google login functionality")

* Adding Test Steps:
* Within each testscript create an object of PageWrapper as below:

PageWrapper pageWrapper = **new** PageWrapper(uiDriver().getDriver());

Refer Page Object Repository

* uiDriver() gives all methods available for IUIDriver.

Refer WebUI Plugin Methods

Example:

uiDriver().clickOn(pageWrapper.pageGoogle.signin);

* mobileDriver() gives all methods available for IMobileDriver

Refer Mobile Plugin Methods

Example:

mobileDriver.navigateTo("http://www.facebook.com");

* RestDriver() gives all methods available for IHTTPDriver

Refer API Plugin Methods

Example:

restDriver.BuildAPI(headers, queryParams);

* dbDriver() gives all methods available for IConnect

Refer DB Plugin Methods

Example:

dbDriver.getDataset(“Staging\_MySql”,MyQuery)

* One can also add asserts as an step.

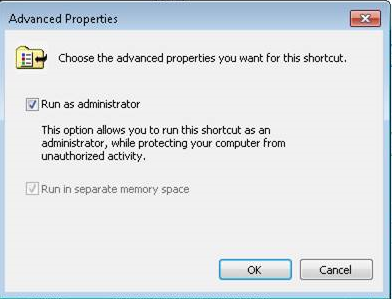
Refer Adding Asserts

# Test Execution

## Execute a Test from Fusion Test Runner

**For Windows 7 Enterprise, ensure that the cmd.exe runs as administrator**

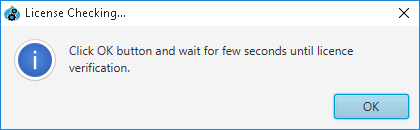
* 1. Go to C:\Windows\System32\
  2. Right click on cmd.exe 🡪 Properties 🡪 Advance Proporties
  3. Check on ‘Run as administrator’



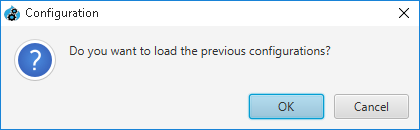
**Note: Refer to section 4.2.1 and 4.2.2 for setting up the config entries** ExecutionUIConfig.ini **for ExecutionUI:**

**Steps to run FusionTestRunner:**

1. Go to the folder where FusionTestRunner is placed. Open the FusionTestRunner.jar by double clicking the exe.
2. If you have done license verification earlier, click OK to proceed.



1. If you have done configuration earlier, it will display this message.

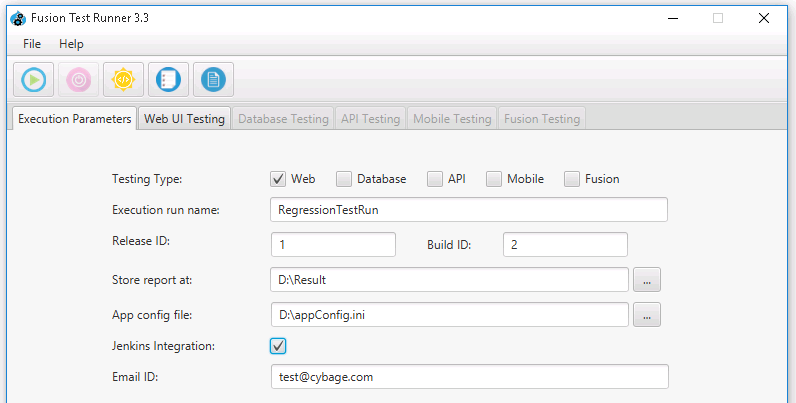


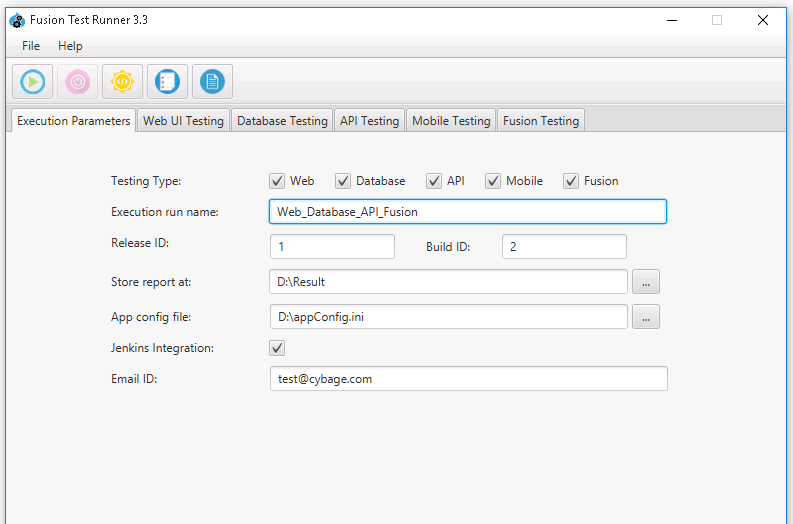
1. Once the Test Runner UI gets open, the spinner will rotate to indicate that the latest source code files are copied to following path mentioned in the file **ExecutionUIConfig.ini**

**TFSSourceRepositoryLocalMappingPath**=\\172.27.56.196\D$\Workspace\CybageCommonFramework\GenericFramework\AutomationTestProject

1. Once you click Ok, 6 tabs will be displayed. Execution Parameters, Web UI Testing, Database Testing, API Testing, Mobile Testing and Fusion Testing.

##### Execution Parameters tab





This is the default tab with generic settings to be configured to run the automation tests.

Select the checkbox for respective test type. Before going to the selected test type tab, first fill the information in Execution Parameters tab which is generic for all test types.

**Mandatory Fields:**

1. Mention the Execution run name of the tests those will be executed.
2. Mention the corresponding Release ID and Build ID.
3. Mention the location of the report to be placed after the execution completed in "Store report at".
4. Mention the location of AppConfig file through "App Config file " which is having all configuration details to run the DB/API automation tests.

**Sample Appconfig.ini entries:**

***[Staging MySQL]***

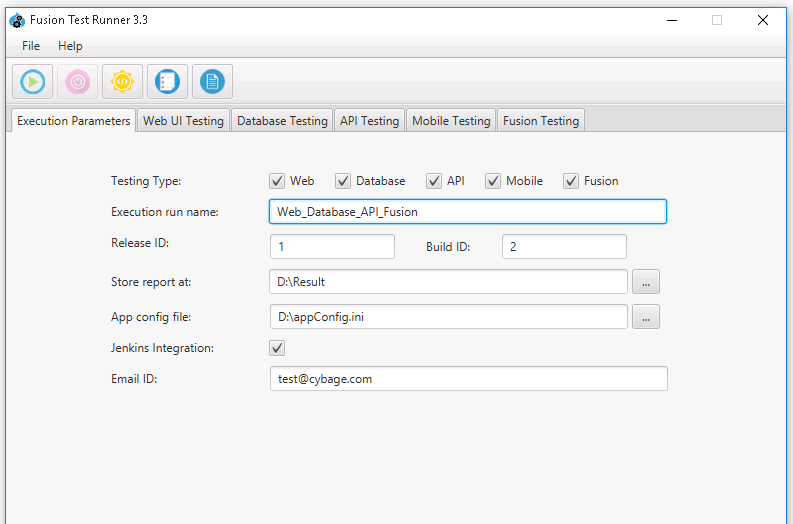
connectionString="URL=jdbc:mysql://localhost:3306/;USERNAME=root;PASSWORD=root;TYPE=RDBMS;DATASOURCE=Staging\_MySQL;Provider=jdbc"

***[Target\_csv]***

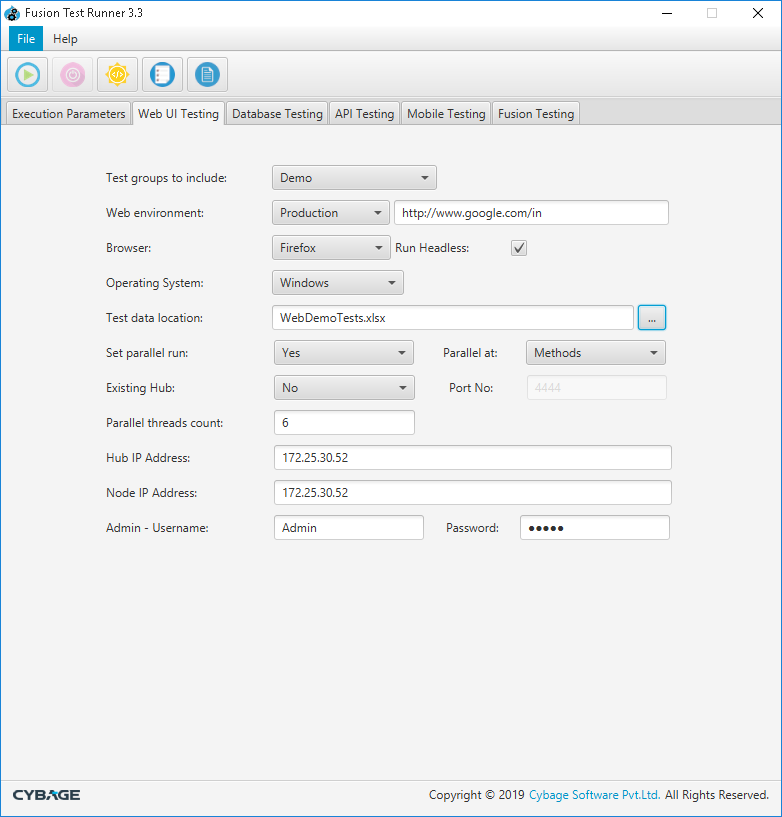
ConnectionString="URL=D:\test\Data\test.csv;TYPE=FILE;DELIMITER=",";COLUMNLINE=1;Provider=csv; DATASOURCE=Target\_csv"

**Optional Fields:**

* Select the checkbox for "Jenkins Integration" if you need to run the tests build from Jenkin tool. [ This will copy the source code / configuration files into Jenkin Jobs workspace folder.
* Mention the Email ID to get the test results report on the given email id.



Web UI Testing tab



If "Web" is selected as a test type then go to the "Web UI Testing" tab

**Mandatory Fields:**

1. Mention the Web Base URL of the automation tests from given dropdown.
2. Select the "browser" from given dropdown on which you need to execute the automation tests.
3. Select the checkbox “Run Headless” if you need to run tests on headless browser based on selected browser.
4. Select the "Operating System" from given dropdown on which you run the test cases.
5. Select the respective "Test groups to include" from given dropdown which you have targeted for execution.
6. Select the dropdown of "Set parallel run" as "Yes" if you need to execute UI test cases in parallel. Otherwise select "No".
7. If "Set parallel run" is selected as "Yes":

* Select the "Parallel at" support as either "Tests" or "Methods".
* Mention the "Parallel thread count" as one of the value from 1 to 9.
* Mention the Hub IP Address
* Mention the Node IP Address
* Mention Admin - Username and password.

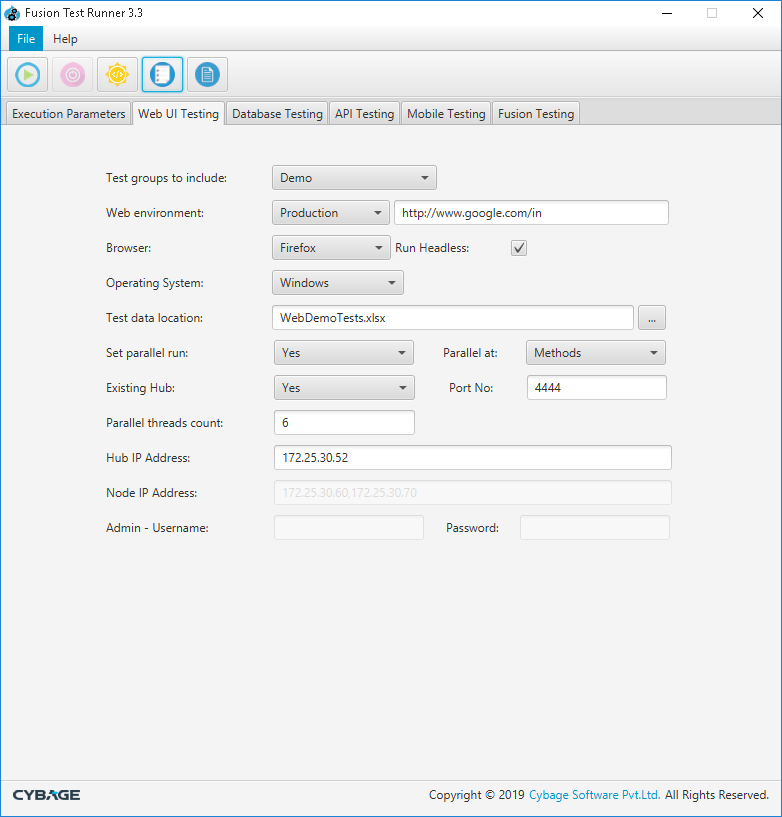
1. If "Set parallel run" is selected as "No":

 "Parallel at", "Parallel thread count", Hub IP Address, Node IP Address, Admin - Username and password will be disabled.

1. Existing Hub Feature: Fusion framework supports consumption of existing hub IP, in order to carry out execution on user established selenium grid. This feature is available for one operating system at a time i.e. Linux or Windows

In this user will be responsible to establish the selenium grid for browsers selected of single Operating system at a time. So the Framework would require below details to accept the user defined grid for execution

* Select “set parallel run” as Yes (Note: if this flag is Yes then only user can mention the Existing Hub IP of user defined grid.)
* Select Parallel at support as either "Tests" or "Methods".
* Select Existing Hub = Yes (Note: if this flag is yes then only user can mention the Hub IP of user defined grid.) (for this below mentioned fields are mandatory
* Port No: 4444
* Mention the "Parallel thread count" as one of the value from 1 to 9.
* Mention the Hub IP Address where your grid is running



1. Automatic hub node setup through framework, by mentioning details as Step 7 , this is applicable to Windows OS or Windows platform only, and this is not done for Linux or Ubuntu platform.

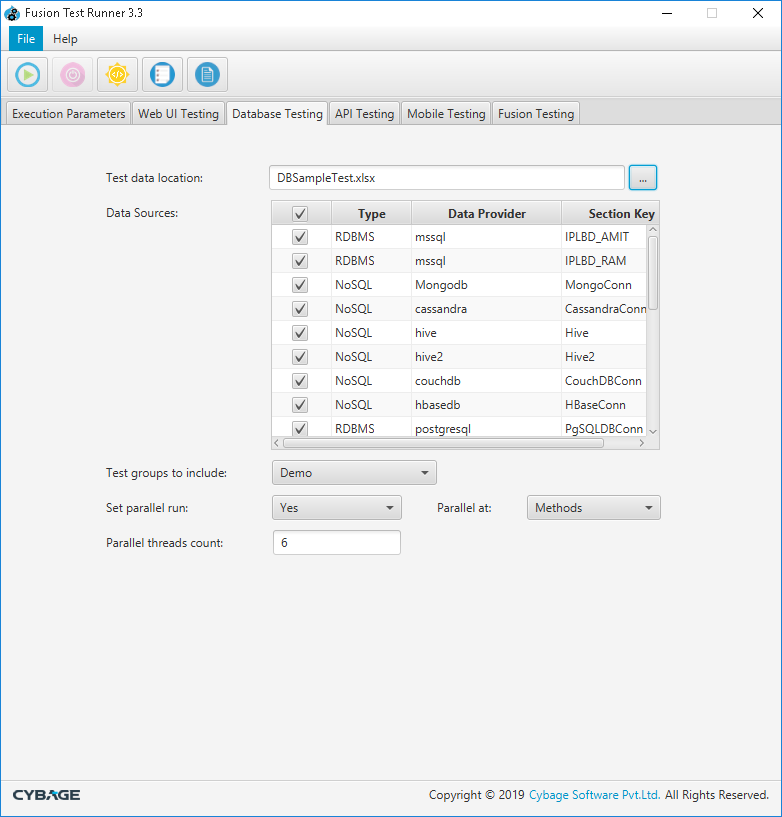
**Optional Fields:**

1. Mention the "Test data location" where you have placed the test data of the automation tests.

**Note:** If Parallel is selected as "Yes" but hub, node, username and password is not provided then execution will on local machine without Hub/Node setup.

Once added all the required fields in "Execution Parameters" and "Web UI Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for WebUI.

##### Database Testing tab



If "Database" is selected as a test type then go to the "Database Testing" tab.

**Mandatory Fields:**

**Data Sources:**

1. It will display the data from "appConfig.ini" uploaded through "Config file location" from "Execution Parameters"
2. This section will be having entries for Database Type, Data Provider, Section Key and checkbox for each entry present in the section.
3. Select the checkbox for the   corresponding combination of Database Type and Data Provider you need to require for Database tests.
4. Select the respective "Test groups to include" from given dropdown which you have targeted for execution.

**Optional Fields:**

1. Mention the "Test data location" where you have placed the test data of the automation tests.
2. If "Set parallel run" is selected as "Yes":

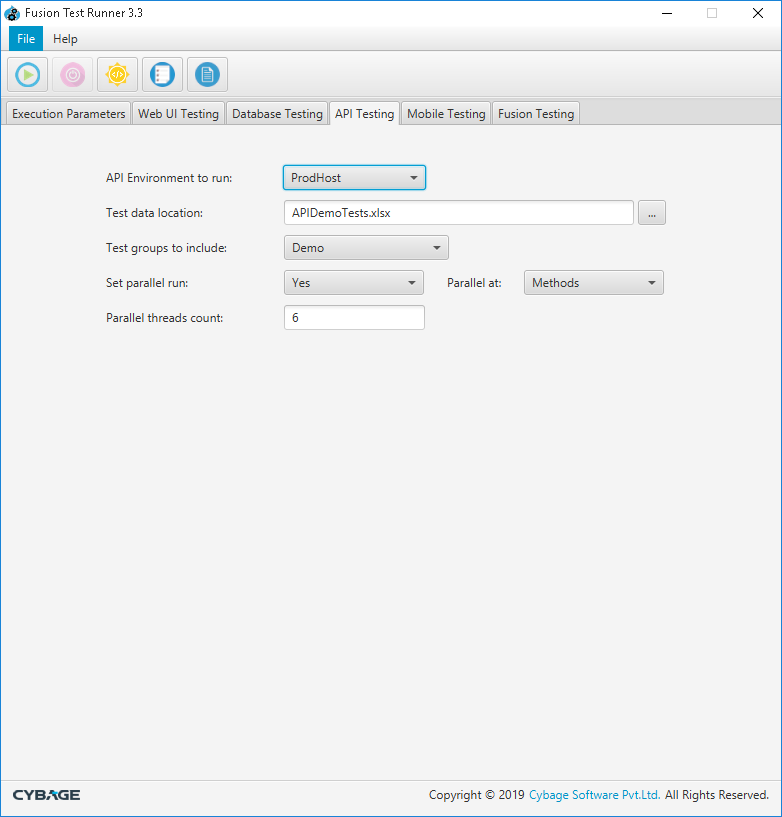
* Select the "Parallel at" support as either "Tests" or "Methods".
* Mention the "Parallel thread count" as one of the value from 1 to 9.

1. If "Set parallel run" is selected as "No":

* "Parallel at" and "Parallel thread count" will be disabled.

1. Once added all the required fields in "Execution Parameters" and "Database Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for Database.

##### API Testing tab



If "API" is selected as a test type then go to the "API Testing" tab.

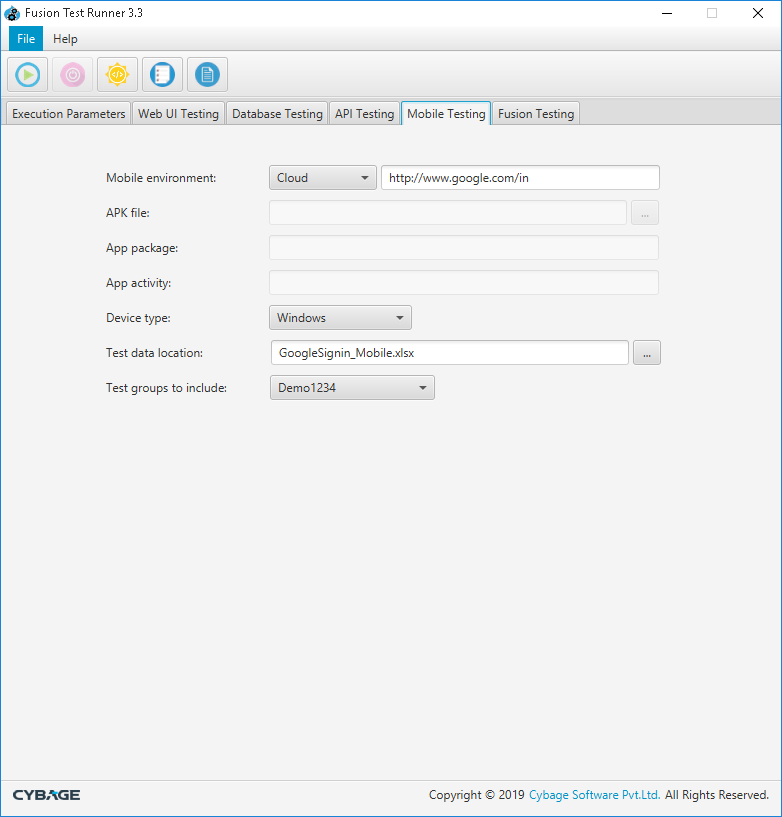
**Mandatory Fields:**

* Select the corresponding "API Environment to run" for API from the given dropdown required for API tests.
* Select the respective "Test groups to include" from given dropdown which you have targeted for execution.

**Optional Fields:**

* Mention the "Test data location" where you have placed the test data of the automation tests.
* If "Set parallel run" is selected as "Yes":
* Select the "Parallel at" support as either "Tests" or "Methods".
* Mention the "Parallel thread count" as one of the value from 1 to 9.
* If "Set parallel run" is selected as "No":
* "Parallel at" and "Parallel thread count" will be disabled.
* Once added all the required fields in "Execution Parameters" and "API Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for API.

##### Mobile Testing tab

If "Mobile" is selected as a test type then go to the "Mobile Testing" tab.

**Mandatory Fields for mobile Web application tests:**

* Mention the Mobile Base URL of web application automation tests from given dropdown.
* Select the "Device type" from given dropdown on which you need to execute the mobile automation tests.
* Select the respective "Test groups to include" from given dropdown which you have targeted for execution.

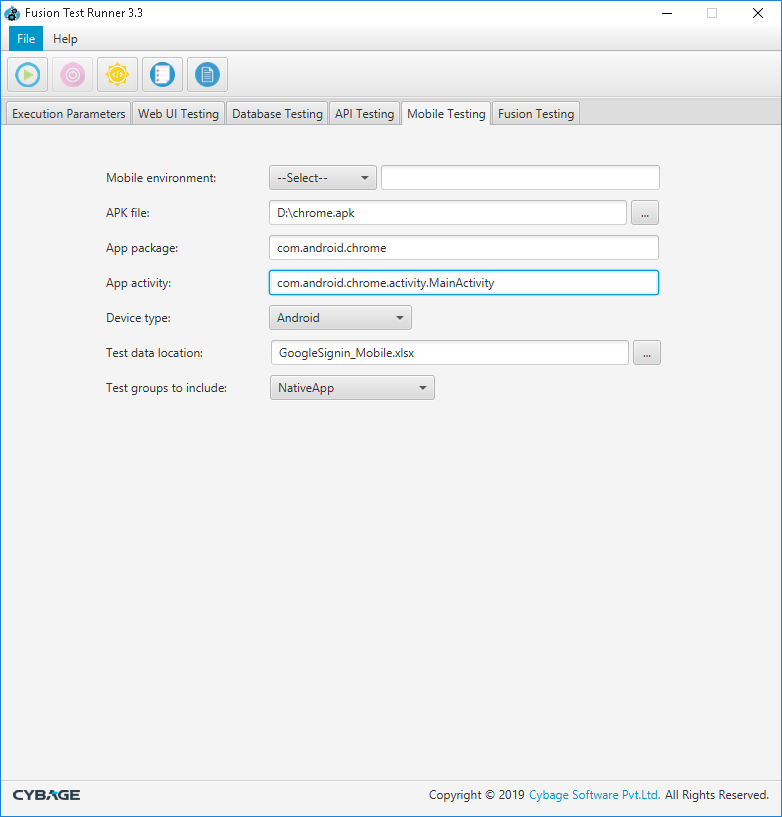
**Optional Fields:**

* Mention the "Test data location" where you have placed the test data of the automation tests.

Once added all the required fields in "Execution Parameters" and "Mobile Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for **Mobile**.

**Mandatory Fields for mobile Native or hybrid application tests:**

( Note: This feature is limited to Android devices only)



* Browse the native or hybrid app .apk file from “APK file”.
* Mention native or hybrid apps App package under “App package”.
* Mention native or hybrid apps App package under “App activity”.
* Select the "Device type" from given dropdown on which you need to execute the mobile automation tests.
* Select the respective "Test groups to include" from given dropdown which you have targeted for execution of native app or Hybrid app.

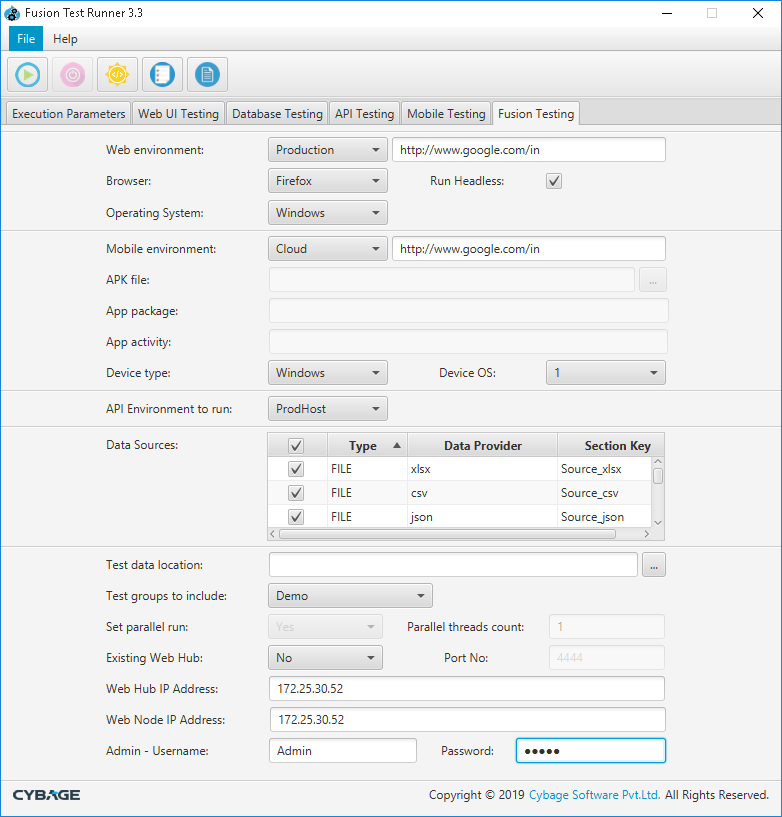
**Note:** When you are executing Native or hybrid app tests on device then Web application tests cannot be executed at the same time , this is true vice-versa. So “Mobile environment” should not be selected , if selected then data present under “APK file” ,”App package”,”App activity” will vanish.

**Optional Fields:**

* Mention the "Test data location" where you have placed the test data of the automation tests.

Once added all the required fields in "Execution Parameters" and "Mobile Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for **Mobile**.

##### Fusion Testing tab



**Mandatory Fields:**

**WebUI section:**

* + Mention the Web Base URL of the automation tests from given dropdown.
  + Select the "browser" from given dropdown on which you need to execute the automation tests.
  + Select the checkbox “Run Headless” if you need to run tests on headless browser based on selected browser.
  + Select the "Operating System" from given dropdown on which you run the test cases.

**Mobile section:**

* + Mention the Mobile Base URL of the automation tests from given dropdown.
  + Select the "Device type" from given dropdown on which you need to execute the mobile automation tests.
  + Select the "Device OS" from given dropdown on which you need to execute the mobile web application automation tests.

For Mobile Native, hybrid app end-to-end tests with API or database below configurations will be required for testing

* + Browse the native or hybrid app .apk file from “APK file”.
  + Mention native or hybrid apps App package under “App package”.
  + Mention native or hybrid apps App package under “App activity”.
  + Select the "Device type" from given dropdown on which you need to execute the mobile automation tests.
  + Select the respective "Test groups to include" from given dropdown which you have targeted for execution of native app or Hybrid app.

**Note:** When you are executing Native or hybrid app tests on device then Web application tests cannot be executed at the same time , this is true vice-versa. So “Mobile environment” should not be selected , if selected then data present under “APK file” ,”App package”,”App activity” will vanish.

**API section:**

* + Select the corresponding "API Environment to run" for API from the given dropdown required for API tests.

**Database section:**

* + Data Sources: If the "appConfig.ini" uploaded through "Config file location" from "Execution Parameters" tab is having the correct information about the database configuration then corresponding details will get populated in Data Sources section.
  + This section will be having entries for Database Type, Data Provider, Section Key and checkbox for each entry present in the section.
  + Select the checkbox for the corresponding combination of Database Type and Data Provider you need to require for Database tests.

**Common section for Fusion:**

* + Select the respective "Test groups to include" from given dropdown which you have targeted for execution.
  + Mention the "Test data location" where you have placed the test data of the automation tests.
  + If only Web Details (Web Base URL, Device type and OS) provided in web section then "Set parallel run" will be by default set as “No” and parallel threads count will be by default set as “1” and editable. In this case, hub, node, username and password will be enabled and editable. If hub, node, username and password fields are not provided then execution of web tests in Fusion tab will be triggered on local machine. If hub, node, username and password fields are provided then execution of web tests will be triggered on respective machine.
  + If only Mobile Details (Mobile Base URL, Device type and OS) provided in mobile section then "Set parallel run" and parallel threads count will be disabled. In this case, fields parallel threads count, hub, node, username and password will be disabled.
  + If both Web Base URL and Mobile Base URL are provided in mobile and web sections and irrespective of either “API Environment to run” for API or “Data Sources” for Database or both mentioned then "Set parallel run" and parallel threads count fields will be disabled. But fields hub, node, username and password will be enabled. If hub, node, username and password fields are not provided then execution of web tests in Fusion tab will be triggered on local machine. If hub, node, username and password fields are provided then execution of web tests will be triggered on respective machine.
  + If both Web Base URL and Mobile Base URL are not provided in mobile and web sections but either “API Environment to run” for API or “Data Sources” for Database or both mentioned then fields "Set parallel run", parallel threads count, hub, node, username and password will be disabled.

**Optional Fields:**

* + Mention the "Test data location" where you have placed the test data of the automation tests.

**Note:** If Parallel is selected as "Yes" but hub, node, username and password is not provided then execution will be on local machine without Hub/Node setup.

* + Once added all the required fields in "Execution Parameters" and "Fusion Testing" tabs, click "Generate Execution Run Config File" to start execution of test cases for Fusion.
  + Once added all the required fields in "Execution Parameters" and in respective testing tabs, click on "Generate Execution Run Config File" to start test execution. "Success" dialog box will appear with message "ExecutionRunConfig.ini" file has been successfully generated.
  + Click "OK", "Confirmation" dialog box will appear with message "Do you want to start the test cases execution?". If clicked "OK", spinner will be displayed to indicate execution is started.
  + If clicked "Cancel", "Confirmation" dialog box will disappear and execution will not trigger.

## Execute a Test from Framework

**Following are the ways to execute the test from Framework:**

1. POM
2. TestNG
3. KickOff.java.

Right click kickoff.java under com.invocation and run as java application

For debugging right click TestNG files of respective plugins and run as TestNG suite or debug as TestNG suite.

Note: if you are running through TestNG xmls then pre-requisite steps present under kickoff won’t execute.

## Execute a Test from Jenkins

**Following are the steps to execute the test from Jenkins:**

1. Refer to the Jenkins section to setup the Jenkins project and job.
2. Once the Jenkins project is created, click on the go to the project and click on Build Now – here it will create a workspace in Jenkins.
3. Go to Fusion Test Runner 🡪 ExecutionUIConfig.ini
4. Make sure below entries are filled:

**SourceRepositoryLocalMappingPath**=\\<<machineIP>>\TFSSourceCode\NKSampleCodeLocal*.(Note:Shared Path where you’re actual AutomationTestProject is present on local machine if you are using Test runner from windows machine. If you are using Fusion TestRunner on Linux machine then path will be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject) (Note :In order to access the project you need to share the root folder of your project with users who are going to access it on Windows machine)*

**SourceRepositoryPhysicalPath**=D:\Fusion\_Restructure\_v3.1\AutomationTestProject. *(Note: Actual Path where your AutomationTestProject is present on local Windows machine. Alternatively, if you were on Linux machine then this path would be for e.g. /home/cybage/Fusion\_Restructure\_3.1/AutomationTestProject)*

**TestServerIP**=172.27.56.xxx

**TestServerUsername**=cybage\username(For Linux machine ,do not specify the domain)

**TestServerPassword**=\*\*\*\*\*\*\*

**JenkinsServerIP**=172.27.56.xxx

**JenkinsServerUsername**=cybage\username(For Linux machine ,do not specify the domain)

**JenkinsServerPassword**=\*\*\*\*\*\*\*

**JenkinsWorkspacePath**=\\<<machineIP>>\\Jenkins\workspace\Jenkinsproject. (Shared Path of your Jenkins workspace to copy AutomationTestProject) *(Note :In order to copy the project you need to share the root folder of your Jenkins workspace with the users who are going to access it) (If you are using Fusion TestRunner on Linux machine then path will be for e.g. /var/lib/Jenkins/workspace)*

**JenkinsWorkspacePhysicalPath**=C:\Program Files (x86)\Jenkins\workspace\Jenkinsproject.(Actual Path of your Jenkins workspace to copy AutomationTestProject)

1. Enter all the Execution parameters and the configuration required test type to be run.
2. Open the Fusion Test Runner
3. On the Execution parameters tab**, click on Jenkins Integration checkbox.**
4. Enter the Execution Run Name e.g. “TE\_Smoke”
5. Click on ‘Generate Execution Run config file’
6. This will do following:

* It will copy source code from LocalMappingPath to Jenkins workspace
* Also it will create a folder with “TE\_Smoke” and add the configuration files in that folder.

(executionRunConfig.ini and appConfig.ini)

1. Now go to Jenkins 🡪 Select the Jenkins project and click on Build Now

**Note:** To achieve execution through Jenkins ,you need to schedule build trigger from Jenkins as per your requirement.

# Plugin Library

## API –plugin

##### Existing Plugin features and support

* 1. API adapter plugin is designed to adapt and support all features provided by REST Assured library.
  2. All features supported by REST Assured can be implemented within current adapter plugin to support. However some basic features are already implemented in it and are available .
  3. Users can add more features and support to plugin as per need of their individual project and test requirements using extensibility rules .Refer 8.1.4API plugin extension
  4. In addition to REST Assured features , current plugin and framework is designed to support parallel test execution at different levels like Method/Test/Suites/Environment.
  5. Other than REST web services , current plugin can also be used to test SOAP and GraphQL over HTTP/HTTPs.

Some of the existing library methods available for use are as follows . For more details and complete method list refer java-doc.

##### Basic API library methods

|  |  |
| --- | --- |
| com.cybage.apiframework.RESTBaseDriver Methods | Comments |
| RESTDriver | Constructor helps in setting up following features configured by user through appConfig.ini   * 1. Internal HTTP Status code assertion   2. API request/response logging for failures   3. Handling UrlEncoding for APIs |
| setHostUrl | Set the Host URL/URI for API to hit |
| buildAPI | Various overloaded buildAPI methods to build headers, query/path/form/query parameters and body into API request. |
| getAPI/postAPI/putAPI/ deleteAPI/ optionsAPI /headAPI/patchAPI | Various HTTP method based functions to hit the built API with corresponding resource parameter. |
| XmlUtils.buildSoapBody(bodyTemplate, bodyParams) | Method to build soap request body from a template |
| GraphqlUtils.createGraphqlQuery(query) | Method to create graphqlQuery from a query template |
| GraphqlUtils.createGraphqlQuery(String query, String variables) | Method to create graphql Query and the variables values separately as per the parameterization |
| GraphqlUtils. createCRUDGraphqlQuery(String query, String respid) | Method to create CRUD operation for graphql Query with the id as per the correlation |
| buildBody(String bodyTemplate, Map<String,String> bodyParams | Method to build rest json request body from a template |
| getResponseNodeValue(String path) | Returns string value of xml/json node as per given path. |
| getResponseStatusCode/ getResponseHeaders/ getResponseCookies / getResponseBody | Methods to return the response status code, headers, cookies and body or even values for a specific key |

(For more details and complete method list refer java-doc)

##### API plugin configuration features

* 1. Internal HTTP Status code assertion/ API request-response logging:
  + This feature is implemented based on RESTAssured’s internal feature. This provides one point access to configure HTTP status code hard assertion for all API requests send through @Test. If the given status code doesn’t match with response then corresponding @Test gets failed and all corresponding request/response data gets logged into API\_Data.log .
  + To enable this feature add following lines in appConfig.ini under ‘APITesting’ section.

ValidHTTPCodes=200,201,202

* + [Optional] This feature can also be used to analyse API request-response content during exploration and test script building by adding non HTTP status codes like ValidHTTPCodes=1 which will log corresponding API details due to internal assertion failure. But this will mark the test method (@Test) as Fail.
  + [Optional]- This feature is used to display response contents on Report, in order to insert assertions or analysis. So if this key if enabled ('Yes') then it will print more details of API Response on Report. This key is present under aapConfig.ini with name as below. If this

PrintResponseOnReport=Yes.

If this key is set as no “PrintResponseOnReport=No” then it will not print API response on Report.

* UrlEncoding :
* Rest Assured by default does URL encoding for special characters , however if your server doesn’t respond to certain encoded characters you can disable this feature by adding UrlEncodingEnabled = false into **[APITesting]** section of appConfig.ini.
* However there are certain special characters like (“,|,^,<,>,\,# etc.)which are always required to be encoded irrespective of above setting due to limitations of REST Assured and it is encoded explicitly in API plugin utility (com.cybage.apiframework.rest.encoding. UrlEncoding). Usually in such scenario user will have to pass manually such encoded characters into the url/paramaters but here in our plugin we have taken care of this ,User just have to disable this feature and pass the proper URL with any special characters as needed . Plugin will identify its special characters and will encode it by itself .
* Note: To use RestAssured library in testing, Application Server must respond to encoded characters (of “,|,^,<,>,\,#,%,`(tilde), (blank space)) else RESTAssured is not fit for use.

##### API plugin extension

In order to add custom/additional methods to use REST Assured features , make sure to follow below given rules else your customization may get affected with upcoming versions of Fusion framework and plugins.

* + **No** changes/updates should be done to any existing classes or interface other than class RESTDriver and interface IHTTPDriver.
  + Additional classes and utilities can be added , however its reference and use should **not** be made in any existing classes/interface other than RESTDriver and interface IHTTPDriver.
  + Any customization of API driver should be done only in RESTDriver class for implementation and interface IHTTPDriver for its reference and java-doc. Refer this class/interface for sample.
  + RESTDriver class can be further extended if required however pass Report object as an argument to constructor and call super class constructor within extended class constructor.
  + Use try/catch blocks within methods.
  + Within updated method make sure to use logger and reportLogger objects to add info/error messages into logs and reports .
    - (Refer code snippet of RESTDriver)



## DB - PLugin

##### Existing Plugin features and support

CybageDBLibrary plugin is to cater the need of Database testing, primirily, support for RDBMS and Filesystems is added. This library comes with some common ready to use methods which caters around 90-95% need of database testing.

CybageDBLibrary is based on Apache MetaModel 5.0.1version, that provide unique interface to access various database systems.

##### Basic DB library packages, methods and interfaces

DB Library contains below classes and interfaces

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package | Classs/Interface | Extends | Implements | Extends |
| com.cybage.dbFramework | DBContextFactory | DBBaseClass | IConnect |  |
| IDBBaseInterface |  |  |  |  |
|  | IConnect |  |  |  |
| IDBBaseInterface |  |  |  |  |
|  | DbListRow |  | IListRow |  |
|  | FuDataSet |  | IDataSet |  |
|  | FusionDataSetTableModel |  |  |  |
|  | IDataSet |  |  |  |
|  | IDBBaseInterface |  |  |  |
|  | IListRow |  |  |  |
| com.cybage.genericlibrary | CommonFunctions |  |  |  |
|  | DBBaseClass |  |  |  |

**Below is the list of default methods provides as a part of dblibrary.**

|  |  |
| --- | --- |
| Class | Methods/implementation |
| DBContextFactory | public DataContext getConnection(String dbSectionName) |
|  | public DataSet getDataSet(String conn, String sqlquery) |
|  | private DataContext getJsonConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getXmlConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getXlsConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getCSVConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getCassandraConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getMongoConnection(Map<String, String> connectionParamPool) |
|  | private DataContext getJDBCConnection(String dbSectionName) |
|  | private DataContext getCouchDBConnection(String url) |
|  | private DataContext getCassandraConnection(String cassandraHost,int cassandraPort, String keyspaceName) |
|  | public IDataSet getDataSet(String conn, String qry) |
|  | public IListRow getListRows(String conn, String qry) |
|  | public FusionDataSetTableModel getDataSetTableModel(String conn,String sqlQuery) |
| CommonFunctions | public List<String> getListOfDataSources(String testType) |
|  | public synchronized static void buildConnectionObject(IConnect dbDriver,String testType) |
|  | public static Map<String, Object> getConnectionPool() |
|  | public DataSet getSelectedColumnData(DataSet ds1, SelectItem[] selectItems) |
|  | public String getColumnName(DataSet ds, String columnName) |
|  | private Map<String, String> splitToMap(String stringval) |
|  | public Map<String, String> getConnectionParams(String dataSourceName) |
|  | public String getConnectionProperties(String Conn) |

##### DBBaseClass.java

It is not recommended to make any changes in the exisiting libraries. Any further required changes needs to be carried out with extensibility feature. Please do refer DB Plugin Extensibility Section for this.

All the methods in dblibrary are accessible using dbDriver object at test script level.

##### DB plugin Extensibility

To extend exisiting plugin libraries for new implementation user can add those methods by adding them in IConnect interface and then implementing them in respecitive classes.

Below procedure is recommended to get extensibility in DBLibrary

1. Add Try/Catch blocks for each method.
2. Added reporter and logger statements, these objects are available in dbFrameworklibrary classes using inheritance.

* **For Reporter**: User can make a user of below methods as per need

1. ReportLogger.LogFail(String Message) or
2. ReportLogger.LogPass(String Message)

* **For Logger**: User can make a user of below methods as per need

1. logger.Log(LogType type, String Message)
2. To terminate the execution from custom methods, user needs to add below statement.
   1. org.testng.Assert.fail(String Message)

## Mobile –Plugin

##### Existing Plugin features and support

CybageMobileLibrary is a pluin for Mobile tests.

Mobile tests in a Fusion are designed considering Appium (version 1.7.2), following best design practices and standard PageObject pattern using PageFactory.

CybageMobileLibrary gives ready to use methods for Mobile Web interaction and has a provision to extend the library using its IMobileDriver interface and MobileDriver

##### Basic Mobile library methods

Below is the list of default methods:

* public void setup(String deviceName, String url);
* public AppiumDriver getDriver();
* public void setWebDriver(AppiumDriver driver);
* public void navigateTo(String url);
* public void clickOn(Object element);
* public void scrollAndClickOn(Object element);
* public void typeInText(Object element, String text);
* public void getDimensionAndScroll();
* public String getScreenShot();
* public String getTitle();
* public String getPageSource();
* public void quit();
* public void scrolling(ScrollDirection direction);
* public void scrolling(int scrollFrom, int ScrollTo);
* public void buttonclickByexecutor(Object element);
* public void selectDropdown(Object element, DropdownOptionBY val, String passParameter);
* public void pressEnterKey(Object element);

##### BaseMobileDriver.java

It is recommended not to make any changes in this class. Any customization in the existing methods or addition of new method can be done by way of plugin extension. New method can be added in MobileDriver. Refer Mobile plugin extension.

All methods from BaseMobileDriver and MobileDriver can be accessed in test script through IMobileDriver method ones you create an object of MobileDriver in DriverBase.java as below

IMobileDriver mobileDriver = **new** MobileDriver(*ReportLogger*);

mobileDriver.navigateTo(url);

##### Mobile plugin extension

IMobileDriver and MobileDriver can be used for extending the Web Plugin functionalities.

New functionalities can be added in MobileDriver.

Recommendations:

* Add Try/Catch blocks for each method.
* Reporter and logger objects are available in MobileDriver for use though inheritance.
  + ReportLogger - for adding the execution information in the reports.

ReportLogger.LogPass("Opened " + browser + " Browser for navigating to---"

For more reporter methods refer Reporting.

* + logger-

logger.Log(LogType.INFO, "Opened Browser and navigated to---" + url);

For more logger methods refer Logging.

* Driver object from BaseMobileDriver can be used in MobileDriver using IMobileDriver.

mobileDriver.navigateTo(url);

* While creating a new method, if method requires any parameter as WebElement, it is recommended to specify it as Object. And then inside a method do the required conversion as below.

**public** **void** selectCheckBox(Object element) {

**try**{

WebElement ele = (WebElement)

## WebLibrary- Plugin

##### Existing Plugin features and support

CybageWebLibrary is a pluin for WebUI tests.

Web tests in a Fusion are designed considering Selenium 2.0(version 2.53.0), following best design practices and standered PageObject pattern using PageFactory.

CybageWebLibrary gives ready to use methods for Web interation and has a provision to extend the library using its IUIBaseDriver interface and SelUIBaseDriver

##### Basic Web library methods

Below is the list of default methods:

* + public void clickOn(Object element);
  + public void typeInText(Object element, String text);
  + public String takeScreenshot() throws IOException ;
  + public void uiDriverSetup(String url, String os, String browser,String testType);
  + public void navigateTo(String url);
  + public void navigateBack();
  + public void alertAccept();
  + public void alertDismiss();
  + public void pageRefersh();
  + public String getPagetitle();
  + public void selectByVisibleText(Object element,String text);
  + public void selectByIndex(Object element, int index);
  + public void selectByValue(Object element, String value);
  + public void switchToFrame(Object element);
  + public void switchToWindow (Object element);
  + public void waitForElementVisibility(Object element, String timeoutinseconds);
  + public void selectCheckBox(Object element);
  + public String getTextOfElement(Object element);
  + public void selquit();
  + public WebDriver getDriver();

##### SelUIBaseDriver.java

It is recommended not to make any changes in this class. Any customization in the existing methods or addition of new method can be done by way of plugin extension. New method can be added in SelUIDriver. [Refer WebUI plugin extension](#_WebUI_plugin_extension).

All methods from SelUIBaseDriver and SelUIDriver can be accessed in test script through uiDriver() method ones you create an object of SelUIDriver in DriverBase.java as below

IUIDriver selUIDriver = **new** SelUIDriver(*ReportLogger*);

uiDriver().pageRefersh();

##### WebUI plugin extension

IUIDriver and SelUIDriver can be used for extending the Web Plugin functionalities.

New functionalities can be added in SelUIDriver.

Recommendations:

* Add Try/Catch blocks for each method.
* ReportLogger and logger objects are available in SelUIDriver for use though inheritance.
  + ReportLogger - for adding the execution information in the reports.

ReportLogger.LogPass("Opened " + browser + " Browser for navigating to---"

For more ReportLogger methods refer Reporting.

* + logger-

logger.Log(LogType.INFO, "Opened Browser and navigated to---" + url);

For more logger methods refer Logging.

* Driver object from SelUIBaseDriver can be used in SelUIDriver using getDriver().

getDriver().navigate().to(url);

* While creating a new method, if method requires any parameter as WebElement, it is recommended to specify it as Object. And then inside a method do the required conversion as below.

**public** **void** selectCheckBox(Object element) {

**try**{

WebElement ele = (WebElement) element;

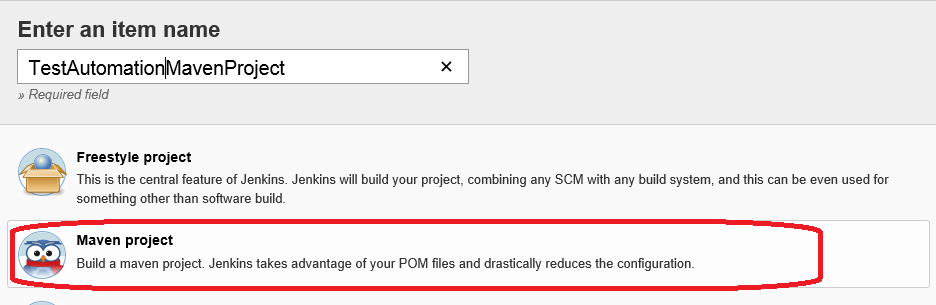
* IEDriver and Chrome driver exe are placed at [\\AutomationTestProject\setupfiles](file://AutomationTestProject/setupfiles) folder.

If you required to update the version of those exes, you can directly replace the files at above place keeping the file name same.

# Jenkins CI

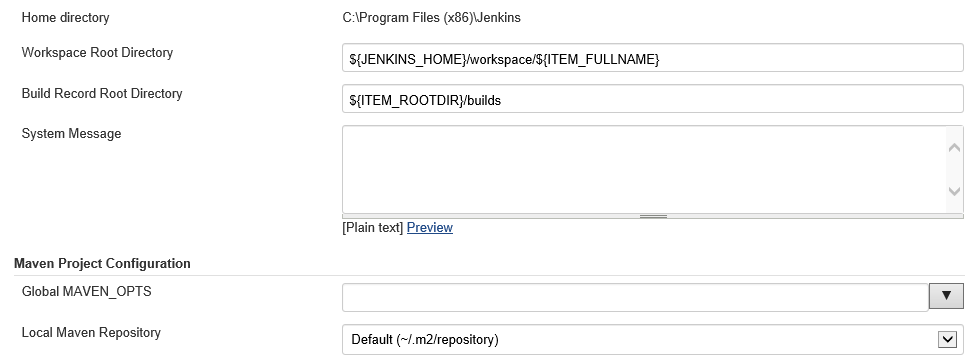
**Following are the steps to setup Jenkins for Fusion:**

1. Create a new item Maven Project in Jenkins.

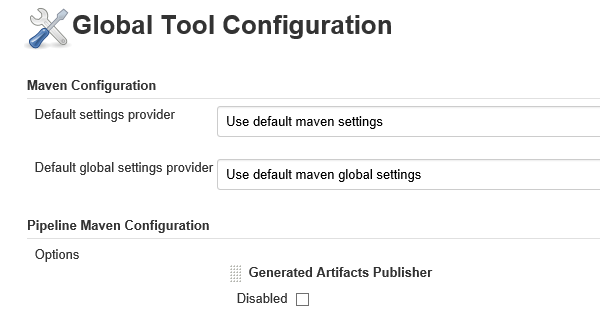


After this Build your job ,so that Jenkins would create the workspace. After creating Jenkins job workspace, please refer section 7.3Execute a Test from Jenkins

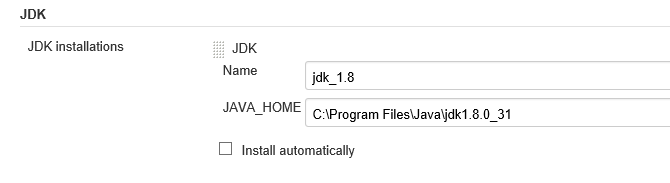
1. Go to Manage Jenkins 🡪 Configure System Set the Workspace and Maven Project configuration as below:



1. Go to Global Tool Configuration Set the Maven configuration as below:



**JDK Installations:**



1. CSS is striped because of the Content Security Policy in Jenkins. To relax this rule:
2. Go to Manage Jenkins->Script console and type in the following command:

System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "")

1. and Press Run.

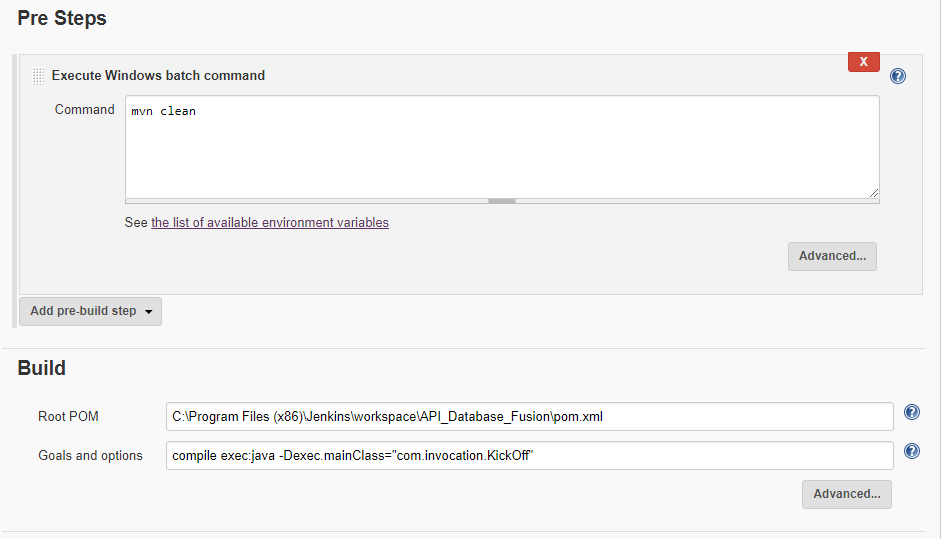
OR

* 1. Go to C:\Program Files(x86)\ Jenkins\jenkins.xml
  2. Add following entry -Dhudson.model.DirectoryBrowserSupport.CSP in the arguments section.

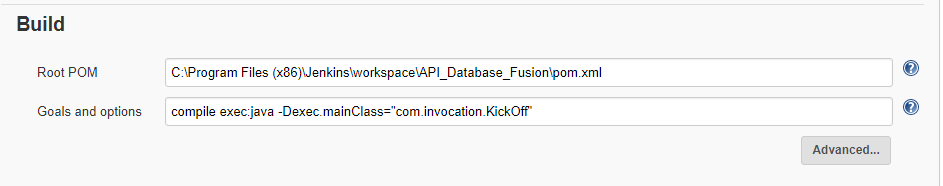
E.g. <arguments>-Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -Dhudson.model.DirectoryBrowserSupport.CSP -jar "%BASE%\jenkins.war" --httpPort=8080 --webroot="%BASE%\war"</arguments>

By doing this you will be able to view any HTML report in Jenkins.

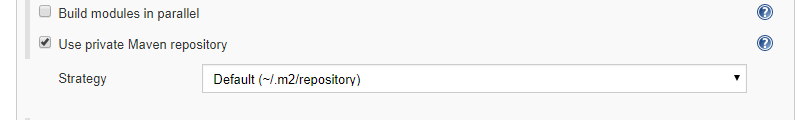
1. Go to Jenkins 🡪 TestAutomation Maven project.
2. Add mvn clean in Pre Build steps



1. Build Execution details-Root POM path will be your Automation test project pom path and Goals and options will be as compile exec:java -Dexec.mainClass="com.invocation.KickOff"



Click advanced under Build section and under that select Use private Maven repository as below



1. **Add Post Steps:**

Note: This below mentioned post steps are only for Windows Jenkins configuration.For Linux-Ubuntu separate shell script need to be added to copy LOG and Report folder from Execution Run name folder to newly created destination folder.

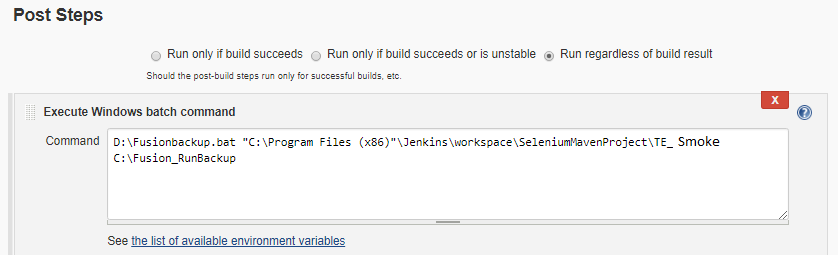
**Purpose:**

This will be used to take backup of Fusion Test Automation execution reports from Jenkins.

It will execute the Fusionbackup.bat file. This needs two command line input parameters SOURCE and DESTINATION path separated by space.

NOTE: Here **TE\_Smoke** is the Execution Run name given on the Test Runner Execution UI.

In case you change the execution run you need to update the path with the new execution run name.

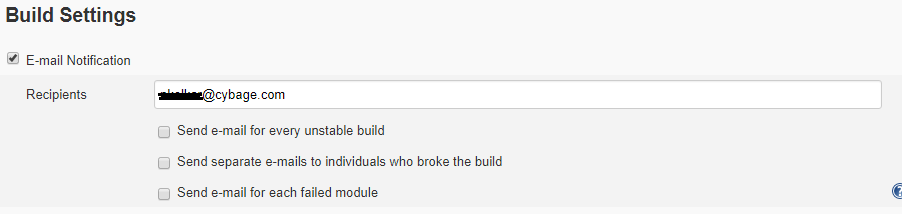


**Sample command** : C:\Fusionbackup.bat "C:\Program Files (x86)"\Jenkins\workspace\SeleniumMavenProject\TE\_Smoke C:\Fusion\_RunBackup

In the above fig what each path specifies is as below

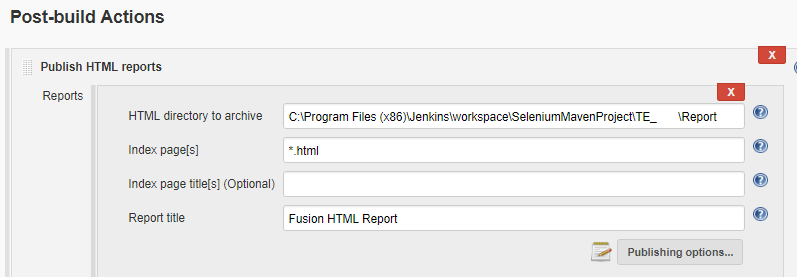
* + - C:\Fusionbackup.bat- This is path where your batch file is stored
    - "C:\Program Files (x86)"\Jenkins\workspace\SeleniumMavenProject\TE\_Smoke- This complete path is of your Jenkins project and “TE\_Smoke” is the folder created for your Run and has configuration files reside. Please Note: This folder name will be new for each Run as you create it from Execution UI, so you will have to modify your Post steps if you are going to run the tests through Jenkins.
    - C:\Fusion\_RunBackup- “Fusion\_RunBackup” is the folder created to store the backup.

**Build Settings:** In the build setting, check the email notification

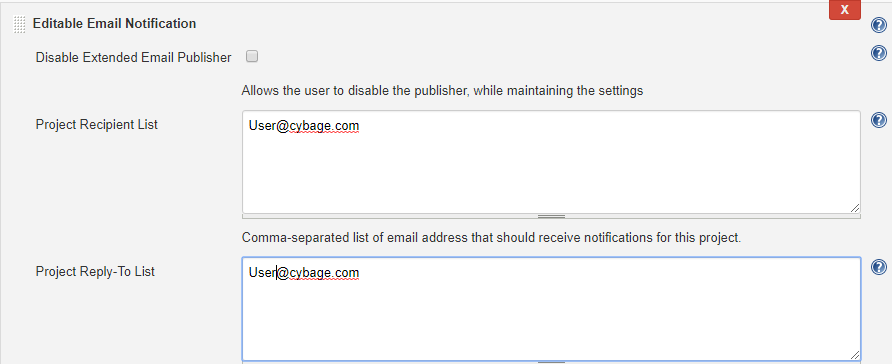
****

1. **Post Build Actions:**

This will be capture the Fusion Report and publish it in Jenkins build wise.



1. **Editable Email Notifications:** Add this job in the post build



1. Click the ‘Add post-build action’ drop-down and select the ‘Editable Email Notification’ value.
2. Fill the ‘Editable Email Notification’ fields.
3. Click the ‘Advance Settings…’ button in the ‘Editable Email Notification’ box.
4. Click the ‘Add Trigger’ drop-down and select the ‘Always’ option.
5. Click the ‘Save’ button.

**Configure Email in Jenkins:**

1. **Add following plugins From Manage Jenkins**
   * 1. [Email Extension Plugin](http://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin)
     2. [Email Extension Template Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+Template+Plugin)
2. **Go to Manage Jenkins 🡪** 
   1. Go to the Jenkins home page and click the ‘Manage Jenkins’ menu option. Then, select the ‘Configure System’ option.
   2. Enter the SMTP server name under ‘Email Notification’. Click the ‘Advanced’ button and then click the checkbox next to the ‘Use SMTP Authentication’ option. Now, set the following fields.
   * **SMTP server name** : smtp.gmail.com
   * **User name**: user\_email\_id@gmail.com
   * **Password**: 123456
   * **Use SSL**: Checked
   * **SMTP Port**: 456
   1. Check the email notification functionality by clicking the checkbox next to the ‘Test configuration by sending Test e-mail recipient’ option. Enter a valid email id and click the ‘Test configuration’ button to check whether the email id is valid or not.

# Add-ONS

There are various add-ons or plugins available from Cybage Fusion Team in addition to what Fusion provide . This add-ons will be provided on project need basis. Add-ons are like enablers which further accelerates automation. In addition to below given add-ons there are other utilities which can be directly used with Fusion and are tested for its integration and compatibility like [Wiremock](http://wiremock.org) server for API mocking, [SauceLabs](https://saucelabs.com) cloud platform for web application testing , [Klov](http://extentreports.com/docs/klov/) analytics server for test result analysis over past executions etc.

## POM Generator

<To be added>

## API Scripter

This add-on will auto generate API test scripts in few seconds which will be fusion compatible and ready to use. This is applicable if you have REST APIs with Swagger (OpenAPI) specification in place. It will create test scripts for each API resource/method as defined in Swagger. This add-on will also autogenerate necessary API resource details , test data excel templates against each API resource which can be directly copied within Fusion framework. For more details on its usage refer README provided with add-on.

## ZAPI Integrator

This enabler will provide direct integration with Zephyr-JIRA add-on to update corresponding test results directly in test management tool Zephyr. With necessary configuration like marking TestID annotations in Fusion scripts with resect to Zephyr test case ids , results will get uploaded in it for every test execution. More details will be provided with enabler when requested.

# Limitations

## Web

* Test execution on Mac machines is not supported.
* Hub node setup through fusion framework on Linux –Ubuntu machine is currently not present.
  + Workaround- User needs to setup manual grid and pass execution to that grid through Existing hub:
* Cannot run web tests on Jenkins server locally with Firefox driver.

This is a known issue with Gecko Driver

<https://github.com/mozilla/geckodriver/issues/1068>

Workaround: Create a hub and remote machine and then execute tests on Firefox browser.

## Mobile:

* No. of devices connection support will be as per Appium support
* Automation of Native or Hybrid app is not supported.

## API:

* To use RestAssured library in testing, Application Server must respond to encoded values of given special characters( “,|,^,<,>,\,#,%,`(tilde), (blank space)) else RESTAssured is not fit for use. For more details refer section8.1.3 UrlEncoding.
* If you are running GraphQL tests having Mutation operations like create, update, delete nodes then select ParallelRun as “NO”. This is because GraphQL doesn’t support Mutation operations to run in parallel. But GraphQL tests having only Query operations like read or retrieve can be performed in parallel. This limitation is from the GraphQL side. This is not the limitation of Fusion Framework.

## Fusion TestRunner

* While running Web test cases in Hub and Node mode through FusionTestRunner on Fusion machine, hub node establishment may have problem because of which test cases may fail.
  + Workaround: While running Web test cases in Hub and Node mode through FusionTestRunner use FusionTestRunner from different machine other than the machine on which Fusion is installed or run the web test cases in hub node mode through Jenkins jobs.
  + If you still want to run Web test cases in Hub and Node mode through FusionTestRunner on which Fusion is installed restart the machine after each run or carry out steps mentioned in the link below

<http://growlycode.blogspot.in/2011/05/how-to-fix-specified-service-has-been.html>

## DB

NA

# Known issues

## TestNG v 6.11

* When multiple fusion suites/ test types are executed then for some tests duplicate exceptions will be seen. This is TestNG specific bug as described [here](https://github.com/cbeust/testng/issues/956) and should be stabilized once this issue is fixed from TestNG team.
* When a configuration method fails due to some run time exceptions then browsers may remain opened and won’t get closed . This is because in such case @AfterMethod is not being called and is a TestNG specific bug as described [here](https://github.com/cbeust/testng/issues/1465).

# Glossary of Terms

|  |  |
| --- | --- |
| Server | A Execution machine on which you download your source code from repository |
| Client | A machine from which you can trigger execution |
| Hub | The hub is the central point where you load your tests into |
| Node | A node is a machine that you register with your hub for executing tests |
| Plugin Library | The plugin includes ready to use libraries for Web, API, DB and Mobile |
| Core Framework | It contains the framework’s execution core |

# FAQs

In case of any queries you can contact Fusion DL.

# Contact Us

In case of any queries you can contact Fusion DL.