**Ignite Automation Framework**

Version- 1.0

April 2016

Contents

[Objective 1](#_Toc454441275)

[Framework Features 1](#_Toc454441276)

[Framework Benefits Reusability 1](#_Toc454441277)

[Reusability 1](#_Toc454441278)

[Optimum utilization of the tool 1](#_Toc454441279)

[Less effort 1](#_Toc454441280)

[Increased quality 1](#_Toc454441281)

[Greater productivity 1](#_Toc454441282)

[Maintenance 1](#_Toc454441283)

[No scripting skills required by the end user 1](#_Toc454441284)

[Return on investment is high 1](#_Toc454441285)

[Framework Architecture 1](#_Toc454441286)

[Framework Architecture 1](#_Toc454441287)

[Framework 1](#_Toc454441288)

[Abstract Layer 1](#_Toc454441289)

[External Data 1](#_Toc454441290)

[Framework at Glance 1](#_Toc454441291)

[Inputs 1](#_Toc454441292)

[Scripting standards 1](#_Toc454441293)

[Standards for Test Scripting 1](#_Toc454441294)

[Getting Started 1](#_Toc454441295)

[Keyword List 1](#_Toc454441296)

[Appendix 1](#_Toc454441297)

# Objective

Automation testing draws maximum benefits with minimum effort. The benefit of automation testing is its ability to increase the efficiency of resources, increase test coverage, and increase the quality and reliability of the software. This document introduces one particularly effective type: the Ignite Automation Framework. In the Ignite Automation Framework, the discrete functional business events that make up an application are described using test dictionary of keywords used in test cases written plain English with in the test management tool SpiraTest. This approach fosters code reusability, optimum utilization of the tool, and greater productivity.

Automation testing requires a well-defined approach, based on a comprehensive framework, in order to reap maximum benefits. A framework is a hierarchical directory that encapsulates shared resources, such as a dynamic shared library, image files, localized strings, header files, and reference documentation, in a single project. There are various ways to design framework using one of the following approach available for automation, such as:

* Test Script Modularity Framework
* Test Library Architecture Framework
* Data-Driven Automation Framework
* Hybrid Automation Framework
* Keyword-Driven Automation Framework

Test Management tool based test design and test automation is formed on the idea that the discrete functional business events that make up any application can be described using manual test written in particular format in short text description using keywords. By designing plain English test dictionary for keywords that describe those discrete functional business events, testers begin to build a common library of dictionary and keywords that can be used to create test scripts. The Ignite Automation Framework is a test management tool driven automation framework that works with Selenium Web Driver. This framework allows testers to develop test cases using SpiraTest management tool using test dictionary. When the test is executed, the framework processes the Test Run Excel workbook (extracted from tool) or can connect to the tool directly to read and update test result. Driver engin calls functions associated with the keywords found in the test step. These keyword functions in turn perform specific actions against the application under test (AUT). The framework interprets the keyword and executes a set of statements in the program. With this framework in place, applications can be automated without starting from scratch. Testers simply use the application-independent test dictionary of keywords and create extra application-specific test dictionary.

Framework Features

In addition to standard features such as performing operations and verifications on the objects, the framework includes other sophisticated features, such as:

**1. Use of variables:** Variables can be defined and used across the generated test script. This can be used to capture runtime values, which can be reused as input elsewhere during test execution.

**2. Conditional checking:** Conditional constructs such as ‘if’ can be implemented using test dictionary of keywords to handle different flows based on various conditions.

**3. Data-driven testing:** This framework supports data-driven testing by importing data from an external data sheet.

**4. Reports:** Customized reporter messages can also be used to perform effective analysis on execution reports. These reports can be customized to display the pass or fail condition of any functionality, even during the verification of any checkpoints.

**5. Calling functions and reusable actions:** Common functions or actions can be triggered through keywords. This framework supports a functional decomposition approach. This increases the reusability of functions, which in turn reduces the unnecessary repetition of steps.

**6. Exception handling:** Runtime errors can be effectively handled and reported using this framework.

Framework Benefits Reusability

### Reusability

The Ignite Automation Framework is an application-independent framework that deals with all possible actions and verifications that can be performed on an object. Therefore, the code for the same object can be used across different applications. Duplication of work is minimized at every level. For instance, a user might have to perform a certain action on an object of a similar class (e.g., clicking a button) repeatedly. This can be in the same test case or in a different application altogether. In both cases, the same code can be reused.

Optimum utilization of the tool

The framework has the advantage of using Test dictionary of keywords as the input for triggering an action. This well-built framework uses the features of the tool effectively. For instance, there is a shared object repository where all the objects required can be added and reused across the scripts for an application under test.

Less effort

The effort involved in coding and reviewing is minimal when compared to other frameworks, since a good percentage of coding is done within the framework. The tester simply has to enter the test step using test dictionary of keywords, reducing the time required for coding. Recording is also not required as the global repository is used. The amount of rework required for migrating from one application to the other on the same platform is reduced since the code remains the same.

### Increased quality

The scripts will be of uniform quality since they make use of the same code.

### Greater productivity

The Ignite Automation Framework provides both qualitative and quantitative benefits for automation, and it is highly productive compared to any other framework. This framework also addresses the ongoing maintenance of the test scripts in a cost-effective manner.

Maintenance

Simple modifications to the application can be easily handled. The changes will be done only in the external file containing the script and the code need not be changed. Hence it is easy to maintain the scripts and provide cost-effective solutions for the test automation.

No scripting skills required by the end user

No coding skills are required to automate and review the scripts. The scripts are user-friendly with good readability. Scripts can be interpreted easily by a person who does not have complete knowledge of the tool.

Return on investment is high

Although the initial effort for building the framework is high, in the long run, the return on investment will be high because of the reusability and optimum utilization of the tool.

# Framework Architecture

## Framework Architecture

In this approach, the goal is to develop an application-independent reusable-keyword framework that can be used directly across any application without spending any extra time on it. In order to make all the components of the system work in sync, it is important to define the components and its functionalities, as well as the binding relationship between them. An Automation Framework Architecture comprises the following components:

Framework

The framework consists of the following sub-components:

* AppModules
  + All application specific functions will appear here
* automationFramework
  + All Driver and Translate Engine will be here
* ObjectMap
  + Object resolution takes place here
* pageObject
  + Any page level objects will be created here

## Abstract Layer

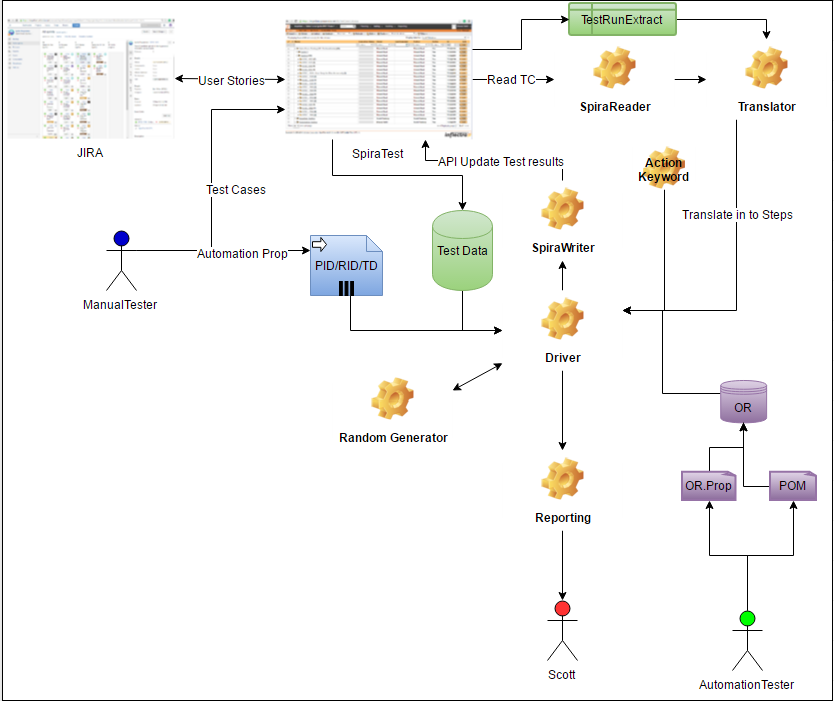
The abstract layer consists of the following sub-components:

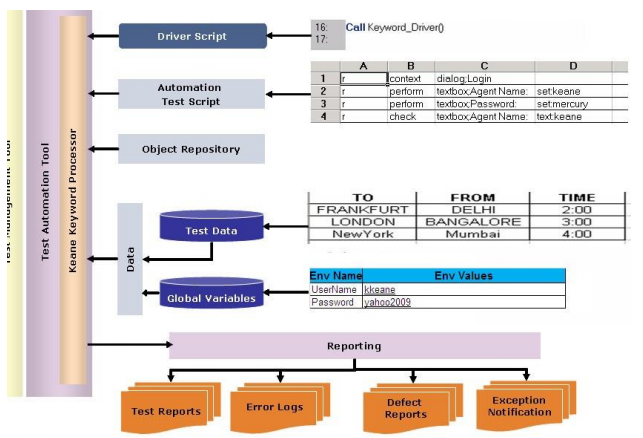
* SeleniumWD
  + Old typical keyword framework is here to mix and match Ignite
* Utility
  + All utility Browser Factory, Constants, ExcelUtility, Log, RandonGenerator, Utils
* SpiraTest Soap API
  + All spiratest soap library will come here
* SpiraTest Listener
  + SpiraTest listener will appear here

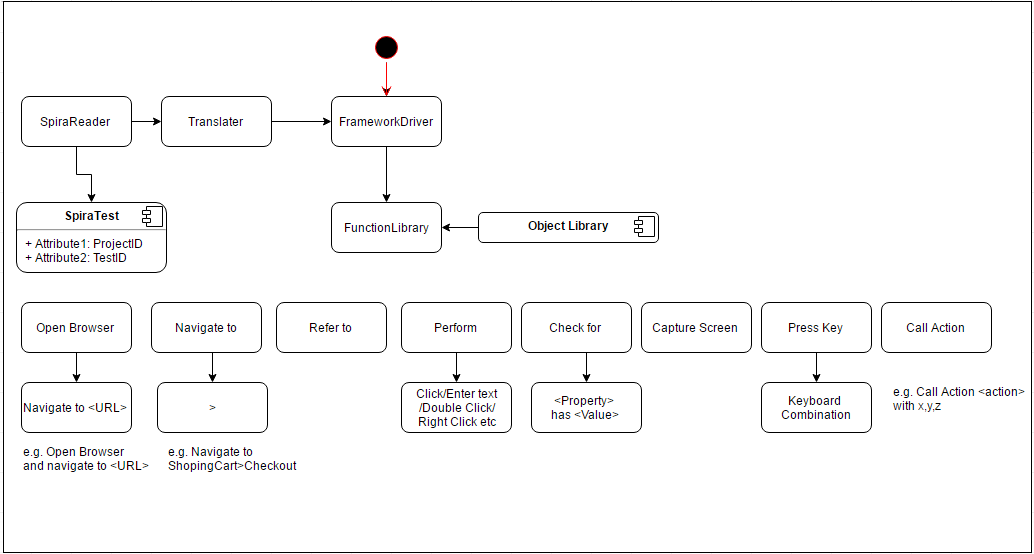
## External Data

External data consists of the following sub-components:

* TestData
  + Any fixed test data will come here e.g. Lookup for random data generation
* ObjectRepository property file to add objects and properties will come here



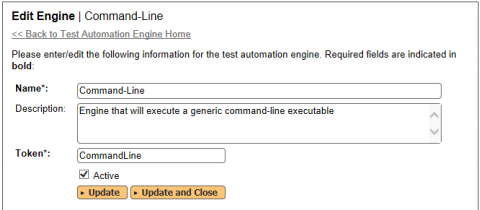




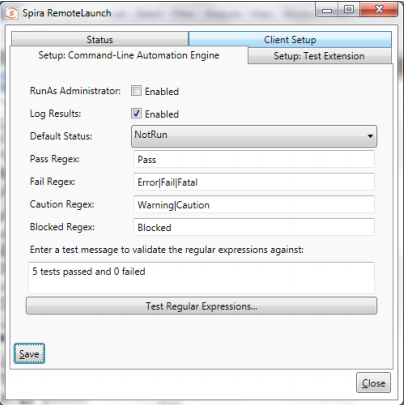
# Framework at Glance

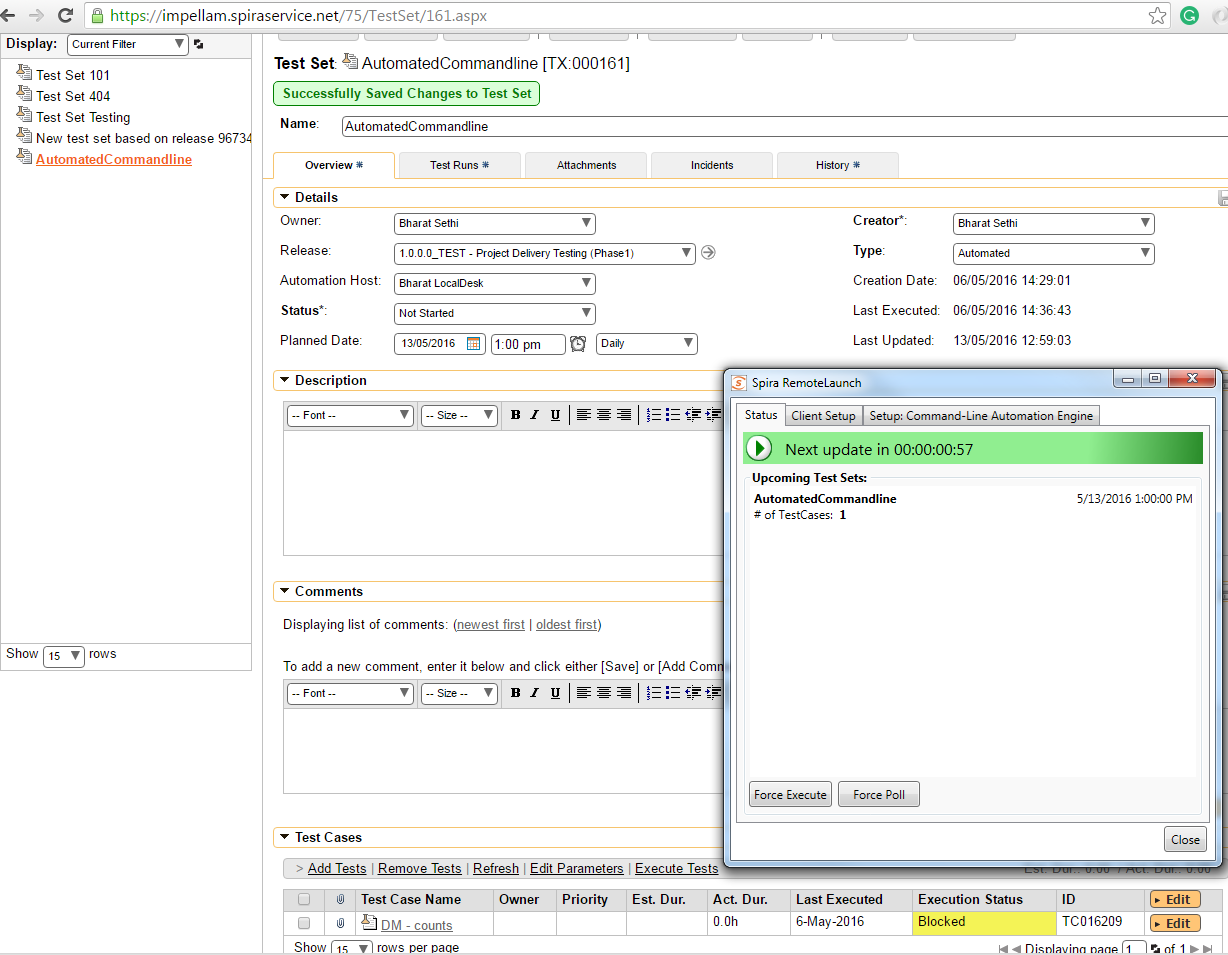
All the input and output components are explained here.

* Ensure that command line engine is installed and configured as explained below:
  + Download and extract the CommandLineAutomationEngine.zip file from the Inflectra website and locate the CommandLine.dll
  + Copy the file “CommandLine.dll” into the “extensions” sub-folder of the RemoteLaunch installation.
  + Log in to SpiraTeam as a system administrator and go into SpiraTeam main Administration page and click on the “Test Automation” link under **Integration**.
  + Click the “Add” button to enter the new test automation engine details page. The fields required are as follows:



* + - **Name**: This is the short display name of the automation engine. It can be anything that is meaningful to your users.
    - **Description**: This is the long description of the automation engine. It can be anything that is meaningful to your users. (Optional)
    - **Active**: If checked, the engine is active and able to be used for any project.
    - **Token**: This needs to be the assigned unique token for the automation engine and is used to tell RemoteLaunch which engine to actually use for a given test case. For Command-Line this should be simply “CommandLine”.
  + Once you have finished, click the “Insert & Close” button and you will be taken back to the Test Automation list page, with Command-Line listed as an available automation engine.
* Ensure command line setting of Remote Launch
  + You may need to modify the Command-Line configuration for some of the specific automation hosts, by right-clicking on the RemoteLaunch icon in the system tray and choosing “Configuration”. That will bring up the RemoteLaunch configuration page. The Command-Line engine adds its own tab to this page which allows you to configure how the Command-Line engine operates:





The following fields can be specified on this screen:

**RunAs Administrator** – This normally should not be checked. However if your automation tool requires Windows UAC elevation to operate, you will need to select this option. We recommend initially trying your tool with the value unchecked. Then, if you get an error message “requires elevation” in the test results you will need to select the option.

**Log Results** – Normally the command-line engine will capture the output results from the command-line and send the results back to SpiraTeam as the test result. When you are executing a tool that directly integrates with SpiraTeam (e.g. a NUnit test suite that is already integrated with SpiraTeam) you don’t want two different results to be sent back. In such a scenario, deselecting this option will prevent the command-line engine from sending back its own test result.

**Default Status** – This specifies the execution status that will be returned to SpiraTeam in the event that none of the regular expressions (Regex) specified match the results returned from the test application. By default, the system will return “Passed” if none of the other regular expressions match correctly.

**Pass Regex** – This is the regular expression that is used to match a passed test result. By default the system will search for the phrase “Pass” in the test output and return a Passed status if the match is successful.

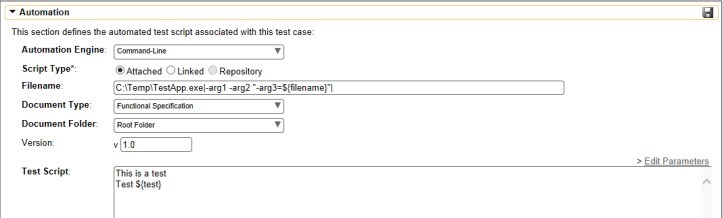
**Fail Regex** – This is the regular expression that is used to match a failed test result. By default the system will search for the phrases “Fail”, “Error” and “Fatal” in the test output and return a Fail status if any of the matches are successful.

**Caution Regex** – This is the regular expression that is used to match a caution test result. By default the system will search for the phrases “Warning” and “Caution” in the test output and return a Caution status if any of the matches are successful.

**Blocked Regex** – This is the regular expression that is used to match a blocked test result. By default the system will search for the phrase “Blocked” in the test output and return a Blocked status if the match is successful.

**Test Regular Expressions** – This text box lets you enter in some sample text and see how the Command-Line extension would interpret it. Once you have entered in the text, click “Test Regular Expression…” and the system will display a popup message box letting you know what the outcome of such a test would be interpreted as:

* Ensure to setup each automated test case in SpiraTest with command line filename and setup to execute IgniteFramework(Selenium WebDriver based) from command line
  + First navigate to the list of test cases in SpiraTeam (by clicking Testing > Test Cases) and then add a new test case. Once you have added the new test case, click on it and select the “Automation” tab:



**Automation Engine** - Choose the Command-Line Automation Engine that you created in the previous section from the drop-down list.

**Script Type** – This should be set to Attached/Linked for this case

**Filename** – This needs to consist of the following sections separated by a pipe (|) character:

ω The full path to the command-line tool. To make this easier across different machines, you can use several constants for standard Windows locations:

[MyDocuments] – The user’s “My Documents” folder. The user indicated is the user that ran RemoteLaunch.

[CommonDocuments] – The Public Document’s folder.

[DesktopDirectory] – The user’s Desktop folder. The user indicated is the user that ran RemoteLaunch.

[ProgramFiles] – Translated to the Program Files directory. For 64-bit machines, it’s the 64-bit directory.

[ProgramFilesX86] – Translated to the 32-bit Program Files directory.

ω Any arguments for the command-line tool, with the filename specified as ${filename}. This special token will be replaced by the actual filename of the test script when RemoteLaunch downloads it from SpiraTeam. In addition, you can use the following additional tokens for some of the special SpiraTeam ID values:

[TestCaseId] – the ID of the test case

[TestSetId] – the ID of the test set

[ReleaseId] – the ID of the release (if specified)

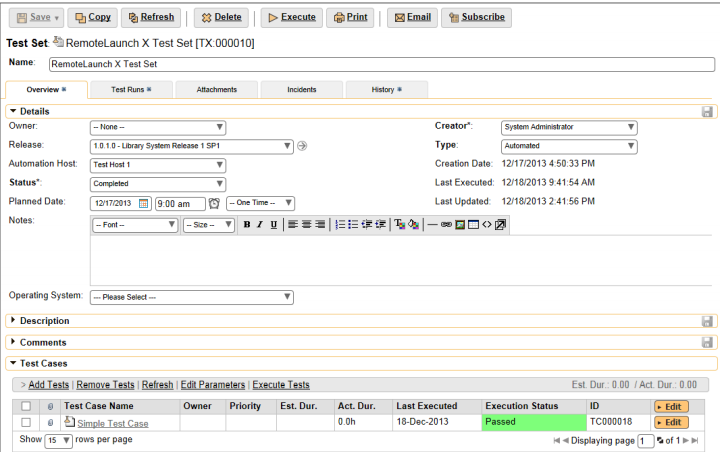
[ProjectId] – the ID of the project

ω An example filename would be: C:\Temp\TestApp.exe|-arg1 -arg2 "-arg3=${filename}"|

Version – The version of the test script (1.0 is used if no value specified)

Test Script – This needs to contain the complete test script in whatever language and syntax is being expected by the command-line application

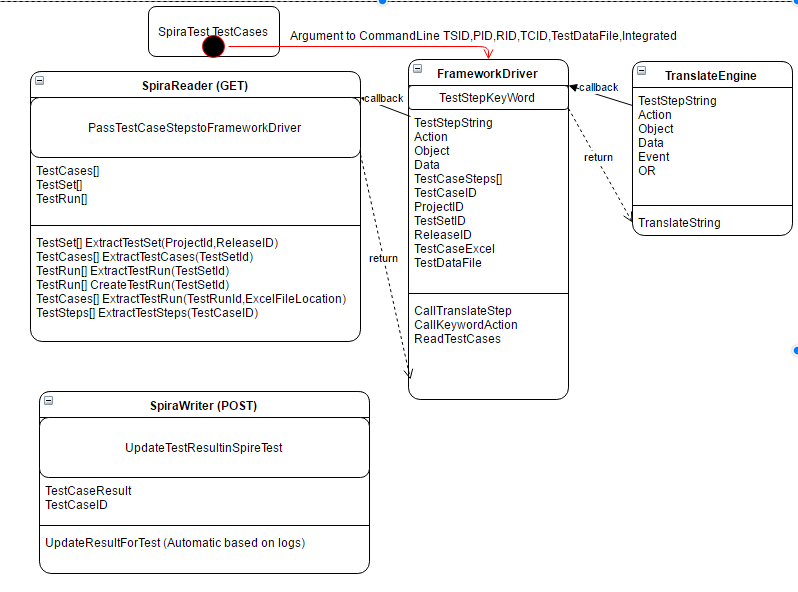
Once you are happy with the values, click [Save] to update the test case. Now you are ready to schedule the automated test case for execution.

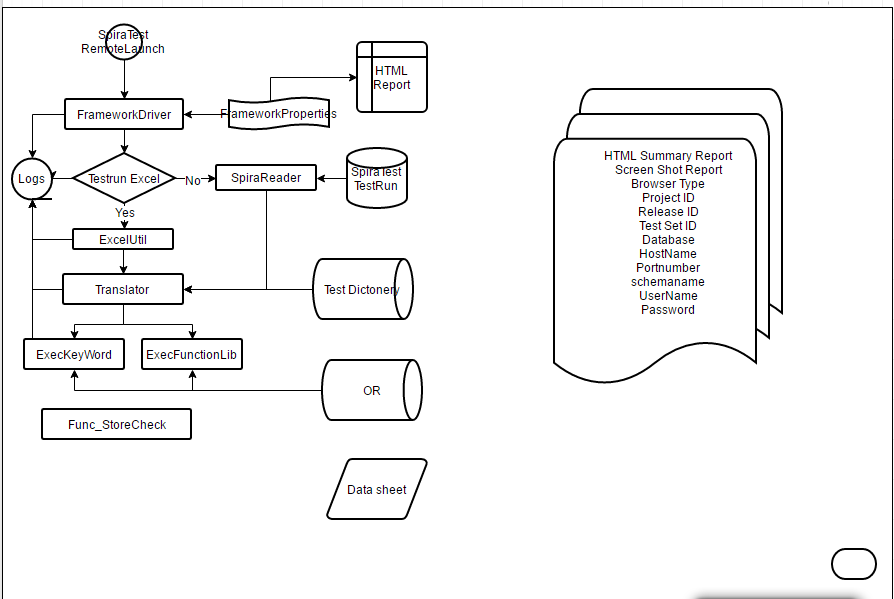
* Ensure to pass ProjectID, ReleaseID, TestSetID, TestCaseID in each of the argument
* Ensure Automation Host has been added in SpiraTest on Testing>Automation Hosts screen
  + Make sure that you have created an Automation Host for each computer that is going to run an automated test case. The name and description can be set to anything meaningful, but the Token field must be set to the same token that is specified in the RemoteLaunchX application on that specific machine
* Ensure test has been scheduled to run test set from SpiraTest
* 
  + You need to add at least one automated test case to the test set and then configure the following fields:
    - Automation Host – This needs to be set to the name of the automation host that will be running the automated test set.
    - Planned Date – The date and time that you want the scenario to begin. (Note that multiple test sets scheduled at the exact same time will be scheduled by Test Set ID order.)
    - Status – This needs to be set to “Not Started” for RemoteLaunch to pick up the scheduled test set. When you change the Planned Date, the status automatically switches back to “Not Started”
    - Type – This needs to be set to “Automated” for automated testing
* Ensure Command Line test has got SysOut to print actions on the console so that Spiratest can have them in detail
* Use of FAIL, ERROR, FATAL, WARNING, CAUTION in sysout for any fail test

## Inputs

SpiraTest Remote Launch fires a command line program for a project with Integrated Flag as true or false. If Flag is passed as true then Framework will read test steps from SpiraTest API else it will read from Excel.

For each step read from excel/spiratest will be passed to translator





# Scripting standards

This section provides details about the various ways to be used during scripting, the keywords in plain English and their descriptions, along with methodologies to be followed while scripting in Test Management tool.

## Standards for Test Scripting

Getting Started

Before learning about the keywords used for test scripting, the user should know the basic test script and how the framework is called. The execution is started by running the file TestAction.Java with 6 arguments where the method main(String Args[]) is called.

#### Description of Test Suite

Ignite Framework is designed to work with Test Management Tool(SpiraTest), so that testers can write Test Cases, Test Set and link them with Releases in any project. When framework is being executed it will read all the test set linked with a release and then read each of the test case in the test set. Test cases are written in plain English using keywords in it and Framework is capable to translate those test scripts in to keyword driven framework.

##### Add Test Cases to Test Set

##### Link Test Sets to Release

#### Description of Test Case

This section describes the keywords used in Test Case for test scripting in plain English.

Keywords are expending based on day today usage of English in test steps

##### Step

Every test step must start with the defined list of keyword in plain English such :

Open Browser or Open - To Open a browser

Store value – to Store a value in a variable

Enter – to Enter given text into a textbox or textarea

Click on – to Click on any object

The translation engine translate above steps into keywords those are:

1. LaunchApp „LaunchApp‟ is used to launch the AUT. This keyword triggers the driver script to launch the application. This will launch the application either in Internet Explorer or in Firefox or inChrome, **depending upon the browser specified in the test set name**.

2. Perform „Perform‟ is used to perform an operation on a particular object (ex: clicking on a button on a Web page, or typing some text in a textbox). This keyword has different plain English keyword mapped to resolve which operations to be performed.

3. Check „Check‟ is used to check if the required property of a particular object is attained at runtime. This is a type of validation step (expected result).

4. Condition „Condition‟ is used to compare two variables, check properties for the existence of windows, etc.

5. Storevalue „Storevalue‟ is used to store the property values of different objects in different environment variables. These environment variables can later be used as input parameters in various functions and also in scripts.

6. Wait „Wait‟ is used to place static waits in the test script.

7. Loop „Loop‟ is used to loop a set of actions given in the data table.

8. Context ‘Context’ Keyword is supported to set the context on the page or Dialog/alert.

##### Operation

Second part of the test step must be

## Keyword List

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Action** | **Object** | **ObjProp** | **Event** | **TestData** | **Comments** | **Existing** | **Sample Test Step** |
| launchapp | <BrowserName> | <URL> |  |  | Launches the given URL | Y | Open Browser “URL”, Open “URL, Navigate to “URL” |
| wait | <milliseconds> |  |  |  | Waits for the given interval | Y | wait for 1000 |
| wait | <ObjectName> |  |  |  | Dynamic wait to wait for object to appear | Y | wait for obj=username |
| importdata | <Path of Data excel file> |  |  |  | Imports the Data from the given path (Data Sheet path) | Y | import data "c:\data.xls" |
| screencapture |  |  |  |  | Captures the screen shot for all the (Perform,Storevalue,Check) Test steps | N | Screenshot,  screen shot , capture screen |
| screencaptureoption | <<Perform>>/<<Storevalue>>/<<Check>> |  |  |  | Captures the screen shot for all the Perform/Storevalue/Check test steps, if indicated in the Object column | N | Screeenoption to perform |
| Condition | Var1 | equals | Var2 |  | Checks whether the the given variable Var1 equals Var2. If so returns true else false | Y | condition #test=#test1, condition #test equal to #test1 condition #test="true",condition #test equal to "true" |
| Condition | Var1 | not equals | Var2 |  | Checks whether the the given variable Var1 not equals Var2. If so returns true else false | Y | condition #test not equals #test1, condition #test != #test1 |
| Condition | Var1 | greaterthan | Var2 |  | Checks whether the the given variable Var1 greater than Var2. If so returns true else false | Y | condition #test > #test1  condition #test greater than #test1 |
| Condition | Var1 | lessthan | Var2 |  | Checks whether the the given variable Var1 less than Var2. If so returns true else false | Y | condition #test < #test1  condition #test less than #test1 |
| Endcondition |  |  |  |  |  | Y | End condition |
| loop | <count> |  |  |  | Executes the statements inside the loop for <count> iterations | Y | loop 4 times, loop 4 |
| Endloop |  |  |  |  |  | Y | End loop |
| perform | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Frame/ iFrame/ Table/element> | <ObjectName> | click |  | Clicks the required object | Y | click on textbox obj=username |
| perform | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Frame/ iFrame/ Table/element> | <ObjectName> | altclick |  | In some elements, particularly in sub menu items, if Ignite's 'Click' action doesn't work, use this action. We use Java script executor to perform 'click' operation in tests where the Selenium 'click' method doesn't work | Y | alt click on textbox obj=username |
| perform | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Frame/ iFrame/ Table/element> | <ObjectName> | enter | “Value” | When Ignite's doesn't work for certain controls (especially when click on command button) doesn't work use this action. We actually perform an 'Enter' key stroke in this case. | Y | Hit enter on obj=txtbox |
| perform | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Frame/ iFrame/ Table/element> | <ObjectName> | hover |  | Some menu items / elements we need to hover mouse the web elements, use this mehod. | Y | hoverover on image obj=img |
| perform | <ComboBox> | <ObjectName> | select | <value> | Choose a value from dropdown. We can select from the data table as well as an environmental variable. | Y | select "mr" from obj=username |
| perform | <ListBox> | <ObjectName> | listselect | <Value1>,<Value2> | Select set of values from listbox. One data variable and envvironmental variable is allowed. | Y | listselect "mr","mrs" from obj=username listselect "mr" and "mrs" from obj=username |
| perform | <TextBox> | <ObjectName> | set | <text> | Enters <text> in a text value | Y | set text "sayemul.makki" in textbox obj=username  enter "sayemul.makki" in textbox obj=username |
| perform | <CheckBox> | <ObjectName> | check | <ON/OFF> |  | Y | Select “on” on checkbox obj=chkbox |
| check | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> | <ObjectName> | Visible | <True/False> | Returns whether object is Visible or not. If so True, else False; Data table variables/ Environment variables can be used here for True/false. | Y | check visible textbox obj=usename check not visible textbox obj=usename |
| check | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> | <ObjectName> | enabled | <True/False> | Returns whether object is enabled or not. If so True, else False; Data Data table variables/ Environment variables can be used here for True/false. | Y | check enable webelement obj=username check not enable webelement obj=username |
| check | <TextBox/ Button/ ComboBox/ RadioButton/ CheckBox/ TextArea/element> | <ObjectName> | text | <Text to compare> | Checks the Text property of the given object.Data table variables/ Environment variables can be used here for text to compare | Y | check text in textbox as "sayemul.makki", check text textbox obj=usename |
| check | <Link> | <ObjectName> | linktext | <Text to compare> | checks the displayed text of the link.Data table variables/ Environment variables can be used here for text to compare | Y | check link text textbox obj=usename |
| check | <ComboBox> | <ComboBoxName> | value | <value to be compare> | Validate the selected item from the combobox.Data table variables/ Environment variables can be used here for value to compare | Y | check value textbox obj=usename |
| check | <RadioButton/CheckBox> | <Object Name> | checked | <ON/OFF> | Data table variables/ Environment variables can be used here for value to compare whether checked is ON/OFF. | Y | check if checked checkbox obj=usename |
| storevalue | <TextBox/ Button/ ComboBox/ RadioButton/ CheckBox/ TextArea/element> | <Object Name> | text | <Variable Name> | Assigns the text value to a variable | Y | read text/store text from  <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> obj=Objprop in #vartext |
| storevalue | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> | <Object Name> | Visible | <Variable Name> | Assigns the display status of the element in a variable. | Y | read visible/store visible from <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> obj=Objprop in  #varvisible |
| storevalue | <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> | <Object Name> | enabled | <Variable Name> | Returns whether object is enabled or not to a boolean variable | Y | read enable/store enable from <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> obj=Objprop in #varenable |
| storevalue | <ComboBox> | <TextBoxName> | value | <VariableName> | Assigns the selected item to the variable | Y | read value/store value/store from  <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> obj=Objprop in #varvalue |
| storevalue | <Link> | <Object Name> | linktext | <VariableName> | Stores the linktext to the variable | Y | read link text/store link text from <TextBox/ CheckBox/ RadioButton/ Button/ Link/ ComboBox/ TextArea/ Image/ Table/ListBox/element> obj=Objprop in #varlinktext |
| Context | <frame/iframe> | <frame object name> |  |  | Use to switch the context to the frame. Once you switch the context to the frame, you will be able to access and work on the elements inside the frame.  **Note:** If you want to work with the frame and again if you want to access the objects outside the frame, context should be set to the page . | Y | Set Context <frame>/<iframe> |
| Context | Browser | WebBrowser | page | <pagetitle>::<frame/iframe>;<frame object name> | Performs the same operation explained in the above row.  Note: If you migrate from Ignite for QTP to Ignite for Selenium or vice versa, this syntax will be useful. | Y | Set context browser page with <title> |
| Context | <frame/iframe> |  |  |  | Switch the control to the parent frame when context is inside child frame. | Y | Set Context <frame>/<iframe> |
| Context | Page | pagetitle |  |  |  |  |  |
| Context | Browser | WebBrowser | page | <pagetitle> | Set the context to the page. If multiple windows with the same title, context will be set to the first window. If page title is blank, context will be set to the next window. |  |  |
| perform | Page | pagetitle | close |  | Close the page with the pagetitle | Y | close page with <title> |
| perform | Page |  | close |  | Close current page | Y | close page |
| check | Page | pagetitle | exist | true/false | check whether the page available. Data and environmental variable are also allowable for true/false |  |  |
| check | Page | pagetitle | pagetitle | <pagetitle> | Check the page title.Data and environmental variable are also allowable for true/false | Y | check page title obj=username |
| Storevalue | Page | pagetitle | exist | <Var> | store the existence of the page |  |  |
| storevalue | Page | pagetitle | pagetitle | <var> | Store the page title to the variable. | Y | page title "xyz"/#title exist in #pagetitle |
| Context | Browser | WebBrowser | dialog | <Optional title> | Set the context to the dialog pop up |  |  |
| perform | Dialog | <Optional title> | ok |  | Accept the dialog (equialent to click on 'OK' in the msg/dialog) | Y | click ok on dialog *<title>* |
| perform | Dialog | <Optional title> | close |  | close the dialog |  |  |
| perform | Dialog | <Optional title> | cancel |  | Abort the dialog operation |  |  |
| Check | TextElement | <ObjectName> | text | <text to be compared> | To compare the static text with the given text.Data and environmental variable are also allowable for text to compare |  |  |
| StoreValue | TextElement | <ObjectName> | text | <Variable Name> | Store the static text in the variable. |  |  |
| download | link;<<link Object Name>> | <ObjectName> | <<Path to save the file>> |  |  |  |  |
| upload | <<File path in C column of Test Script>> |  |  |  | Upload the file . **Precondition:** A file upload dialog is Open |  |  |
| upload | <<File path in C column of Test Script>> |  | closeupload/ cancel upload |  | Close file upload dialog.Precondition: A file upload dialog is Open |  |  |
| perform | Calendar | cal\_Calendar1 | setdate | <Date in mm-dd-yyyy format only>> | Object type should be **calendar** and object name should be prefixed by **cal\_** |  |  |
| callaction | <<Give the TestCaseID. i.e, the test script, which we call as a reusable component>> |  |  |  | Use if any testscript has to be called from another Test Script |  |  |
| callfunction | <<Give the user defined function name (Which you have created inside the functionlibrary class)>> | <Variable>/dt\_<datasheet in teatdata workbook>/<<Arg1,Arg2,Arg3………………………………>> | <<Arg1:Arg2:Arg3………………………………>> |  | Use if any project related/ customized functionality to be supported. Add the functionality as a Java method inside the framework class itsel and pass the method name and arguments in your Ignite test script. |  |  |
| check | Table | <tableobjectname> | rowcount | < Give the rowcount to be compared> | Compare the row count of the table object with the given number. | Y | check table row count for table obj=tablename in #rowcount |
| storevalue | Table | <tableobjectname> | rowcount | < Variable Name> | Store the row count of the table to the given variable. | Y | get/read/store row count for table obj=tablename in #rowcount |
| check | Table | <tableobjectname> | columncount | < Give the rowcount to be compared> | Compare the column count of the table object with the given number. | Y | check table column count for table obj=tablename in #rowcount |
| storevalue | Table | <tableobjectname> | columncount | < Variable Name> | Store the Column count of the table to the given variable. | Y | get/read/store col/column count for table obj=tablename in #colcount |
| check | Table | <tableobjectname> | getcelldata | <Value to be compared>,<rownumber>,<Column Number> | Get the cell present in the given cell(row, column) and compare with the given data. **Limitation:** Pass only the direct input. i.e., Don't pass Test Data (or) Environmental Variable. |  |  |
| storevalue | Table | <tableobjectname> | getcelldata | < Variable Name to store the value in the cell> | Get the cell present in the given cell(row, column) and store in the given data variable. **Limitation:** Pass only the direct input. i.e., Don't pass Test Data (or) Environmental Variable. | Y | get/read/store data in cell from table obj=tablename in #celldata |
| fetchdb | sql Select Query | <<MSSQL / MYSQL>> |  |  | SQL query will get execute . And result set is copied to Excel file in current working dir. Pre Req. Utlity file should have db conncetion details along witht Name, username and password |  |  |
| comparedbcell | Tableelement | <objectName> | text | dt\_columnname | it will search : compare the text of given object with the data table  cell value from given column of data table |  |  |
| close | broweser name | URL | close |  | close the browser | Y | close the browser |

Keywords that used in dictionary are:

1. LaunchApp „LaunchApp‟ is used to launch the AUT. This keyword triggers the driver script to launch the application. This will launch the application either in Internet Explorer or in Firefox, depending upon the browser specified in the Selenium.java file.

2. Perform „Perform‟ is used to perform an operation on a particular object (ex: clicking on a button on a Web page, or typing some text in a textbox). This keyword should be entered in the corresponding row in the second column if any such operations are to be performed. 3. Check „Check‟ is used to check if the required property of a particular object is attained at runtime. This is a type of validation step (expected result).

4. Condition „Condition‟ is used to compare two variables, check properties for the existence of windows, etc.

5. Storevalue „Storevalue‟ is used to store the property values of different objects in different environment variables. These environment variables can later be used as input parameters in various functions and also in scripts.

6. Wait „Wait‟ is used to place static waits in the keyword script.

7. Loop „Loop‟ is used to loop a set of actions given in the data table.

8. Context ‘Context’ Keyword is supported to set the context on the page or Dialog/alert.

# Appendix

* <https://www.inflectra.com/Documents/SpiraTest-Team%20RemoteLaunch%20Automated%20Testing%20User%20Guide.pdf>
* <https://www.inflectra.com/Documents/SpiraTest-Team%20Automated%20Testing%20Integration%20Guide.pdf>