Enterprise Automation Team

Humana, Inc.

Selenium CSharp Framework

APPENDIX:

[Overview of Framework](#_Overview_of_Framework:)

[Main Components of Framework](#_Main_Components_of_1)

[Framework Setup in Our Project](#_Sample_Project_–)

[Common Function Libraries](#_Common_Function_Libraries:)

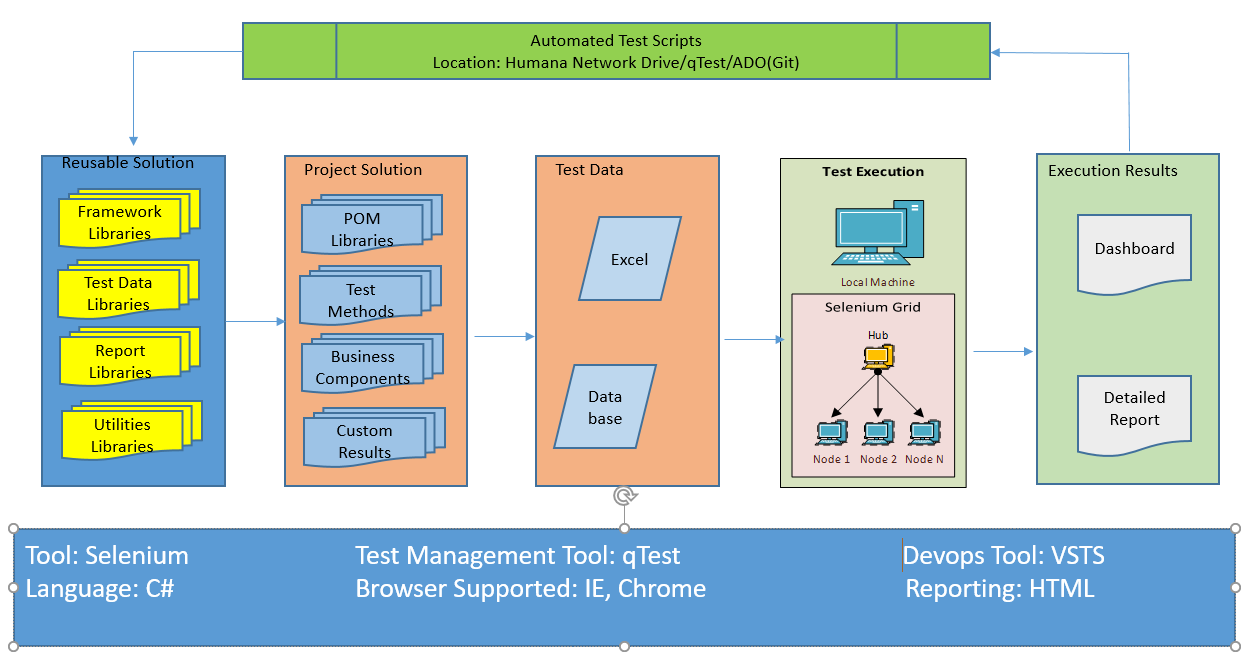
[Selenium Grid](#_Selenium_Grid)

[Integration with qTest and CI/CD:](#_Integration_with_qTest)

[Headless Automation](#_Headless_Automation)

# Overview of Framework:

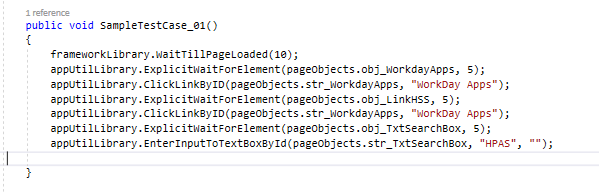
This framework was implemented using .Net Framework (Unit Test Project) and is configured to perform unit testing. There are six primary code components of this framework and they are the **Business Components**, **Sample Tests**, **Driver Servers**, **Test Data**, **Common Function Libraries** and **APP Config File**. A detailed description of each of these components will be given in the proceeding sections of this document. A brief description of each component is given below.



**Business Components:**

Under Business Components Folder we have to create two class file (Business Library, Page Objects). In Business Library Class file we design scripts for all test cases according to our Functionality.

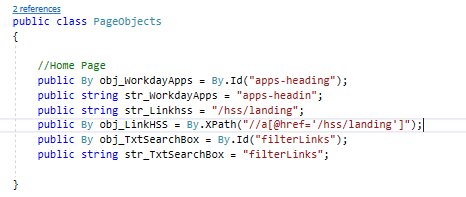
**Business Library Class File**



**Page Objects Class File**

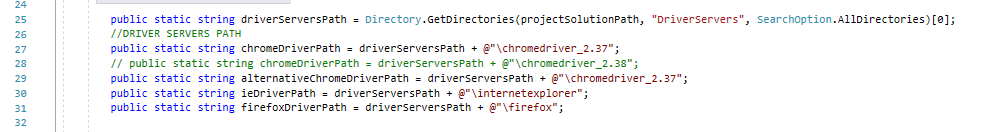
Page Object Model, also known as POM, is a design pattern in Selenium which has gained more popularity in the market for test automation development for the maintenance of code, such as reusability, extensibility, and avoiding code duplication etc. Page Object Model is used for creating Control properties or Object Repository for controls on a webpage. For each webpage which we want to automate there should be a separate class such as if we are performing the automation for the login page, we need to maintain all the login page control properties in the separated class file. If we consider a login page, there are controls available in a login page such as Username, Password, Login, Forget Password etc. Each control will have unique control properties such as Name, Control ID, Tag Name, XPath, Class Name, Css Selector etc.

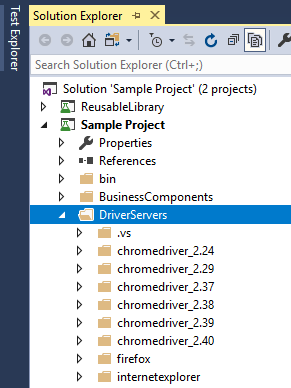
Each procedure script in business library class file refer the Page Objects in its corresponding Page Objects class file. In Page Object Class File all web elements or web objects are stored here for reusable purpose and Easy updates in script.



**Driver Servers:**

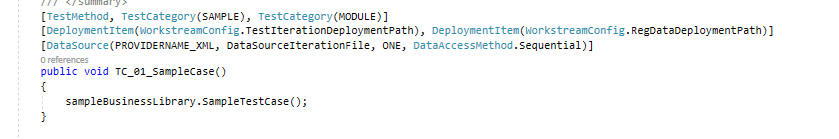
In Driver Server Folder all type of browsers like (IE, Chrome and Firefox), .exe files and their versions are stored as per the requirement. For each test run the driver initialization take the path from the driver server path is mentioned in Run Configuration Class file as static variable.





**Sample Tests:**

Under Sample Tests Folder we have to create one class file (Sample). In the Sample Class file we have the test cases for design scripts in business library class file. Using object Creation we call the object and Functionality according to our test case functionality and each test cases are handled here based on the method categorization.

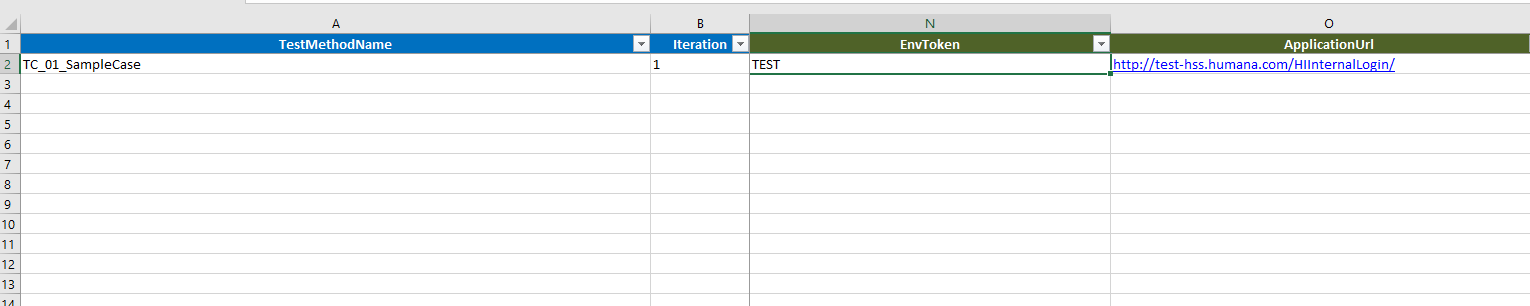


**Test Data:**

For each test case execution the script will fetch the data/information from the Excel spreadsheet (.xlsx). Each test case must specify the test method name, iteration, test summary description, test validation description and test input data for each test case. The Test Data Filename is mentioned in APP Config File in Key Value Pair Combination. In this sheet, we have two sheets named **General data sheet**, **Environment specific data sheet**.

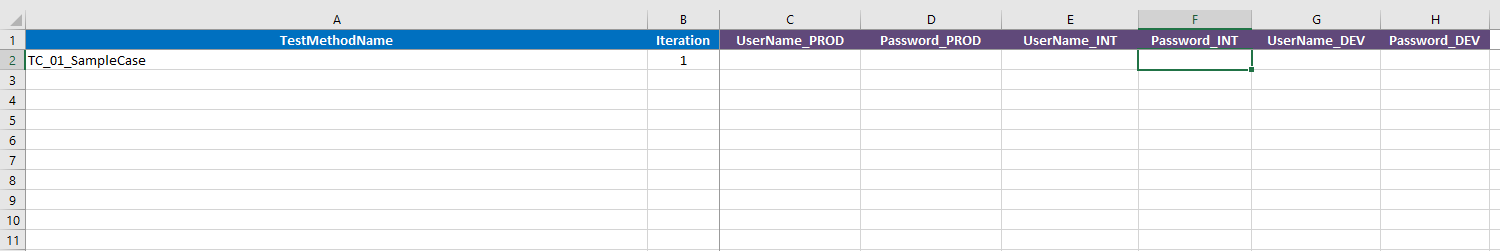
**General data sheet:**

For each test case, the test data like test method name, iteration, Env Token, Application url, etc… are maintained in the sheet.



**Environment specific data sheet:**

For each test case, the login credentials are divided based on the environment tokens are fetched from the sheet.



**Support Libraries:**

The support library contains five class file used by all script components. This library must include at least the following five class file: **Report Factory**, **Send Mail**, **Table Structure**, **Test Data Factory** and **Test DB Data Factory**.

**Report Factory:** The class file holds the generation of results/reports in html format. The result template is designed as per requirement.

**Table Structure:** In the class file the field values required to be displayed in the Report Generated (HTML Report) are coded.

**Test Data Factory:** The test data in .xlsx document is converted as query to fetch the data easily. The code to fetch the data is written in this test data factory class file.

**Test DB Data Factory:** Code for fetching data from database server through query and passing to the script is designed here. DB connectivity is used here for hooking the database server to the script.

**Common Function Libraries:**

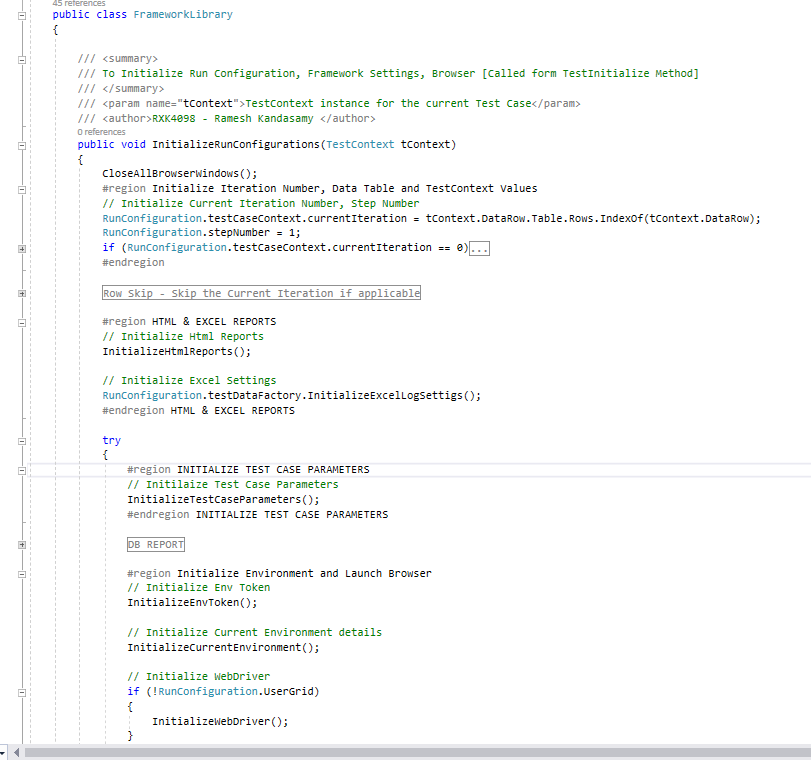
This common function library contains generic functions that are utilized by Business Components Library and the Sample Tests. The majority of user interface object processing occurs within the functions of this library. A common function library contains at least one test case for each user interface component stored in its corresponding Sample Tests. When a code in test case is executed a particular action is performed on the related user interface component. These range from simple actions such as clicking a button or entering text into a text box and validating page within a webpage.

# Main Components of Framework:

The main functions/components which takes a vital part in test case execution are, **InitializeRunConfigurations**, **InitializeDataTable**, **GetTestData**, **WrapUpTestExecution,**  **UpdateTestLog and Test Context**. Below are the brief information for the main components:

**Initialize Run Configurations:**

When each test runs, first its calls the base method **Initialize Run Configurations** present in framework library. The **Initialize Run Configurations** consists of the following default methods used by every tests.



1. **DetermineValuesFromAppConfig**

The lines of code written in this method reads the value from APP Config File. Like which “**DetermineValuesFromAppConfig**” is the key and “**APP Config**” has the value in it. In App configuration file, the execution details like

1. **App settings** (batch run id, release name, test planned, current run category, environment token) Before Running the Test case we need to set the **BatchRunID** in the **App.config** file, the new folder will be created automatically with name as **BatchRunID** value and result will be store under this folder once execution started.
   1. **<add key="BatchRunID" value="SampleProject\_01" />**
2. **HTML** **reporting** (Result folder path -HTML) In App.config file - Change the Html Report Path Key value Local Copied File Location \C:\source\repos\SampleProject\Test ExecutionResults\HTMLReports
3. **ScreenShot** Capturing In App.config file - Change the ScreenShotFolder Keyvalue
   1. LocalCopiedFileLocation\C:\source\repos\SampleProject\TestExecutionResults\ScreenShots
4. **Test data** **file info** (test data file name)
5. **Current browser** (Chrome, IE, Firefox etc., as per requirement)

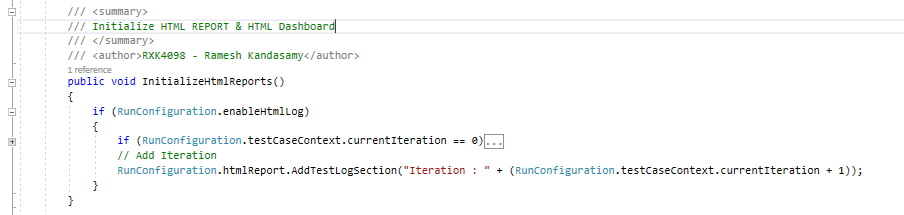


1. **InitializeTestData**

This method is used to pass input values through the test data file (excel spreadsheet) and it has a method called “**InitializeDatatable**” that executes a query to store input data in the form of a Data table. Later, the brief details of Initialize Data table are explained below.

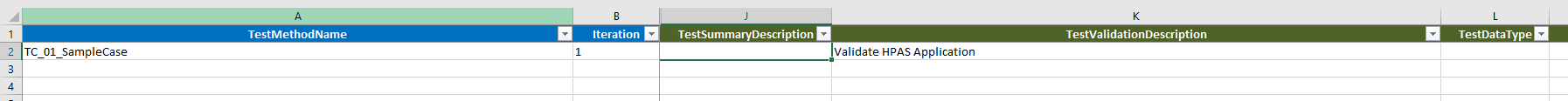
1. **InitializeHtmlReports**

The path where the html reports are stored, result logs, screenshots are all defined in this method. The code to view the screenshot of the test result is written in this method where we can view the result by clicking the hyperlink in html reports.



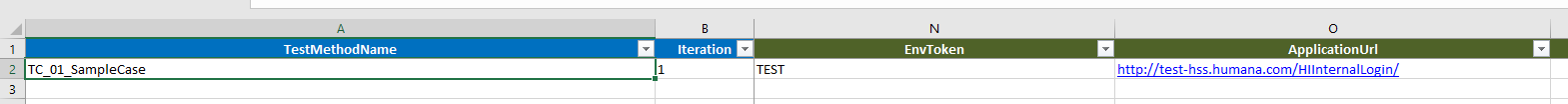
1. **InitializeTestCaseParameters**

The test case information like test case category, test case description, test data, test summary description are passed as parameters through the input data sheet. So that these information are displayed in HTML reports.



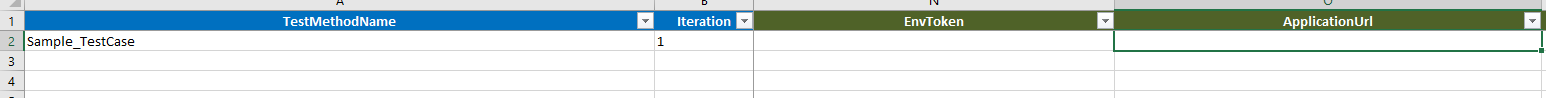
1. **InitializeEnvToken**

The environment in which the tests are to be executed is mentioned in this method. There is a column called “**EnvToken**” in test data spreadsheet (**General\_DataSheet**) where we can specify the test environment. (**Eg**: **TEST**, **PROD** and **QA**)

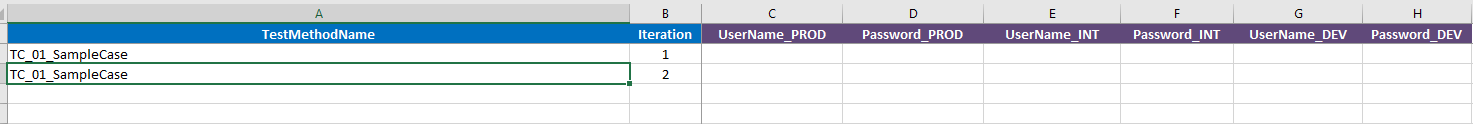


1. **InitializeCurrentEnvironment**

This method consists of the code to pass the application URL. There is a column called “**ApplicationUrl**” in test data spreadsheet (**General\_DataSheet**) where we can specify the URL.

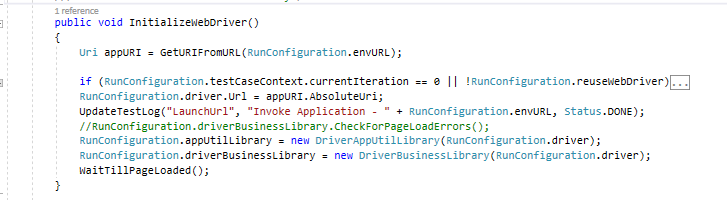


For each test case, the login credentials are divided based on the environment tokens are fetched from the **EnvSpecific\_DataSheet**.



1. **InitializeWebDriver**

Web drivers like chrome, Firefox and IE are initialized in this method. The driver paths are stored as static variable in the Run Configuration class file.

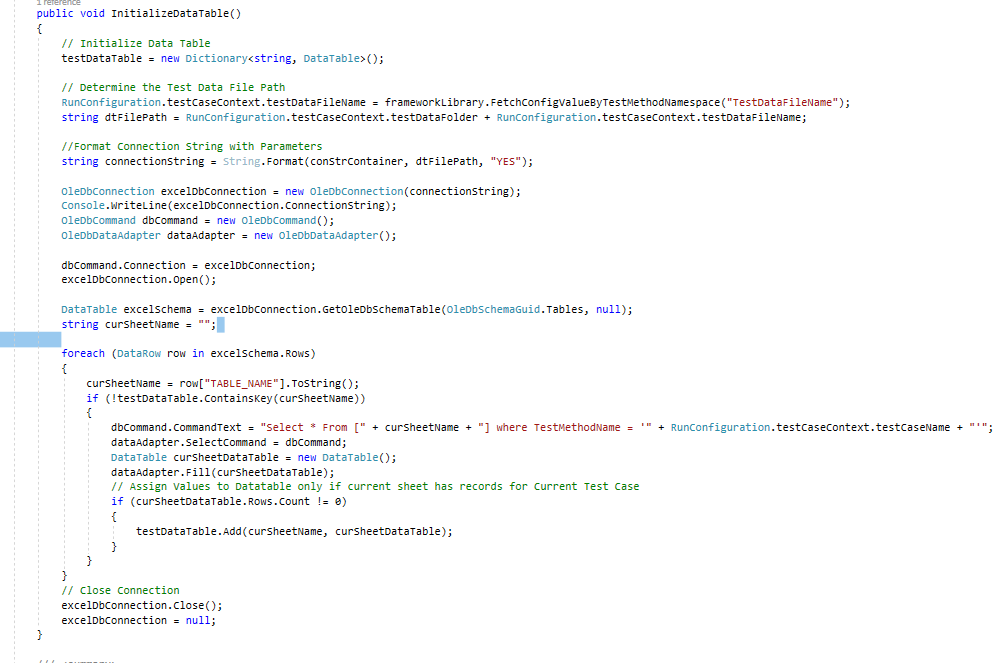


**InitializeDataTable:**

This is the method to Read Test Data from Data Table. The Data table will have data specific to current Test Case and will be initialized from Initialize Method.

The data needed for each test case is read from the data sheet. Then a query is written to fetch all the input rows and columns from input sheets (**General\_DataSheet** and **EnvSpecific\_DataSheet**) to retrieve the values and is stored to a data table.

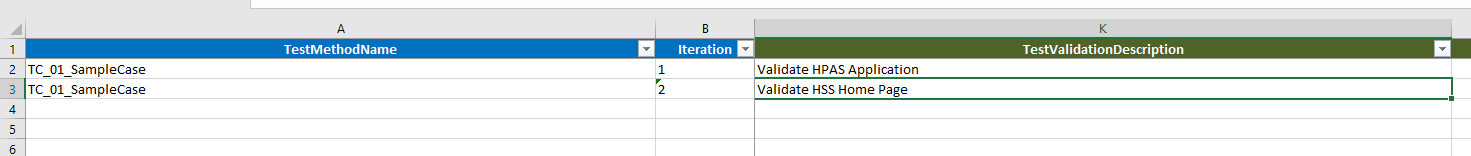
Each time a test case executes, the input value passed to the application is fetched from the value stored in data table.



**GetTestData:**

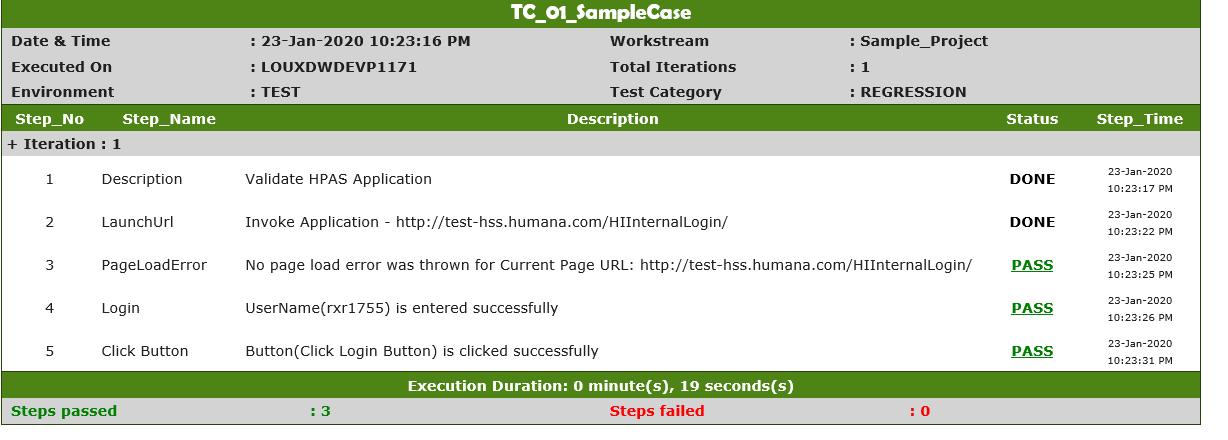
On executing each test case, the data is fetched from input datasheet and passed to the application. But when we want to run the same test case with different data, we can use “iteration” to execute the test case.

When Iteration is set to “1” in input sheet, the test case executes with data provided and we can set the iteration to “2” to run the same case with different input data’s.



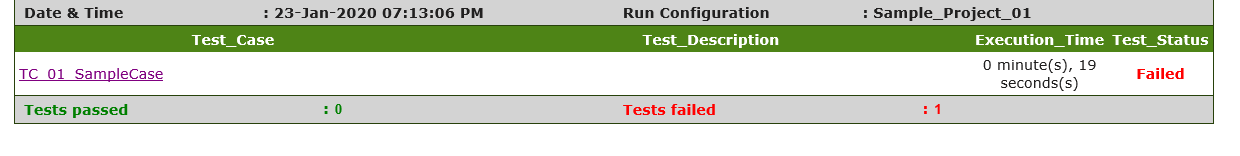
**WrapupTest Execution:**

After each test case run the detailed information of complete test case execution, the total test steps passed, the total test steps failed, execution duration of the test case and all displayed in the HTML Report Format. These lines of code are written in wrap up test execution method in framework library class file.



**UpdateTestLog:**

The code to initialize the HTML reports like how the test logs are updated step by step, where the status of the test results are present as hyperlinks and the screenshot corresponding to the FAIL and PASS status are all defined in the “**UpdateTestLog**” method in framework library.



**Test Context:**

Gets or sets the test context which provides information about the functionality for the current test run. Test Context holds the information of the test case name, method and functions defined in the input datasheet.

**TestcaseContext** is a class present in **RunConfiguration** class file which is defined as an object in **Test Context**. In this class variables are declared static, so that is helps to hold the objects until the test case runs.

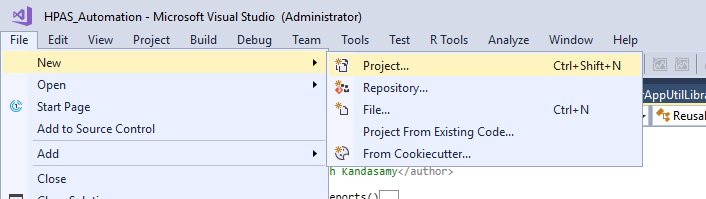
# Sample Project – Setup Guidelines

Get Visual Studio installed in your Local Machine through the following steps:

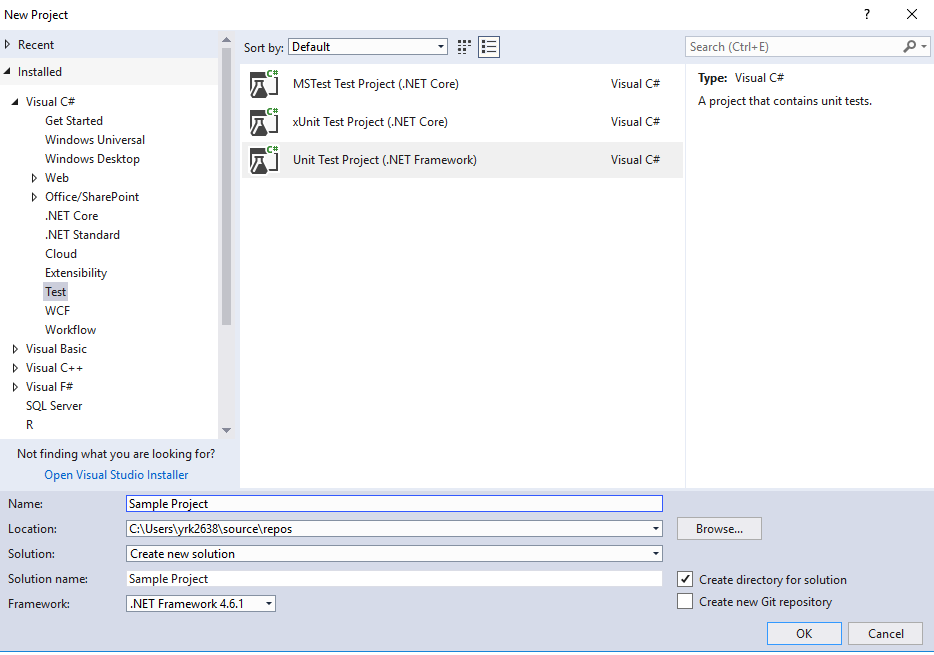
1. Raise a request for “Visual studio 2017 or Latest” in “http://servicecatalog.humana.com/sc/catalog.product.aspx?product%5Fid=hum\_sftwr1e/”.
2. After this Visual Studio will be installed in your local machine.
3. Open the Microsoft visual studio installed in your machine.



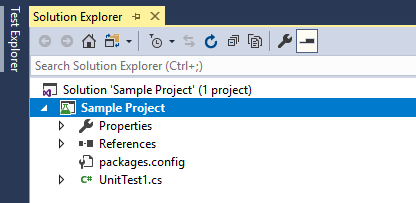
* 1. **Click File 🡪 New🡪 Project/Solutions**.



Then Click🡪Visual C#--> Test-> Unit Test Project (.Net Framework) then Give Project Name in the Name Column and then Click Ok.



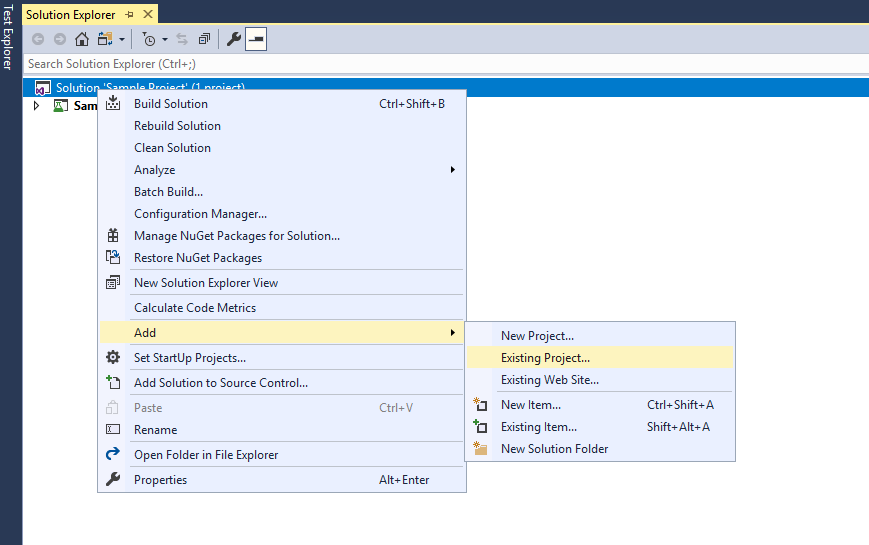
A New Project solution is created.



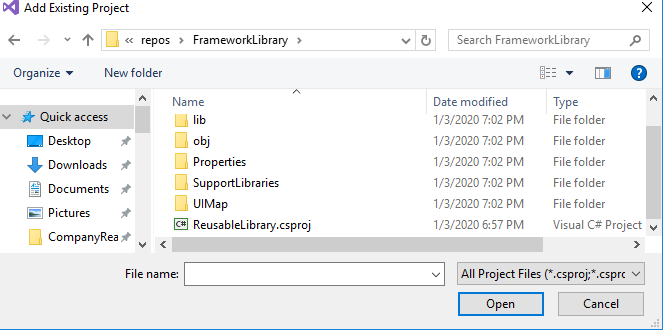
Then Copy the Framework Library, Test Execution Results and Test Results Folders From the “**Sample Project”** and Paste it to the New Project Folder present in the path (Local).

Follow the below steps to add the “reusable library “in Visual studio.

1. Just Right Click on Solution🡪 Add🡪 Existing Project🡪 Dialog box appears.

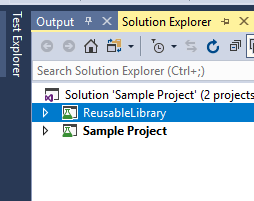


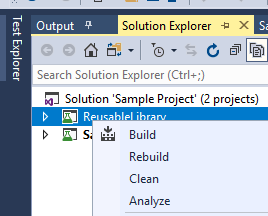
1. Open “**FrameworkLibrary**” folder in the path where we added Framework Library, Test Execution Results and Test Results folders in the created project.

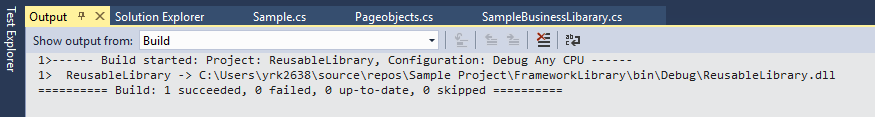


1. Select “**ReusableLibrary.csproj**”. By doing this reusable library will be added to the project Solution Explorer.

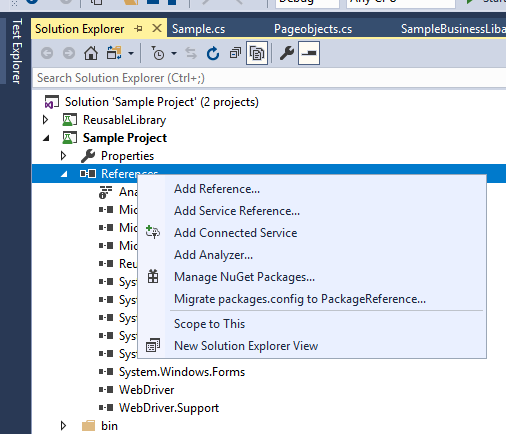
Now we have two projects, the project that is created new and the reusable library.

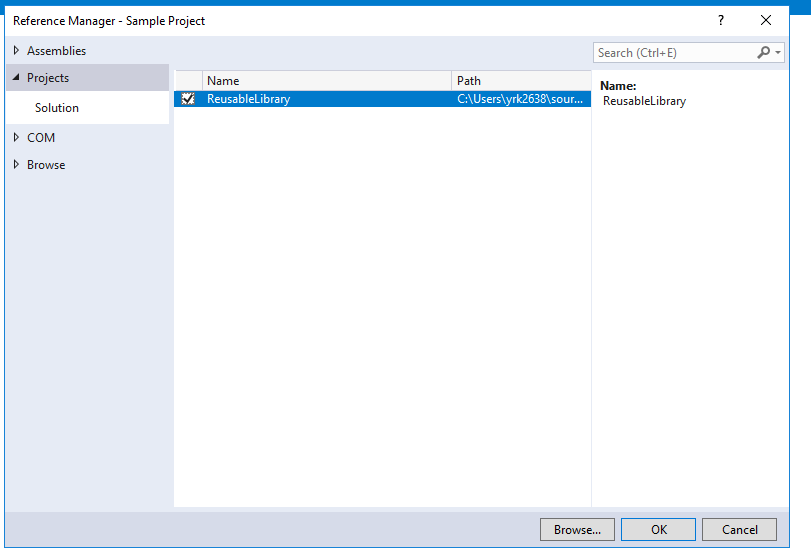


1. Right click the Reusable Library Project and select the Build option.
2. 
3. After Build Succeed Message.

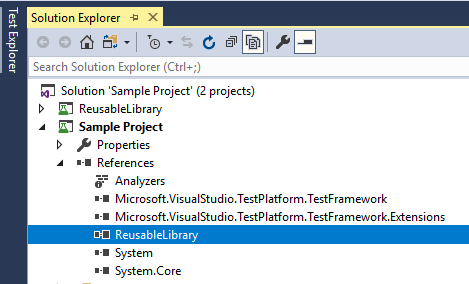


1. Go to Sample Project Click “**Reference**” 🡪 Right click on the reference 🡪 Add reference



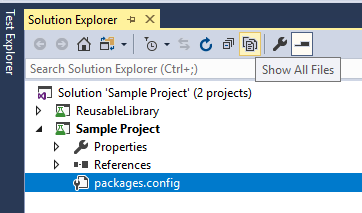
1. On clicking “**Add Reference**”, a Dialog Box called “**Reference Manager - Sample Project**” appears. Select Projects 🡪 Solution and check the “**Reusable library**” present in the window. Then click “**OK**”.

By doing this Reusable Library will be referenced to the Sample project created.

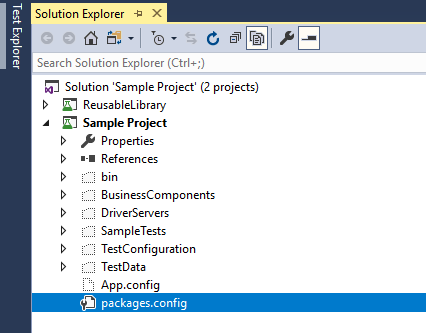


Then include Bin, BusinessComponents, DriverServers, Properties, SampleTests, TestConfiguration, TestData, App, packages by following the below steps:

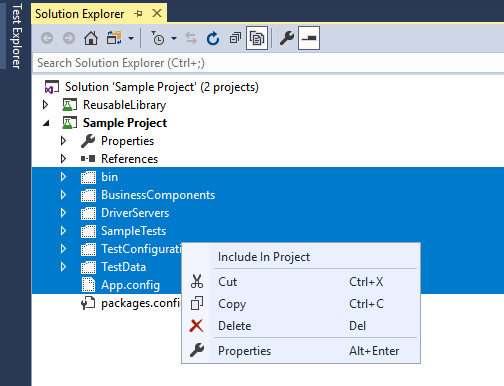
1. Copy the above mentioned folders from “Sample Project” and paste those folders to newly created project.
2. Click “Show All Files” icon in Solution Explorer



1. The included folders will be listed under the project as shown below



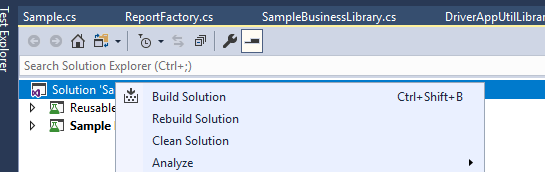
1. Select all the folders 🡪 right click 🡪 Click “Include in project”. By this all the necessary folders will be added.



After including folders in Project rename the class files present in each folders. Rename “**BusinessComponents**” and “**SampleTests**” class files according to the created project. Rename the excel spreadsheet in **“TestData”** folder respectively.

**Customize the “App.config” file according to the project information provided. That is, modify the BatchRunID, ReleaseName, TestsPlanned,** **CurrentRunCategory, HtmlReportPath, TestDataFileName, ScreenShotFolder according to your project in App.config file.**

After completing the whole project setup, click on “**Solution explorer**” 🡪 Right click on **Solution** 🡪 **Clean Solution.**



Once clean project is done, you will get a success message and we can build the project.

“**Solution explorer**” 🡪 Right click on **Solution** 🡪 **Build Solution**. Now the project is set up to execute test cases.

# Common Function Libraries:

The common functions library is where the majority of user interface object processing occurs. Currently, there is one common function library developed, this library is for web applications. This common function library contains functions that can be utilized by the Business Components Class file. The functions within this library should be generic; this ensures any procedure within the application can utilize the functions. To maintain consistency across all application automation frameworks regardless of the development technology, every common function library must include the following functions. Reusability of script code is extremely important because it decreases code redundancy and reduces maintenance efforts. The following sections of the document review each function within the existing common function library for web applications.

There are 2 common functions class files in Reusable Project. They are **Driver Business Library**, and **Driver App Util Library**.

**Driver Business Library:**

Business functionalities such as login validation (website loaded or not), login inputs (passing username, password) and page load errors (Server error, Page availability) of any application is written and handled in class file.

**EnterUserNamePassword(username, username)**

**Purpose:**

To Login to any application, we should provide user’s credentials .Using this function testers do not have to specify their username and password within the test case input file. All usernames and passwords are stored in an input data sheet (excel spreadsheet).   
**Functionality**

Loads the username and password corresponding to the test case specified as input from input data sheet (excel spreadsheet).

**Input**

username, password

**Return Value**

None

**CheckForPageLoadErrors(pageDesc,pageUrl)**

**Purpose:**

In this function checking the application url is loaded correct or not, any errors like 404 Server Error and 500 Server Error.

**Functionality**

To check for errors while loading the page like page not available, blank page etc…

**Input**

pageurl, pageDesc

**Return Value**

Booelan

**NavigateToTargetURL**

**Purpose:**

In this function, we are performing some actions to the current url after the actions performed, the current url navigates to the another url (input).

**Functionality**

To navigate to target URL and validate target page header which is optional

**Input**

Here will give the target url in input data sheet.

**Return Value**

None

**ValidateApplicationLoginSuccessful**

**Purpose:**

The purpose of the function is to verify the login functionality of the application.

**Functionality**

To Validate Login is successful for Application.

**Return Value**

None

**ValidatePageUrl**(**expectedUrl**)

**Purpose:**

The purpose of this function is to verify the application URL is launched and navigated to the respective web page.

**Functionality**

|  |
| --- |
| To Validate Page URL. |

**Return Value**

None

**Driver App Utility Library:**

The common actions to be performed in the applications like Click action, Enter Inputs to textbox, Dropdown action, Mouse actions, Checkbox, capturing screenshot etc., are designed here for reusability and code efficiency.

By using these methods we would be able to view the message in the html report along with function (For Eg: **ClickButtonByName**) being performed.

Functions and Method Name with return type of the C# Framework are listed in the below spreadsheet document.



# Selenium Grid

In our framework, we have used Selenium Grid to execute the tests in parallel. The “**LaunchGrid**” functions in “**SampleBusinessLibrary**” consists of the grid configuration. Selenium Grid is a proxy server that makes it easy to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Hub conducts a concurrent execution of tests on multiple machines, managing different browsers centrally, instead of conducting different tests for each of them. Selenium Grid makes cross browser testing easy as a single test can be carried on multiple machines and browsers, all together, making it easy to analyze and compare the results.

The two major components of Selenium Grid are:

* **Hub** is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
* **Node** is a remote device that consists of a native OS and a remote WebDriver. It receives requests from the hub in the form of JSON test commands and executes them using WebDriver.

Here are the steps to follow for setting up Selenium Grid:–

**Step 1: Installation**

Before getting started, [download the Selenium Server Standalone package](https://www.seleniumhq.org/download/). This package is a jar file, which includes the Hub, WebDriver and legacy RC that is needed to run the Grid. To get started with Selenium Grid, it is essential to have Java already installed, and set up the environment variables.

**Step 2: Start Hub**

Hub is the central point in the Selenium Grid that routes the JSON test commands to the nodes. It receives test requests from the client and routes it to the required nodes. To set up the Hub, open the command prompt, and navigate to the directory, where the Selenium Server Standalone jar file is stored (downloaded in Step 1)

Type the following command

**“java -jar selenium-server-standalone-<version>.jar -role hub”**

This would start the hub automatically using port 4444 by default. Testers can change the default port by adding an optional parameter port, using while running the command.

**“-host <IP | hostname>”**

Testers need not specify the hostname as it can be automatically determined until you are using an exotic network configuration or networking with VPN, where specifying the host becomes necessary.

To view the status of the hub, open a browser window and navigate to **“**[**https://localhost:4444/grid/console**](https://localhost:4444/grid/console)**”**

**Step 3: Start Nodes**

Whether testers are looking to running a grid with new WebDriver functionality or with the Selenium 1 RC functionality or running both of them simultaneously, testers have to use the same Selenium Server Standalone jar file, to start the nodes. To start nodes open the command prompt and navigate to the directory, where the Selenium Server Standalone jar file is stored.

Type the following command

“**java -jar selenium-server-standalone-<version>.jar -role node –hub**” “**https://localhost:4444/grid/register**”

When -role option that is provided is not specified, and it is not the hub, the default port is 5555. So, it is important to define the -role to be a node in this case.

**Step 4: Configure Nodes**

When testers start the nodes, by default, it allows 11 browsers, i.e., 5 Firefox, 5 Chrome, and 1 Internet Explorer for concurrent use. It also allows testers to conduct a maximum of 5 concurrent tests by default.

But in Our Framework we use only Internet Explorer and Chrome.

Testers can change this and other browser settings, by configuring nodes. This can be done by passing parameters to each of the -browser switches that represent a node, based on the parameters.

This way, testers can configure the nodes as per their [cross browser testing](https://www.browserstack.com/live) requirements, using the combination of browsers, their versions, and operating systems.

**Step 5: Using Selenium Grid to run tests**

Once the Selenium Grid setup is done by following the above 4 steps, testers can access the grid to run tests. If Selenium 1 RC nodes are being used, testers can use DefaultSelenium object and pass the same in the hub formation using the following command.

If you are using Remote WebDriver nodes, then use RemoteWebDriver and DesiredCapabilities object to define the browser, version, and platform. For this, create the target browser capabilities to run the test on.

DesiredCapabilities capability = DesiredCapabilities.internetexplorer ();

Once created, pass this set of browser capabilities into the RemoteWebDriver object

WebDriver driver = new RemoteWebDriver(new URL("https://localhost:4444/wd/hub"), capability);

Once this is done, the hub would assign the test to a matching node, if all the requested capabilities meet. If you wish to request any specific capabilities on the grid, specify them before you pass it to the WebDriver object in the following pattern

capability.setBrowserName();

capability.setPlatform();

capability.setVersion()

capability.setCapability(,);

The capabilities, if it does not exist on the grid, returns no match and thus, the test would fail to run.

Let us understand this using an example, considering a node is registered with the setting

-browser browser Name=internet explorer, version=10, maxInstances=4, platform=WINDOWS

Then, it is a match with the following set of capabilities defined for the test

capability.setBrowserName(“internet explorer” );

capability.setPlatform(“WINDOWS”);

capability.setVersion(“10”);

It would also match with the following set of capabilities defined for the test

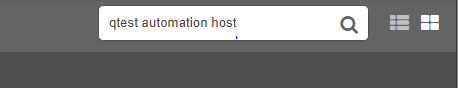
capability.setBrowserName(“internet explorer” );

capability.setVersion(“10”);

# Integration with qTest and CI/CD:

**Installation:**

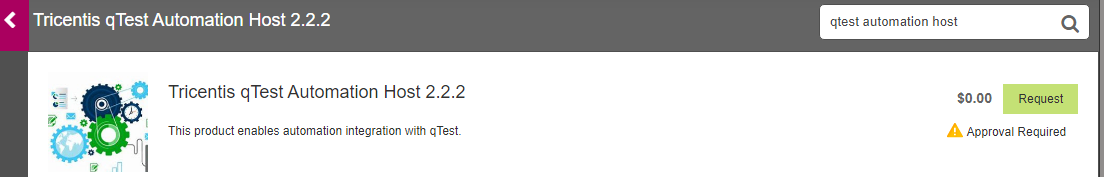
1. Step 1: Navigate to <http://appshop.humana.com/Shopping>
2. Search for Qtest Automation Host



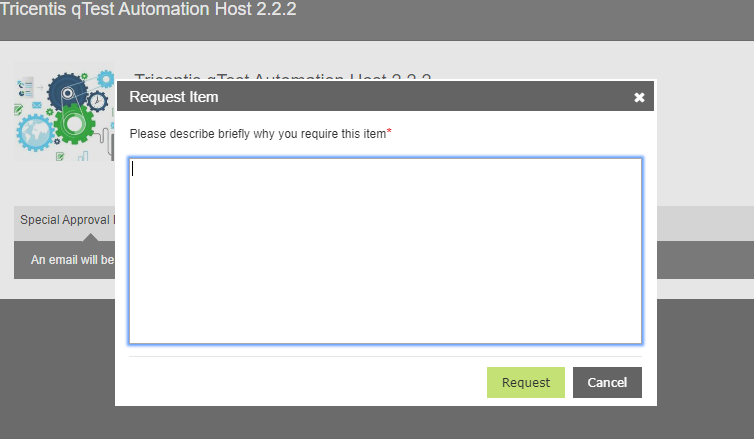
### Select the More info in the Tricentis qTest Automation Host Icon

### 

1. Click Request Button



1. Provide the purpose of the software in the below pop up and click Request Button



Once Submitted you will see the pop up in top of your screen saying you have requested for the Tricentis qTest Automation host.

The installation request takes one or two business days.

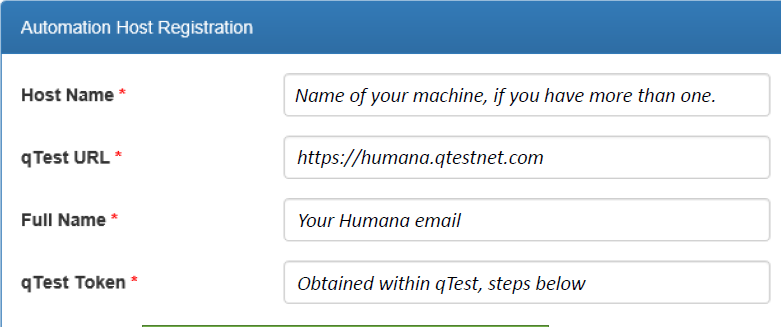
Note: If the Automation host version is 2.2.2 you need to follow the steps in the below attachment.



**Registration:**

Step 1: Navigate to <http://localhost:6789/home>

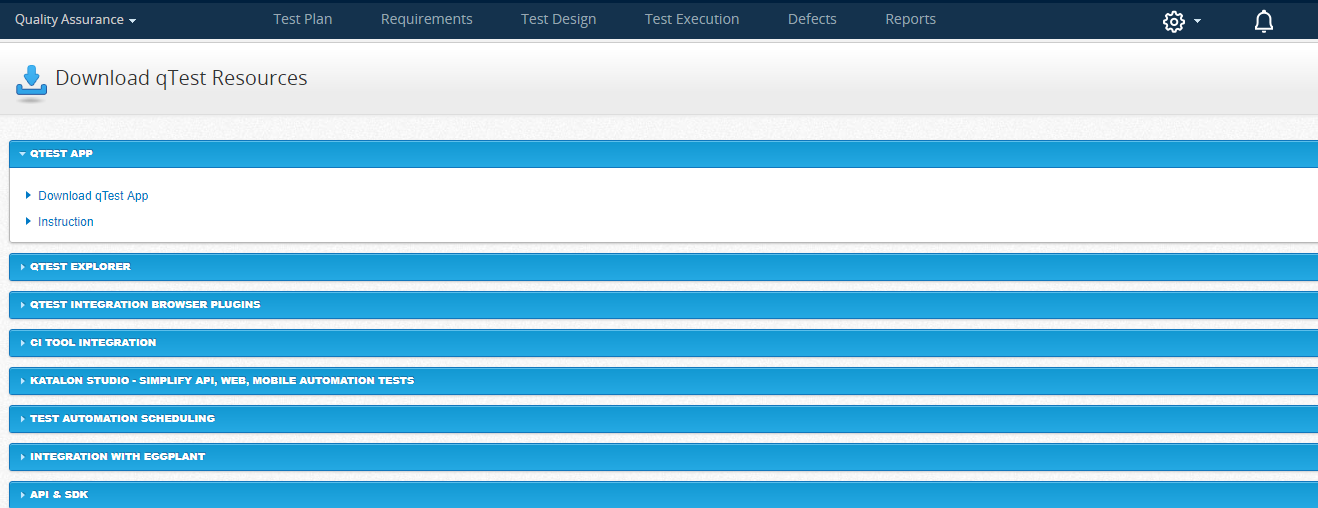
Step 2: The below registration window will come



Step 3: Login to qtest with SSO option. <http://go/qtest>

Step 4: Click Resources Icon in the upper right corner

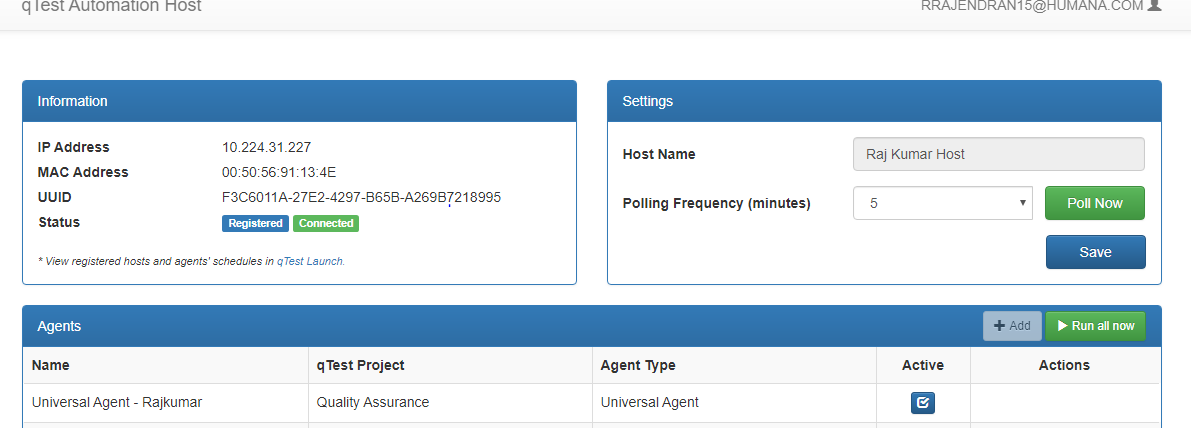




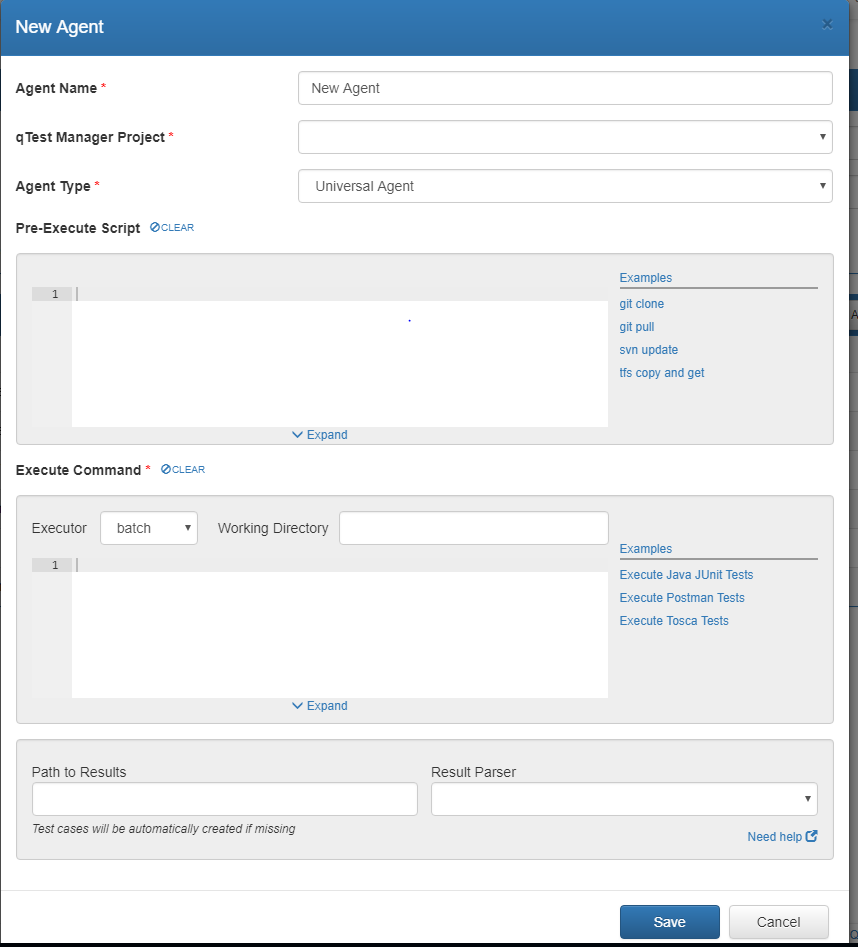
Step 5: In API & SDK section, Copy token which is highlighted in Yellow and populate the same in Qtest token field in registration screen and click register.

**Agent set up:**

Step 1: Navigate to <http://localhost:6789/home> and you will be navigated to the below page



Sep 2: Click Add button in the Agents window. New Agent window will appear.



Step 3: Enter the below information to configure your agent

Agent Name: Custom user defined name

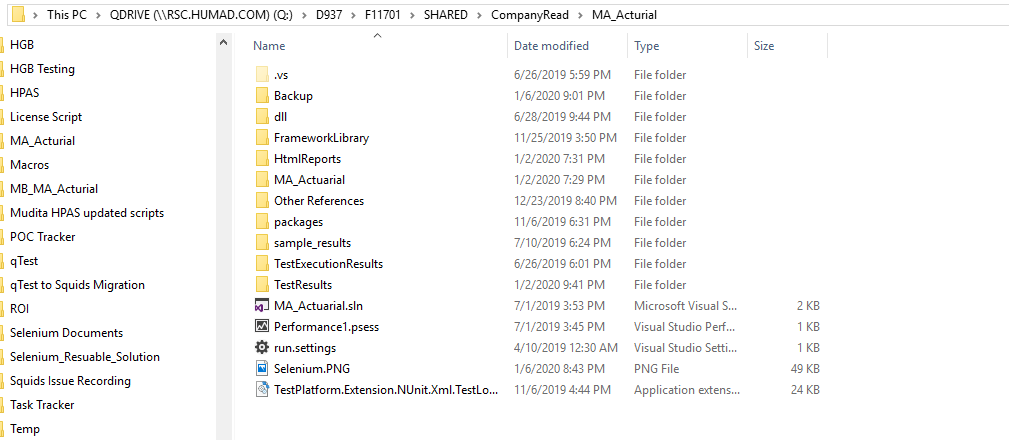
Project: Select your appropriated project

Agent Type: Universal Agent

Executor: Node

Working Directory: Root folder of your project

“Q:\D937\F11701\SHARED\CompanyRead\MA\_Acturial”



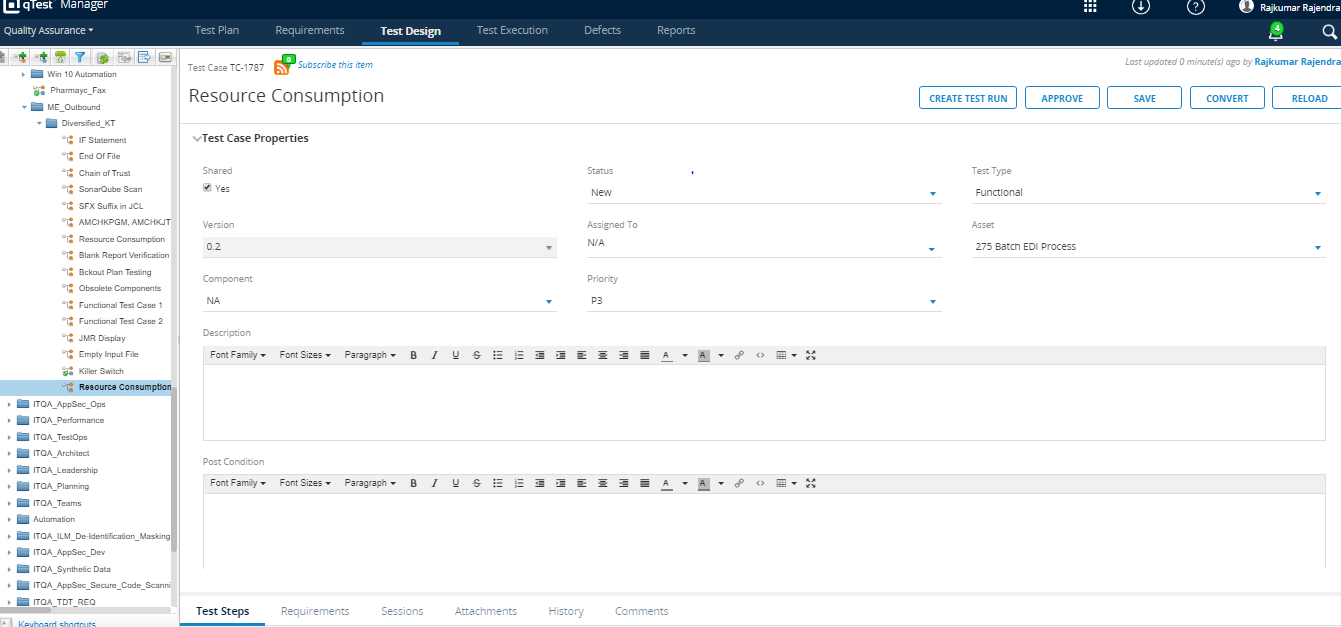
Executor: Copy paste the code in the below attachment.



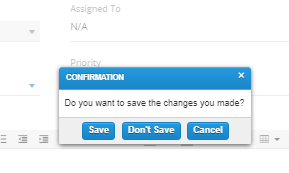
Path to Results: “Q:\D937\F11701\SHARED\CompanyRead\MA\_Acturial\QtestResult”

Result Parser: Nunit (Build-In)

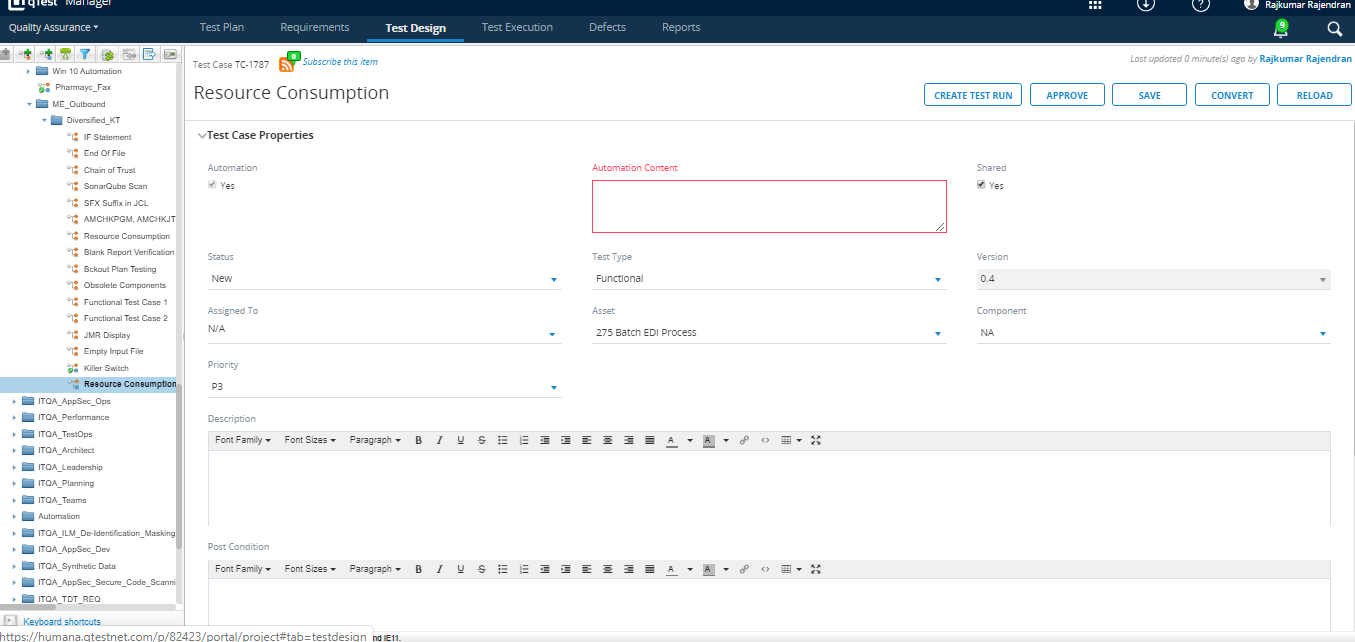
Convert manual test case to automation



Click on convert and you will receive a pop up asking confirmation of save changes as below

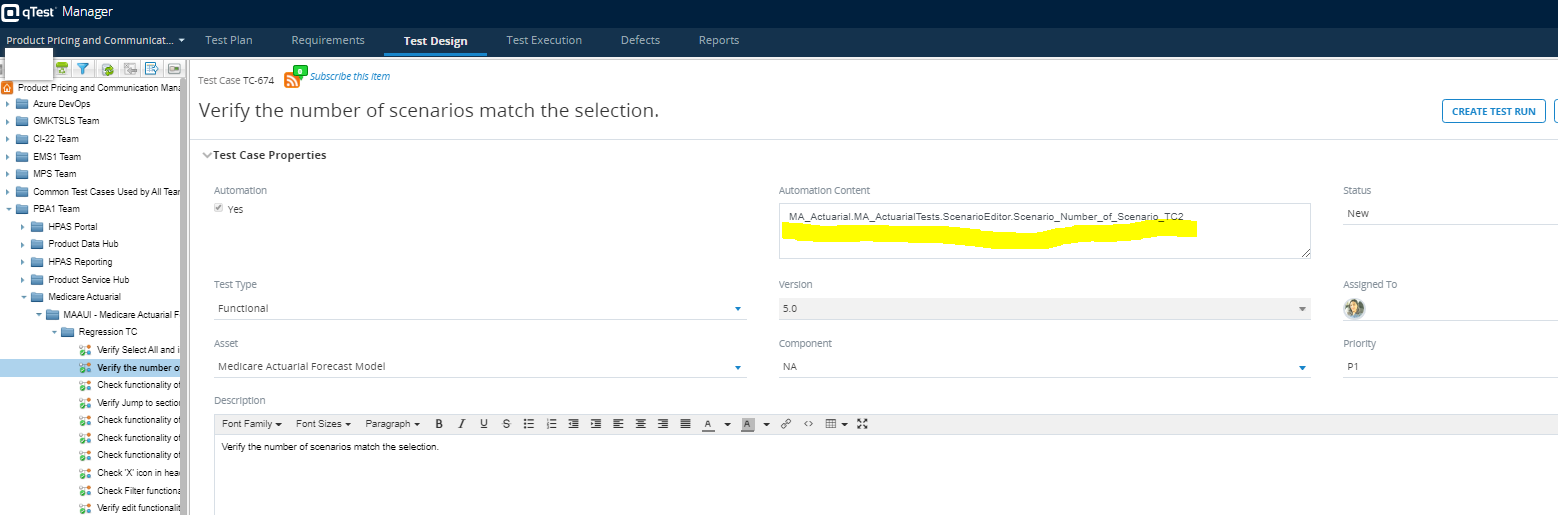


Click save then it gets navigated to below screen

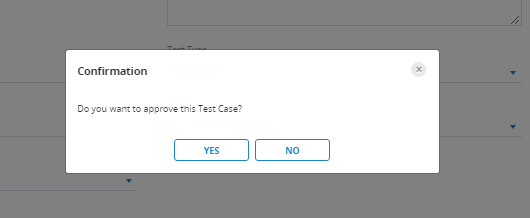


Enter automation content as Testmethod name as shown below

Eg : MA\_Actuarial.MA\_ActuarialTests.ScenarioEditor.Scenario\_Number\_of\_Scenario\_TC2



Click on Approve and it will ask for confirmation, click on yes.

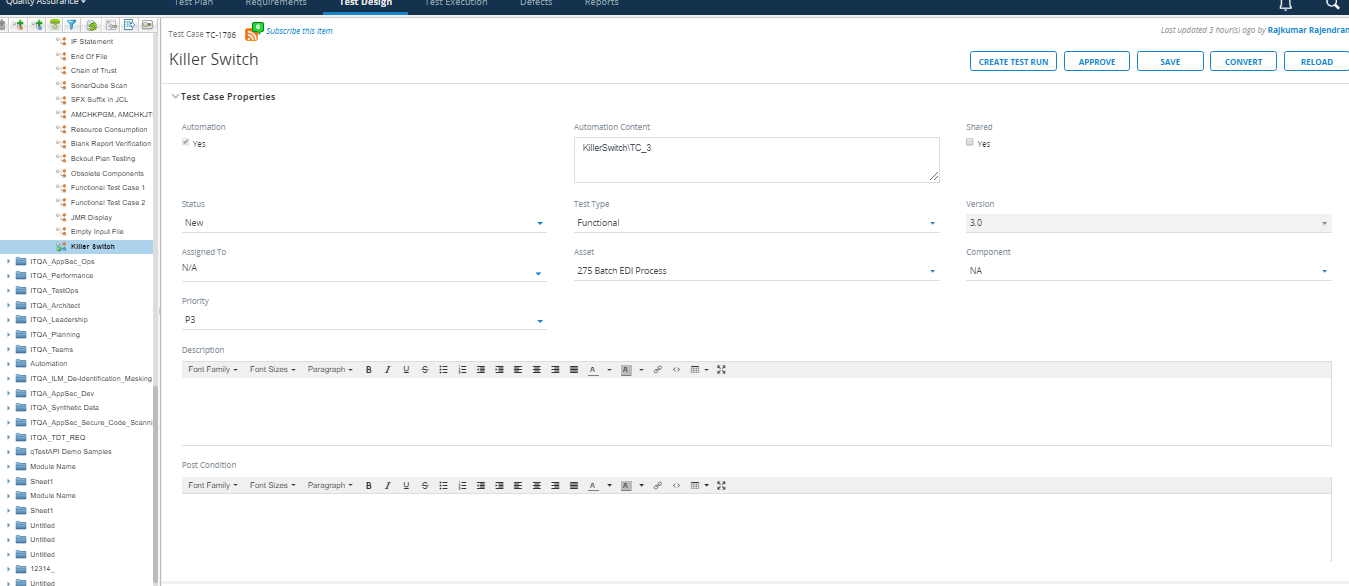


**Scheduling Automation Execution**

Step 1: Click the hyper link that appears in the Create Test Case Progress Result window.

You can also directly navigate to Test Design tab in qTest Manager.

Under the module that appeared in window you can see the test cases that are scanned and selected,

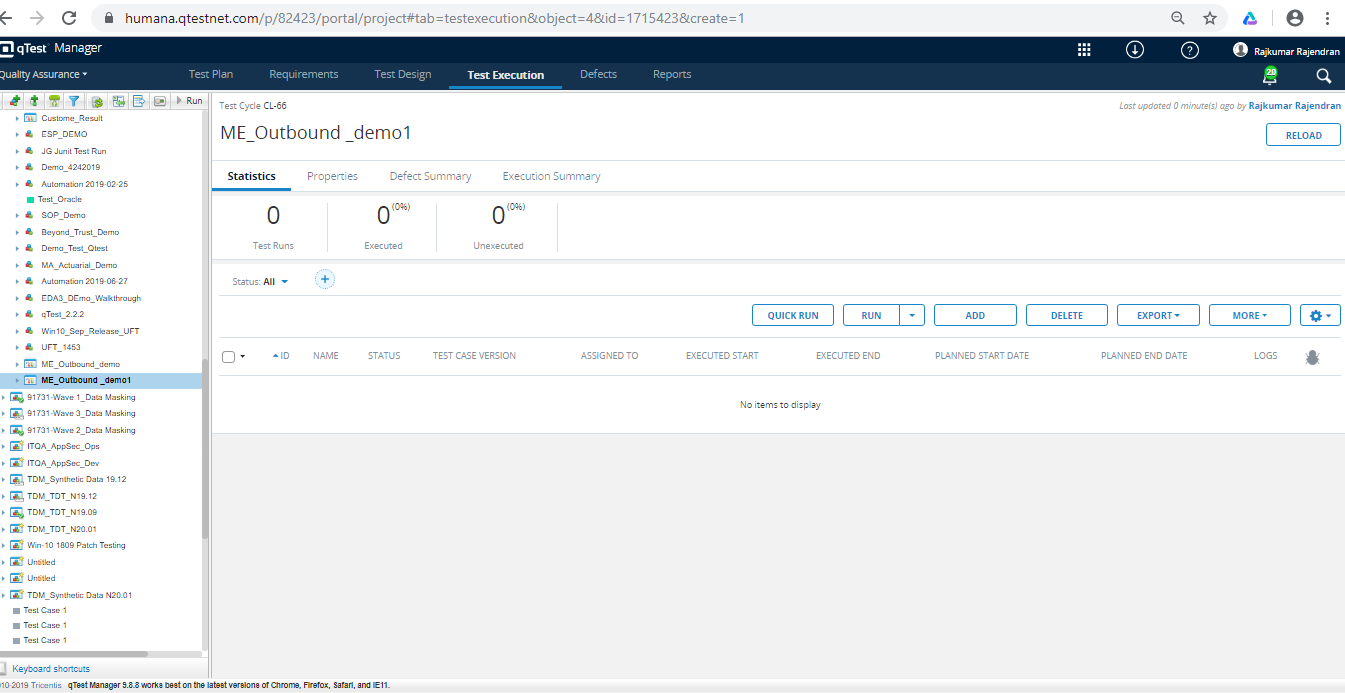


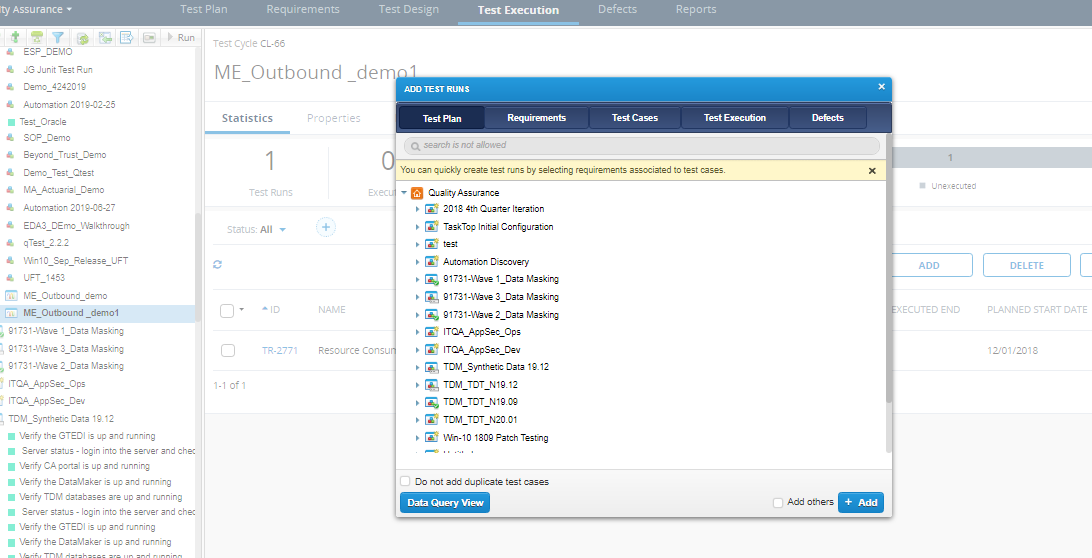
Step 2: Navigate to Test Execution Window

Create a new Test Cycle/ Test Suite or Test Run or continue to use the existing ones as applicable

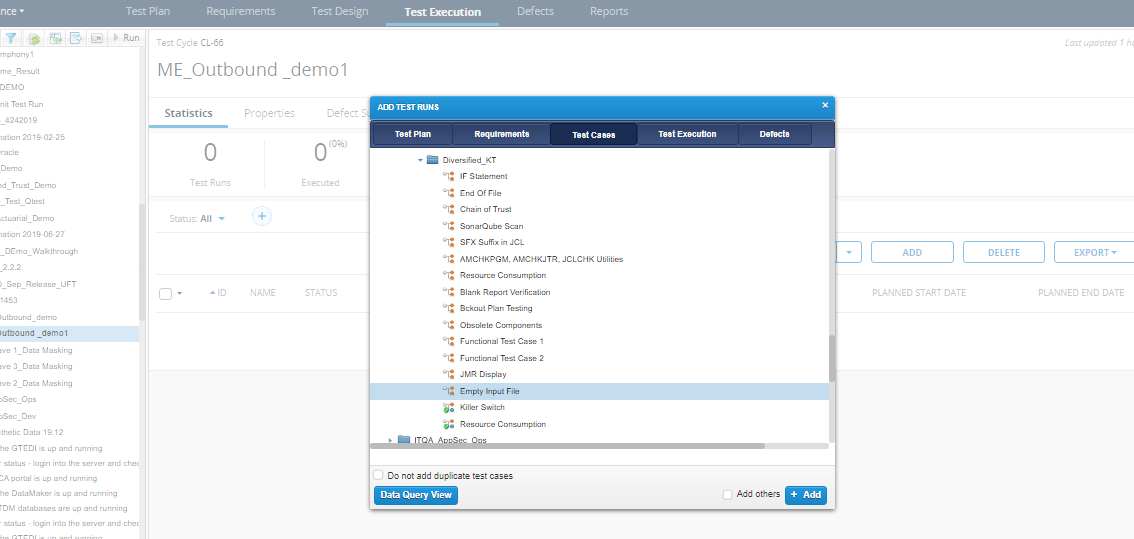
Under the created Test Run

Step 3: Click on Statistics Tab – Click Add Button

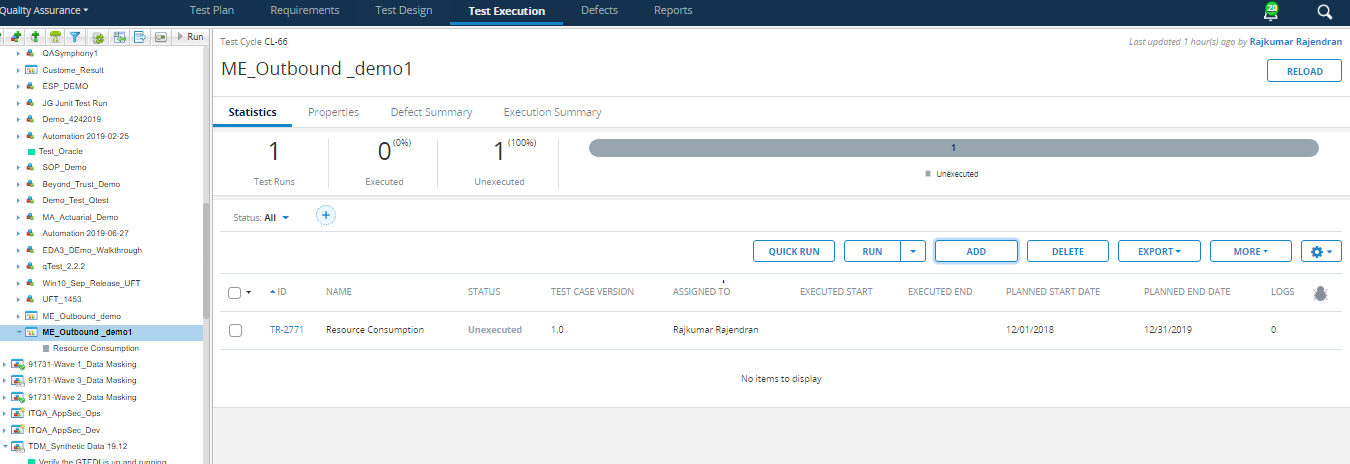




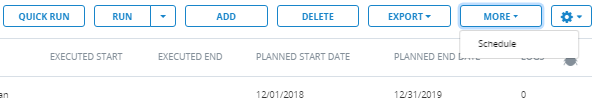
Step 4: Click on Test Cases Tab and expand the appropriate Module. Select one or more test cases that you added recently as part of the Agent configuration.



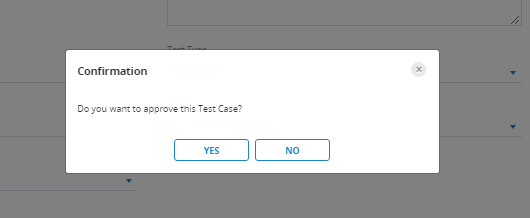
Step 5: Once the desired test cases are added, close the window and you could see the added test case showing in Statistics tab of the main page.



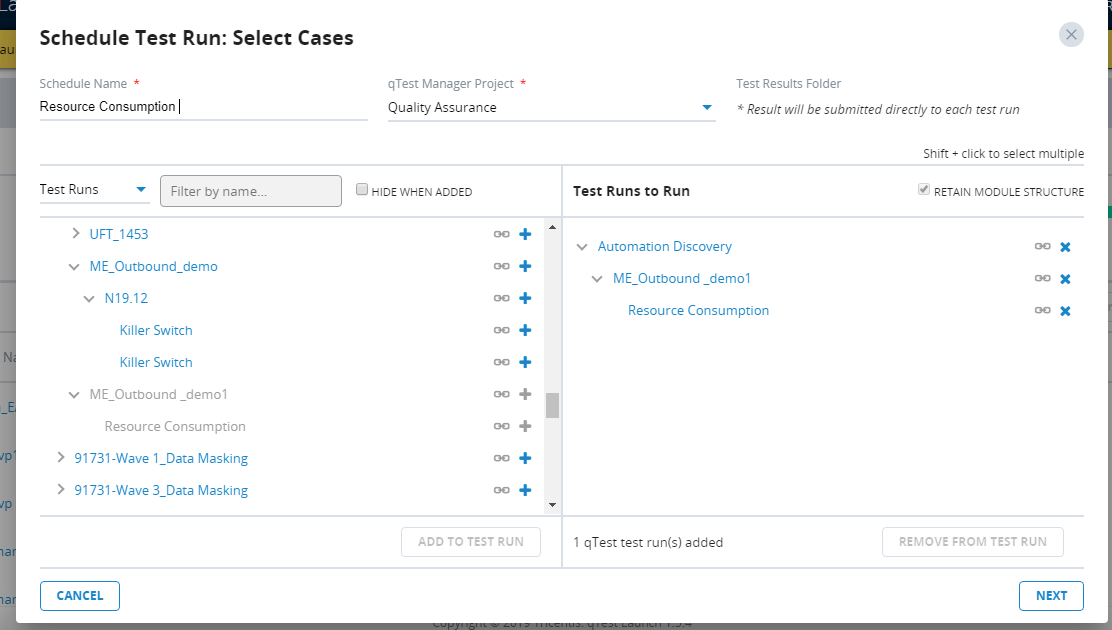
Now, select the test cases that you wanted to schedule for execution. Click “More” option and select Schedule

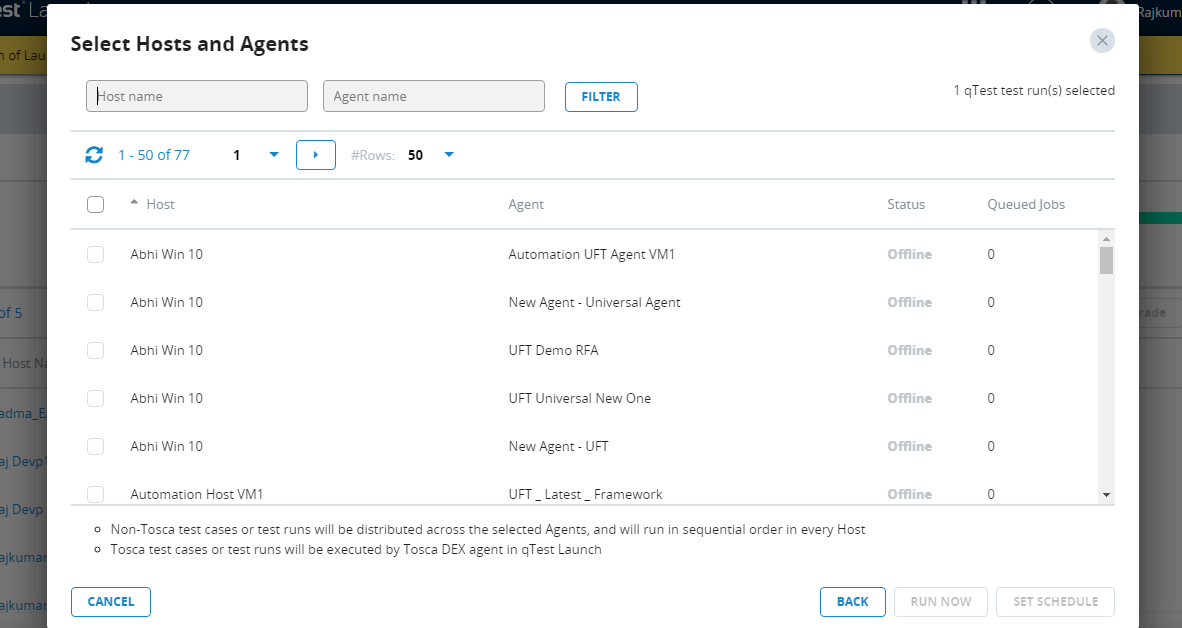


Step 6: Test cases that are not approved will be asked for approval. Select the approve check box and click yes



Step 7: Schedule Test Run Window Appears





Select the Agent that is configured as part of the initial process and click Ok.

Note: If you wish the execution to be started in a specific time period or immediately upon schedule, please make appropriate selections.

**Trigger Execution:**

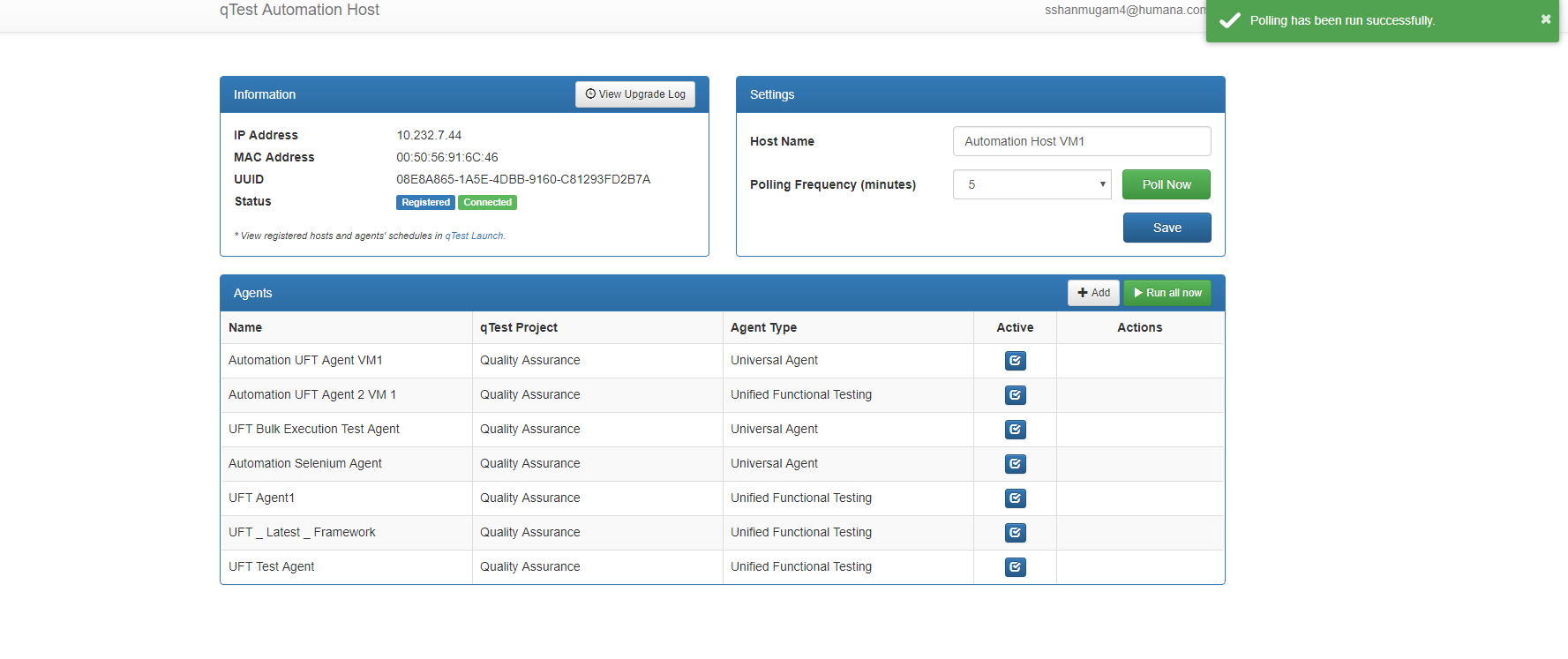
**Automatic Trigger:**

The script will get automatically triggered based on the polling Frequency assigned in qTest Automation Host page

Please use this link to navigate to Host page <http://localhost:6789/home>

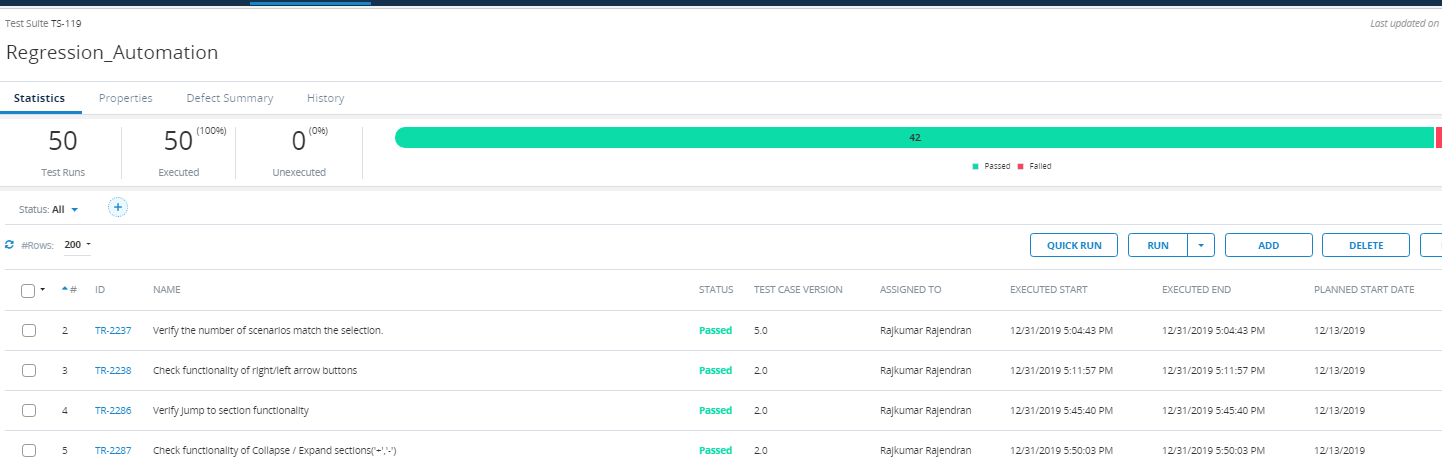
**Manual Trigger:**

Navigate to Host page <http://localhost:6789/home> and click Poll Now Button. The scripts will start to execute immediately



**Validate Result:**

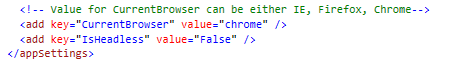
Once execution is completed, please Navigate to Test Execution Tab in qTest and Select the Test Run that you scheduled and view the result.



# Headless Automation

A headless browser is a type of software that can access webpages but does not show them to the user and can pipe the content of the webpages to another program. Unlike a normal browser, nothing will appear on the screen when you start up a headless browser, since the programs run at the backend.

In Our Framework we have these configuration in **“App**.**config”** file as shown below:



According to our framework design the Headless configuration runs in chrome only. As Mentioned above, when value of the key “**IsHeadless**” is set to “False” the script is not run according to Headless Configuration. Instead it executes as per the driver initialization mentioned in “**InitializeWebDriver**”.

When value of the key “IsHeadless” is set to “True” the script triggers the Headless Configuration. The function invoked in “**InitializeWebDriver**” of framework libraryclass file is called from the key, value present in App.Config File.