E2E Test Automation Framework – User Guide

Test Automation

© ValueLabs

Plot No.41 • HITEC City

Phase II • Cyberabad • 500081

Phone +91 40 6623 9000• Fax 6623 9100

Document Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference** | **ValueLabs\_E2E\_Framework\_User\_Guide** | **Version No.** | **1.3** |
| **Release Date** | **28 Apr, 2014** | **Total No. of pages** | **60** |
|  | **Name** | **Designation** | **Date** |
| **Prepared by** |  |  | **28 Apr, 2014** |
| **Reviewed By** |  |  |  |
| **Approved by** |  |  |  |

Document Amendment Record

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Rev. From** | **Rev. To** | **Date of Change** | **Changes Made** |
|  |  |  |  |  |
|  |  |  |  |  |

**Table of Contents**

[1. Introduction 7](#_Toc386467846)

[1.1. Purpose 8](#_Toc386467847)

[1.2. Scope 8](#_Toc386467848)

[1.3. Intended Audience 8](#_Toc386467849)

[1.4. Document Organization 8](#_Toc386467850)

[2. Directory Structure 9](#_Toc386467851)

[2.1. Config File 10](#_Toc386467852)

[2.1.1. Application 11](#_Toc386467853)

[2.1.2. Testcases 11](#_Toc386467854)

[2.1.3. URL 13](#_Toc386467855)

[2.1.4. Browser 13](#_Toc386467856)

[2.1.5. Test Data Source 14](#_Toc386467857)

[2.2. The Properties file 14](#_Toc386467858)

[2.2.1. <appDetails> 14](#_Toc386467859)

[2.2.2. <common> 15](#_Toc386467860)

[2.2.3. <misc> 16](#_Toc386467861)

[2.2.4. <SeleniumGrid> 18](#_Toc386467862)

[2.2.5. <Email> 19](#_Toc386467863)

[2.2.6. <testdataDB> 20](#_Toc386467864)

[2.2.7. <appDataDB> 20](#_Toc386467865)

[2.2.8. <BugTrackerDetails> 21](#_Toc386467866)

[2.2.9. <TCMTool> 22](#_Toc386467867)

[3. Detailed Usage of Keywords 23](#_Toc386467868)

[3.1. Web Keywords 23](#_Toc386467869)

[3.1.1. Check 23](#_Toc386467870)

[3.1.2. Clear 23](#_Toc386467871)

[3.1.3. Clearenter 24](#_Toc386467872)

[3.1.4. Clearsession 24](#_Toc386467873)

[3.1.5. Click 25](#_Toc386467874)

[3.1.6. Clickandwait 25](#_Toc386467875)

[3.1.7. Closewindow 25](#_Toc386467876)

[3.1.8. Draganddrop 26](#_Toc386467877)

[3.1.9. Enter 26](#_Toc386467878)

[3.1.10. Goback 27](#_Toc386467879)

[3.1.11. Isenabled 27](#_Toc386467880)

[3.1.12. Isdisabled 27](#_Toc386467881)

[3.1.13. Keypress 28](#_Toc386467882)

[3.1.14. Mouseover 28](#_Toc386467883)

[3.1.15. Openurl 28](#_Toc386467884)

[3.1.16. Refresh 29](#_Toc386467885)

[3.1.17. Select 29](#_Toc386467886)

[3.1.18. Selectandwait 31](#_Toc386467887)

[3.1.19. Selectwindow 31](#_Toc386467888)

[3.1.20. Storeattribute 31](#_Toc386467889)

[3.1.21. Storevalue 32](#_Toc386467890)

[3.1.22. Uncheck 33](#_Toc386467891)

[3.1.23. Verify 33](#_Toc386467892)

[3.1.24. Verifyattribute 35](#_Toc386467893)

[3.1.25. Verifycontinue 36](#_Toc386467894)

[3.1.26. Verifynotpresent 36](#_Toc386467895)

[3.1.27. VerifyObject 37](#_Toc386467896)

[3.1.28. Waitforelement 38](#_Toc386467897)

[3.2. Generic Keywords 39](#_Toc386467898)

[3.2.1. Executetestcase 39](#_Toc386467899)

[3.2.2. Storecomments 39](#_Toc386467900)

[3.2.3. Waittime 39](#_Toc386467901)

[3.3. Conditional and Looping Keywords 40](#_Toc386467902)

[3.3.1. If 40](#_Toc386467903)

[3.3.2. Loop 41](#_Toc386467904)

[3.4. Webtable Keywords 41](#_Toc386467905)

[3.4.1. Columndatacheck 41](#_Toc386467906)

[3.4.2. Verifysort 43](#_Toc386467907)

[3.4.3. Verifytabledata 43](#_Toc386467908)

[3.4.4. Verifytablerowexist 44](#_Toc386467909)

[3.4.5. Verifyvalueinrow 45](#_Toc386467910)

[3.5. DB Keywords 46](#_Toc386467911)

[3.5.1. DB\_update 46](#_Toc386467912)

[3.5.2. DB\_verifycolumns 46](#_Toc386467913)

[3.5.3. DB\_verifytablecontent 47](#_Toc386467914)

[3.5.4. DB\_verifyduplicatevaluesonprimarykey 47](#_Toc386467915)

[3.5.5. DB\_verifynotnullconstraint 47](#_Toc386467916)

[3.5.6. DB\_verifyprimarykeys 47](#_Toc386467917)

[3.5.7. DB\_Verifyrowscount 48](#_Toc386467918)

[3.5.8. DB\_Verifyspexecutiontime 48](#_Toc386467919)

[3.5.9. DB\_Verifystoredprocedure 48](#_Toc386467920)

[3.6. REST Keywords 49](#_Toc386467921)

[3.7. Using the Testdata Parameters 48](#_Toc386467921)

[3.7.1. Using Date as Parameter 49](#_Toc386467922)

[3.7.2. Using RND Parameter 50](#_Toc386467923)

[3.8. Adding New Keywords 50](#_Toc386467924)

[3.8.1. Using Framework Methods in AppTestType 51](#_Toc386467925)

[3.9. On Fail Procedures 57](#_Toc386467926)

[3.10. Using Function Driven Framework 57](#_Toc386467927)

[3.10.1. Config File 57](#_Toc386467928)

[3.10.2. Guidelines to Write the Function Driven Tests 59](#_Toc386467929)

[3.10.3. The Properties File 60](#_Toc386467930)

[3.11. Encrypted data creation 60](#_Toc386467931)

# Introduction

Valuelabs End to End Framework is built using Java as programming language.

It is a hybrid framework that is built using Selenium as api, to perform the most common web operations listed below.

* Click
* Type
* Check
* Select
* Verify

The framework is built on a keyword driven approach and all the different components are integrated through various functions.

Following are the common features of Framework.

* Supports various datasource formats (Excel / XML / Database).
* Had detailed reports generated in Excel and HTML formats after execution.
* Ability to mail test reports after execution.
* Multiple application parallel execution (using Multi Threading).
* Integrated with Selenium Grid for cross platform execution.
* Integrated with test case management tool **TestLink**.
* Integrated with bug tracking tool **Bugzilla**.
* User has flexibility to Add New Keywords as per requirements.
* Ability to execute failed cases automatically.
* Can be scheduled using cron jobs / task schedulers.
* Can re-use commonly used test steps as template cases.
* Can have test results stored in DB for each and every run.
* Had ability to capture screen shots at each and every step.
* Can create evidence as part of testing with all the screenshots included.
* Execute test cases based on Module / Priority / Requirement.
* Integrated with Mobile Automation Tests (Android**)**.
* Have the ability to execute / skip a step based on condition.
* Can make a set of steps form a loop for required number of times.

## Purpose

The purpose of this document is to provide detailed information on directory structure and key work usage of E2E Framework.

## Scope

This document limits its scope in providing information about test automation on directory structure and keyword usage of E2E Framework.

## Intended Audience

The intended audience for this document is all end users using this framework.

## Document Organization

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Chapter** | **Contents** |
|  | [Introduction](#_Introduction) | Purpose, scope and intended audience of the document |
|  | [Directory Structure](#_Directory_Structure) | Detailed description of the structure of directory |
|  | [Detailed Usage of Keywords](#_Detailed_Usage_of) | Detailed descriptions of keywords with their usage |

# Directory Structure

Directory structure of the framework is maintained as shown below.

|  |  |
| --- | --- |
|  | **Framework** directory contains the required jar files and place holders for any .class and jar files to be added in future. |
| **TestInputs** directory is content holder for configuration files and testdata files. |
| **Properties** directory is content holder for property configuration and information regarding the application to be tested. |
| **TestReports** directory is content holder for writing Results to HTML files and Excel sheets. |
| **Screenshots** directory contains the screenshots generated when running tests. |

**Framework**

Framework directory is the base directory where Java classes and Jar files required for Framework execution are present. It also has some executables, which are considered as webdriver executable drivers for the respective browsers. It also has **AppTestType.java** file, which is used to define new keywords, as per requirement of the user.

**TestInputs**

TestInputsdirectory is the directory where TestInputs like **Config** files, **TestDataSource** files (Excel and XML), **Grid\_BrowserList.xml** (use for execution while using Selenium-gird) and the attachments that are configured by the user for mailing.

**Properties**

Propertiesdirectory contains various lists of property configuration files, which are of XML format. Each property file defines different run-time configurations, which are derived by various flags set in the property file.

**TestReports**

TestReportsdirectory is where the report files are generated in **HTML** and **Excel** formats. Both the reports are generated by a time stamp. Excel report is updated after the completing execution of each and every test case where as **HTML** report is generated after completing the execution of all test cases.

**Screenshots**

Screenshotsdirectory contains the screenshots taken as per the requirement and also the screenshots taken for failed cases. Screenshots folder contains a sub-directory which is created based on the **application name** (read from Application tag in Config).

The directory structure, where screenshot is created is shown below.

..// Screenshots // <app\_name> // <date>

**Example**

..// Screenshots // PHPScheduler // 051013

During runtime, for failed cases name of the screenshot is displayed as shown below.

<testcase\_id> \_ <testcase\_step\_no><date> \_ <time> .png

**Example**

8\_5051013\_190506.png

For detailed step level screenshots, the structure of screenshot directory is shown below.

..// Screenshots // <app\_name> // <date> // Log - <log\_no> // <testcase\_id>

**Example**

..// Screenshots // PHPScheduler // 051013 // Log – 24 // 1

Name of the screenshot is displayed as shown below.

Step\_<testcase\_step\_no>.png

**Example**

Step\_4.png

PDF files generated as part of evidence are stored as shown below.

..// Screenshots // <app\_name> // <date> // Log - <log\_no> // <testcase\_id>.pdf

**Example**

..// Screenshots // PHPScheduler // 051013 // Log – 24 // 1.pdf

* + User can specify the location of screenshots directory in the property file. If specified, screenshots will be saved in the same directory.

## Config File

**Config** file is present in **TestInputs** folder. It can be of two formats Xml and Excel format. This is used to provide the details of test configurations to be run.

* Config.xml
* Config file has **<testconfigurations>** as the root tag.
* User can provide a required test configuration that is child tag <testconfig>.

Once user specifies test configuration, user can run any of the test configurations using **run** attribute of **<testconfig>**. Setting it to true is done as shown below.

**<testconfig run=”true”>**

User can run desired testconfigurations by setting the **run** attribute as required.

Each **<testconfig>** tag has the following child tags.

* [<Application>](#_Application)
* [<Testcases>](#_Testcases)
* [<URL>](#_URL)
* [<Browser>](#_Browser)
* [<TestDataSource>](#_Test_Data_Source)

### Application

Application tag is used to define the name of application on which the tests are being run. Same tag value is used as the name of properties file which will be present in **Properties** folder.

### Testcases

**<testcases>** tag is used to provide required test cases to be run. User can test the scripts based on the following options.

* Test cases by range
* Test cases by feature
* Testtype (smoke / regression)
* Requirement they are mapped with
* Priority of the testcase
* Name of the testplan from testlink

**By Range**

User can specify the details of testcases to be executed by specifying a range as shown below.

<testcases>1-10,13- 15</testcases>

<testcases>1-5,7,9-11</testcases>

**By Feature / Module**

Testcases which belongs to a specific module can be executed by specifying the value for **<testcases>** tag as follows.

<testcases>fe:module1</testcases>

User also can specify the testcases to be executed from multiple modules at once, as shown below.

<testcases>fe:module1,module2,module3</testcases>

* + Names of the specified modules are case-sensitive.

**By TestType**

Testcases which belong to a specific testtype can be executed by specifying the value for **<testcases >** tag as shown below.

<testcases>tt:smoke</testcases>

User also can specify the testcases to be executed from multiple testtypes at once, as shown below.

<testcases>tt:smoke,regression</testcases>

* + Name of the testtype specified is case-sensitive.

**By Requirement**

Testcases which belong to a specific requirement can be executed by specifying the value for **<testcases >** tag as follows.

<testcases>rq:req\_id\_78</testcases>

User also can specify the testcases to be executed from multiple requirements at once, as shown below.

<testcases>rq: req\_id\_78, req\_id\_45</testcases>

* + Name of the requirement specified is case-sensitive.

**By Priority**

The testcases which belongs to a specific priority can be executed by specifying the value for the **<testcases >** tag as follows.

<testcases>pr:normal</testcases>

The user also can specify the testcases to be executed from multiple priorities at once, as shown below.

<testcases>pr:normal,high</testcases>

* + Name of the priority specified is case-sensitive.

**By Testplan**

Test cases specific to a testplan can be executed by specifying the name of test plan prefixed by ‘**tp**’. Testplan needs to be specified as follows and name of the testplan is always **case-sensitive.**

<testcases>tp:Testplan\_Login</testcases>

If nothing is specified, all the testcases present in the testcasemaster of specified datasource are executed.

### URL

**<URL>** tag is used to specify the URL of application on which the tests are to be executed. URL is to be given completely as shown below.

* **<URL>** <http://www.example.com/> [**</URL**](http://www.example.com/%20%3c/URL)**>** -- Valid URL
* **<URL>** [www.example.com](http://www.example.com/) **</URL> --** Invalid URL

|  |  |
| --- | --- |
| **Specified URL** | **$baseURL** |
| <http://www.example.com/login/> | [http://www.example.com](http://www.example.com/) |
| <http://www.example.com/> | <http://www.example.com/> |

**$baseURL** property in the framework is set based on the URL provided here. Prefix of the URL after the first ‘**/**’ is taken as the **$baseURL**.

### Browser

**<browser>** tag is used to specify the browser on which the tests are executed. The framework currently supports different browsers such as **IE, Firefox, Chrome** and **Android**. Theoretically it supports **Opera, Safari** and **HTMLunit** drivers also.

The value of browser tag varies while using the framework with or without **Selenium-gird**.

While using Selenium-grid, name of the browser must be same as the name specified in **Grid\_BrowserList.xml** file which is specified in **TestInputs** folder.

When using the framework without **Selenium-grid,** the value for the browser tag can be as listed below.

* IE – Internet Explorer
* FF – Firefox
* Chrome – Google chrome
* Safari – Safari
* Opera – Opera
* HTML – Html Unit Driver
* Android – Execution on a native android browser

Prerequisites for using **Android** browser are, apk file android driver must be installed in the emulator / device and the respective device must be in an unlocked state. If nothing is specified, the default value is taken as ‘**firefox**’.

### Test Data Source

Test data source, is used to specify the name of the datasource to be used for running the testcases. Datasource can be an xml file (or) excel file (or) a database. Datasource can be configured as shown below.

Files specified in the xls (or) xml format must be present in **Testinputs** folder.

* <testDataSource>PhpScheduler.xml</testDataSource>
* <testDataSource>PhpScheduler.xls</testDataSource>
* <testDataSource>DB</testDataSource>

If nothing is specified **DB** is taken as the default datasource.

## The Properties file

In detail, the properties file is an xml file which has the root tag as properties. The property file is used to define various required runtime configurations.

Major tags available in the property file are listed below.

* <appDetails>
* <common>
* <misc>
* <Seleniumgrid>
* <email>
* <testdataDB>
* <appDataDB>
* <TCMTool>
* <BugTrackerDetails>

### <appDetails>

This tag is used to specify the details regarding the application. Subtags are listed below.

* <logo>

<logo> tag is used to specify the logo of the application, and is used in HTML reports.

* <Prefix>

If we need to mention the prefix for the TestCaseIDs(PH1) then you should mention **<Prefix>** tag ,if you are not using Alphanumeric TestCaseID then no need to mention the **Prefix** tag. If nothing is specified, the default logo is taken as **ValueLabs** logo.

**Example**

<appDetails>

<logo>[http://www.example.com/images/logo.gif</logo](http://www.example.com/images/logo.gif%3c/logo)>

<Prefix>PH<Prefix/>

</appDetails>

### <common>

<common> tag is used to specify the common properties of the application in framework. It has the following subtags.

* executeFailedCases

<executeFailedCases> tag is used to specify whether the failed cases in a test run need to be re-executed or not. If this flag is set to **TRUE,** the failed cases are re-executed once and report is generated only for the failed cases.

* noOfIterations

If we set the executeFailedCases flag as **TRUE**, we can explicitly specify the noOfIteration we intended to repeate failed cases execution. By default framework executes the failed cases for one time.

* useGrid

<useGrid> tag is used to specify whether the testcases need to be executed using **Selenium-gird**  or not.

* reportFailureAsBug

<reportfailureAsBug> tag is used to set, whether the failed testcases are to be reported as Bugs to the bugtracking system or not.

* sendMailReport

<sendMailReport> tag is used to specify whether the testresults are to be sent as mail to the intended stake holders of the application or not.

* updateResultsDB

<updateResultsDB> tag is used to specify whether the testcase results are to be updated to the Database or not.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Usage** | **Default** | **Accepted** |
| **executeFailedCases** | Re-execute failed cases in current run | **FALSE** | **TRUE / FALSE** |
| **noOfIterations** | NoOfIteration to Re-execute failed cases in the run | **1** | **Numeric Value** |
| **useGrid** | Use Selenium Grid to launch Browser | **FALSE** | **TRUE / FALSE** |
| **reportFailureAsBug** | Report Failed cases as Bugs in bug tracking tool | **FALSE** | **TRUE / FALSE** |
| **sendMailReport** | Reporting the Results through e-mail | **TRUE** | **TRUE / FALSE** |
| **updateResultsDB** | Update testcase results to Database | **FALSE** | **TRUE / FALSE** |

**Example**

<common>

<executeFailedCases>false </executeFailedCases>

<noOfIterations>3</noOfIterations**>**

<useGrid>false</useGrid>

<reportFailureAsBug>false</reportFailureAsBug>

<sendMailReport>true</sendMailReport>

<updateResultsDB>false</updateResultsDB>

</common>

* + Even if the above sub tags are not specified, the default values are taken

### <misc>

<misc> tag is used to specify the miscellaneous properties of the application. It has the following subtags.

**reportPriority**

Reports the failed cases in HTML report generated, based on the priority. It gives a quick overview of the application status showing which priority cases that are failing.

**reportReq**

Reports requirement coverage done based on total cases executed and cases passed.

**useObjectRepo**

This is used to specify, whether the user needs to load object repository, which is defined in the corresponding data source

**clearCookies**

This tag is used to, clear the cookies and reload the URL specified in the config file. This is done twice if the flag is set to **TRUE.**

If the flag is set to **FALSE,** the URL is just loaded and no cookies are deleted.This process is done after the execution of each and every testcase

**closeBrowserSessions**

This tag is used to specify whether, all the existing browser sessions of the specified browser in config file are to be closed / not. This flag has no impact when the execution of testcases is done using **Selenium-gird**.

**detailedLogs**

This tag is used to specify whether, step wise status of execution is reported to database or not. If it is **TRUE,** each and every executed step is logged into the Database along with the timestamp

If this flag is set to **TRUE,** the value of **updateResultsDB** is set to **TRUE** automatically

**screenShotdir**

This flag is used to set, the directory location where the screen shots taken must be saved. If any directory is specified, the screenshots are saved in that particular directory else in the Screenshots directory of the Framework

**steplevelscreenshot**

This flag is used to set , whether screen shots are required for each and every step. If it is **TRUE,** screenshots for each and every step are taken

**createEvidence**

This flag is used to create the pdf reports for each and every testcase, by adding the screenshots and their respective step comments to a **PDF** file as an evidence of test done. If this flag is set to **TRUE,** the value of **createEvidence** is set to **TRUE** automatically

**createpdf**

This flag is used to create the pdf report, a pdf version of html report generated

**mergeReports**

This flag is used to merge the reports generated, for multiple threads of the same application. If this flag is pecified as **TRUE,** then the reports generated for the first run are integrated and then mailed based on the **sendMailReport** flag.

* + Reports generated for the failed cases are not used / integrated.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag Name** | **Usage** | **Default** | **Accepted** |
| **reportPriority** | Report the priorities of failed cases in HTML report generated | FALSE | TRUE / FALSE |
| **reportReq** | Report the requirement coverage done after the end of an execution | FALSE | TRUE / FALSE |
| **useObjectRepo** | Load the object repository | FALSE | TRUE / FALSE |
| **clearCookies** | Clear cookies and reload the URL after | FALSE | TRUE / FALSE |
| **closeBrowserSessions** | Close the browser instances of specified browser | FALSE | TRUE / FALSE |
| **detailedLogs** | Log step-wise execution to database with time | FALSE | TRUE / FALSE |
| **createEvidence** | Create PDF with screenshots as an evidence | FALSE | TRUE / FALSE |
| **steplevelscreenshot** | Take screenshots at each and every step | FALSE | TRUE / FALSE |
| **createpdf** | Generate PDF version of HTML report | FALSE | TRUE / FALSE |
| **mergeReports** | Integrate test results after completion of thread execution | FALSE | TRUE / FALSE |

**Example**

<misc>

<reportPriority> true </reportPriority>

<reportReq> true </reportReq>

<clearCookies> true </clearCookies>

<closeBrowserSessions> false </closeBrowserSessions>

<useObjectRepo> false </useObjectRepo>

<detailedLogs> true </detailedLogs>

<mergeReports> false </mergeReports>

<createpdf> true </createpdf>

<createEvidence> true </createEvidence>

<screenShotdir> Q:\Shared\Screenshots </screenShotdir>

<steplevelscreenshot> true </steplevelscreenshot>

</misc>

* + Even if the above sub tags are not specified, the default values are taken

### <SeleniumGrid>

<SeleniumGrid> tag is used to specify the URL and port number of the Selenium grid, if used

|  |  |  |
| --- | --- | --- |
| **Tag Name** | **Usage** | **Default Value** |
| **grid\_ip** | Hostname (or) ip of the machine where hub is running | **localhost** |
| **grid\_port** | Port number where the hub is running | **4444** |

**Example**

<SeleniumGrid>

<grid\_ip> 172.20.146.12 </grid\_ip>

<grid\_port> 6578 </grid\_port>

</SeleniumGrid>

* + Data is read from this tag only if the value of <useGrid> is true.

### <Email>

<Email> tag is used to configure the mailing list of the application. In this tag we can specify the sender, recipients, subject of mail, the body part and attachments. The attachments are optional, if they are specified they must be present in the **TestInputs** directory.

The following properties are required for it.

|  |  |
| --- | --- |
| **Property** | **Description** |
| <SMTP\_HOST\_NAME> | Name of SMTP Server |
| <SMTP\_PORT> | Port number for SMTP Server |
| <recipients> | List of reciepients seperated by ';' |
| <subject> | Subject of Mail |
| <from> | From Address that appears in mail |
| <attachment1> | Attachment to be used from TestInputs dir |
| <attachment2> | Attachment to be used from TestInputs dir |
| <message> | Message body having a certain text |

Message tag contains the body to be present in mail along with the text **andandCountersandand**. andandCounterandand is replaced by table data in script which has details of browser used for execution, No. of Total, Passed and Failed cases.

**Example**

<email>

<SMTP\_HOST\_NAME> **mail.example.net** </SMTP\_HOST\_NAME>

<SMTP\_PORT> 25 </SMTP\_PORT>

<recipients> [user1@example.net](mailto:user1@example.net); [user2@example.net](mailto:user2@example.net) </recipients>

<subject> Application1 Smoke Automation Report </subject>

<from> [user1@example.net](mailto:user1@example.net) </from>

<attachment1>TestData.zip </testDataAttachment>

<attachment2>Manual\_Checklist.xls</attachment2>

<message>andandCountersandand</message>

</email>

* + Data is not read from this tag only if the value of <sendMailReport> is false.

### <testdataDB>

<testdataDB> is used to define the database details where the testdata is stored. The following properties are required for it.

|  |  |
| --- | --- |
| **Property** | **Description** |
| <userName> | User name of the Database |
| <password> | Password for the Database user specified above |
| <driver> | Name of JDBC Driver |
| <url> | URL of the Database |
| <dbName> | Database name |

**Example**

<testdataDB>

<password> root </password>

<userName> root </userName>

<driver> com.mysql.jdbc.Driver </driver>

<url> jdbc:mysql://localhost:3306/ </url>

<dbName> application1db </dbName>

</testdataDB>

Data is read form this tag in any one of the following situations.

* If testdatasource is **Database**
* If the flag <**updateResultsDB**>is **TRUE**
* If the flag <**detailedLogs**>is **TRUE**

### <appDataDB>

<appDataDB> is used to define the database details of the application database. The following properties are required for it.

|  |  |
| --- | --- |
| **Property** | **Description** |
| <appDB\_URL> | URL of the Database |
| <appDB\_driver> | Name of JDBC Driver |
| <appDB\_userName> | User name of the Database |
| <appDB\_password> | Password for the Database user specified above |
| <app\_DatabaseName> | Database name |

**Example**

<testdataDB>

<appDB\_password>root</appDB\_password>

<appDB\_userName>root</appDB\_userName>

<appDB\_driver>com.mysql.jdbc.Driver</appDB\_driver>

<appDB\_URL> jdbc:mysql://localhost:3306/</appDB\_URL>

<app\_DatabaseName>schedule</app\_DatabaseName>

</testdataDB>

### <BugTrackerDetails>

<BugTrackerDetails> tag is defined to store the details of the BugTracking tools like Bug Tracking tools like Bugzilla, Mantis, Jira etc,.

We define the properties that are necessary as part of tool because different tools require different properties. As of now we had updated them for BugZilla.

Bugs are logged, for a testcase if there is no existing Bug, in any of the **bug** **transition states** except **CLOSED**. Comments are added for an already existing bug irrespective of its state

The following properties are required for BugZilla access.

|  |  |
| --- | --- |
| **Property** | **Description** |
| <bt\_url> | URL of the Bugzilla server |
| <bt\_UserName> | Username for Bugzilla access |
| <bt\_Password> | Password for provided user |
| <productName> | Name of the product for which bugs are locked |
| <componentName> | Name of the component under product |
| <version> | version of the Product / component |
| <op\_sys> | Operating System under which it is tested |
| <platform> | Platform on which it is tested |
| <priority> | Priority specified while locking bug |

**Example**

<BugTrackerDetails>

<bt\_url> [http://Server/](http://server/) </bt\_url>

<bt\_UserName> [auto\_bugs@example.net](mailto:auto_bugs@example.net) </bt\_UserName>

<bt\_Password> [automation@bugs](mailto:automation@bugs) </bt\_Password>

<productName> TestProduct </productName>

<componentName> TestComponent </componentName>

<version> unspecified </version>

<op\_sys> Other </op\_sys>

<platform> All </platform>

<priority> P2 </priority>

</BugTrackerDetails>

* + Data is read from this tag only if the value of <reportFailureAsBug> is true

### <TCMTool>

<TCMTool> tag is defined to store the details of the Test Case Management tools like TestLink, Testopia etc.

We define properties that are required as part of tool because different tools require different properties. As of now we had updated them for TestLink.

The following properties are required for TestLink access.

|  |  |
| --- | --- |
| **Property** | **Description** |
| <APIKEY> | 32- bit Key generated in TestLink |
| <tcm\_URL> | URL of the TestLink server |
| <projectName> | Project Name in which application testcases are Configured |

**Example**

<TCMTool>

<APIKEY> e441lit1b411d3fc3adlid5lid2873e76 </APIKEY>

<tcm\_URL> <http://server/lib/api/xmlrpc.php> </tcm\_URL>

<projectName> PHPScheduler </projectName>

</TCMTool>

# Detailed Usage of Keywords

Actions to be used in the testdata sheet are treated as keywords. Each keyword has three parameters defined as listed below.

* Name

Name of the action to be performed.

* Data Value

Text / value to be used as input.

* Data Field

Field on which the operation is performed.

The following shown below is the detailed usage of each and every keyword

## Web Keywords

Web keywords relate to the commonly used web actions like typing, selecting, verification etc. The following section describes each and every web keyword

### Check

This keyword is used to perform a check operation unto a combo box. To check a combo box, details are listed below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | CHK<locator> |  | Check |
| **Example** | CHKremember\_me |  | Check |

### Clear

This Keyword is used to perform clear operation into a textfield. To clear a textfield details are listed below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TXT<locator> |  | Clear |
| EDT<locator> |  | Clear |
| **Example** | TXTuser\_id\_login |  | Clear |
| EDTdesc\_box |  | Clear |

### Clearenter

This keyword is used to perform a clear and type operation into a textbox. To type into a textbox, details are listed below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TXT<locator> | <value to be typed> | Clearenter |
| EDT<locator> | <value to be typed> | Clearenter |
| **Example** | TXTuser\_id\_login | user\_admin | Clearenter |
| EDTdesc\_box | This is a automation message | Clearenter |

For textbox, data values can be given as shown below.

* Plaintext  
  Example : **username**
* Encrypted data

Example : **DECPT6vCDrt0/p0I=**

* Random Value appended with a plain text

Example : **RNDusername**

* Random String appended with a plain text

Example : **RNDSTRusername**

* Value already stored in a HashMap

Example : **HMVusername**

* Value already stored in a HashMap appened with the user given data

Example **: <HMVusername>+<user defined data>+<HMVcountry>**

* Value from the Data Sheet

Example : **dt:username#3**

* Value from the templatecase Data Sheet

Example : **key:Login::username#4**

### Clearsession

This keyword is used to delete all the domain cookies and reload the URL. The following is the format to be used.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | **EMPTY** | **EMPTY** | Clearsession |
| **Example** | **EMPTY** | **EMPTY** | Clearsession |

### Click

This keyword is used to perform a click operation on various web elements like buttons, links, alert boxes etc. The following format is to be used, for normal web elements and alert box.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | LNK<locator> | **EMPTY** | Click |
| BTN<locator> | **EMPTY** | Click |
| XPH<locator> | **EMPTY** | Click |
| IMG<locator> | **EMPTY** | Click |
| JSC<locator> | **EMPTY** | Click |
| ALT | **EMPTY** | Click |
| **Example** | LNKLog Out | **EMPTY** | Click |
| BTNsubmit\_details | **EMPTY** | Click |
| XPH//div[@id='header']//a | **EMPTY** | Click |
| IMG//img[@class='logo'] | **EMPTY** | Click |
| JSCsubmit\_prefs | **EMPTY** | Click |
| ALT | **EMPTY** | Click |

For confirmation boxes, it should be

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** | **Description** |
| **Syntax** | CNFCancel | **EMPTY** | Click | Reject the confirmation |
| CNF | **EMPTY** | Click | Accept the confirmation |
| **Example** | CNFCancel |  | Click |  |
| CNF |  | Click |  |

### Clickandwait

For CLICKANDWAIT, same holds as of Click but data values are to be filled with the time for page sync.

### Closewindow

This keyword is used to close a window with the specified title. The format to be used is as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TTL | <title\_of\_window> | Close window |
| **Example** | TTL | Print - Schedulers | Close window |

For title, data values can be given as shown below

* Plain text

**Example:** **Home Page - Title**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

The user can make use of regular expressions to have the window titles

* Title to match at the start of string : ^(<title>).\*
* Title to match at the end of string : .\*(<title>)$
* Title to match in the middle of string : .\*(<title>).\*
* Title to match the complete string : <title>

### Draganddrop

This keyword is used to perform a drag and drop operation on a specified set of field locators

It requires the source and destination locators. The format to be followed is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <src\_locator>:<dest\_locator> | **EMPTY** | draganddrop |
| **Example** | //input[@id='1']://input[@id='12'] | **EMPTY** | draganddrop |

### Enter

This keyword is used to perform a type / sendkeys operation on a

* Text box
* Button (any html element capable of taking input)

To type into a textbox or a button, below is the format.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TXT<locator> | <value to be typed> | Enter |
| EDT<locator> | <value to be typed> | Enter |
| BTN<locator> | <value to be typed> | Enter |
| **Example** | TXTuser\_id\_login | user\_admin | Enter |
| EDTdesc\_box | This is a automation message | Enter |
| BTNlanguage | Eng | Enter |

For textbox, data values can be given as shown below.

* Plain text

**Example:** **username**

* Encrypted data

Example : **DECPT6vCDrt0/p0I=**

* Random Value appended with a plain text

**Example:** **RNDusername**

* Random String appended with a plain text

Example : **RNDSTRusername**

* Value already stored in a HashMap

**Example:** **HMV**username

* Value already stored in a HashMap appened with the user given data

Example **: <HMVusername>+<user defined data>+<HMVcountry**

* Value from the Data Sheet

**Example: dt:username**#3

* Value from the templatecase Data Sheet

**Example:** **key:Login::username**#4

### Goback

This keyword is used to perform a brower back operation I.e returning to a previous page. The format to be used is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | **EMPTY** | **EMPTY** | Goback |
| **Example** |  |  | Goback |

### Isenabled

This keyword is used to verify whether a particular webelement is Enabled or not. The format to be used is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<locator> | **EMPTY** | Isenabled |
| BTN<locator> | **EMPTY** | Isenabled |
| IMG<Locator> | **EMPTY** | Isenabled |
| **Example** | XPHbtn\_save |  | Isenabled |
| BTNlogin\_submit |  | Isenabled |
| IMGlogo |  | Isenabled |

### Isdisabled

This keyword is used to verify whether a particular webelement is Disabled or not. The format to be used is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<locator> | **EMPTY** | Isdisabled |
| BTN<locator> | **EMPTY** | Isdisabled |
| IMG<Locator> | **EMPTY** | Isdisabled |
| **Example** | XPHbtn\_save |  | Isdisabled |
| BTNlogin\_submit |  | Isdisabled |
| IMGlogo |  | Isdisabled |

### Keypress

This keyword simulates a keypress event over a give locator. Use the below format to keypress over a given web element.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <locator> | <key\_to\_be\_pressed> | Keypress |
| **Example** | password | Enter | Keypress |

The key press events can include all the keys identified by their **Unicode** code points.

### Mouseover

This keyword is used to perform a mouse over operation on the specified locator. Use the below format to keypress over a given web element.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <locator> | **EMPTY** | Mouseover |
| **Example** | password |  | Mouseover |

### Openurl

This keyword navigates to the desired URL, wherever required, in the window which is under selection by default. Use the below format to open a URL.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <url\_to\_be\_navigated> | **EMPTY** | Openurl |
| **Example** | $baseURL/login.jsp |  | Openurl |

For openurl, the data **fields** can be given as shown below

* Plain URL

**Example:** <http://www.qa00.example.com/>login.jsp

* Value already stored in a HashMap

**Example:** **HMVurl\_login**

* Value from the Data Sheet

**Example: dt:URL\_login#3**

* Value from the templatecase Data Sheet

**Example: key:Login::url\_login#4**

* Value including the baseURL

**Example: $baseURL/login.jsp**

**Refresh**

### Refresh

This Keyword is used to perform refresh current page. To refresh current page the format is as follows.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | **EMPTY** | **EMPTY** | Refresh |
| **Example** | **EMPTY** | **EMPTY** | Refresh |

### Select

This keyword is used to perform a select operation on a

* dropdown box or
* an iframe or
* a radio button

To select a dropdown, the format is as follows

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | SLB<locator> | <value to be selected> | Select |
| **Example** | SLBlanguage | English US | Select |

For dropdown data values can be given as shown below.

* Plain Visible Text from dropdown

**Example:** **English US**

* Plain Visible Text from dropdown, but need to be selected using javascript

**Example:** **js:English US**

* Value of the option in dropdown

**Example:** **value=en-us**

* Index of the option from dropdown

**Example: index=6**

* Random Value from dropdown

**Example:** **RND**

* Value already stored in a HashMap

**Example:** **HMV**selectValue1

* Value from the Data Sheet

**Example: dt:s**electValue1#3

* Value from the templatecase Data Sheet

**Example: key:Login::s**electValue1#4

To select an iframe, the format is as follows

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | FRM | <Frame to be selected> | Select |
| **Example** | FRM | Left | Select |
| **Example** | FRM | index=1 | Select |
| **Example** | FRM | //frame[@name=’right’] | Select |
| **Example** | FRM | css= frameset#fs1 frame | Select |

For dropdown data values can be given as shown below.

* Name (or) id of the frame

**Example:** **left**

* Index of the iframe , starting form '0' (zero)

**Example:** **index=6**

* Xpath of the Frame

**Example: //frame[@name=’left’]**

* cssPath of the Frame

**Example:** **css= frameset#fs1 frame**

To select a radio button, the format is as follows

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | RDB<locator> | **EMPTY** | Select |
| **Example** | RDBgender\_male |  | Select |

* + The same datavalues and datafields are required for SELECTANDWAIT.

### Selectandwait

For SELECTANDWAIT, the same holds as of **SELECT**.

### Selectwindow

This keyword is used to select a window with the specified title. The format to be used is as.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TTL | <title\_of\_window> | Selectwindow |
| **Example** | TTL | Print - Schedulers | Selectwindow |

For title, data values can be given as shown below.

* Plain text

**Example: Home Page - Title**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

The user can make use of regular expressions to have the window titles

* Title to match at the start of string : ^(<title>).\*
* Title to match at the end of string : .\*(<title>)$
* Title to match in the middle of string :.\*(<title>).\*
* Title to match the complete string : <title>
  + If title specified is empty, the current window handled is selected

### Storeattribute

This keyword is used to store the specified attribute into a hash map with a key. Use the below format to store an attribute.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <locator>**@**<attrribute> | <key\_to\_be\_used\_to\_store> | Storeattribute |
| **Example** | //a[@name='user']@text | username\_link | Storeattribute |

To use the store attribute, it must be referred by **HMV<key\_to\_be\_used\_to\_store>**

Example: **HMV**username\_link

This allows the value to be re-used from the hashmap

### Storevalue

This keyword is used to store the values of the webelements in hashmap with a desired key. It can store values from the following fields.

* Text present in the text box
* Selected value in dropdown
* Text from a webelement ( Link / Buttton / any other element )
* Text from a particular cell in table identified by row and column
* Value from the database using query

The store value command can be used as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TXT<locator> | <key\_to\_be\_used\_stored> | Storevalue |
| STV<Data to store in the HashMap> | <key\_to\_be\_used\_stored> | Storevalue |
| COB<locator> | <key\_to\_be\_used\_stored> | Storevalue |
| XPH<locator> | <key\_to\_be\_used\_stored> | Storevalue |
| LNK<locator> | <key\_to\_be\_used\_stored> | Storevalue |
| BTN<locator> | <key\_to\_be\_used\_stored> | Storevalue |
| TBL<Locator> | <row>:<column>:<key> | Storevalue |
| DBV<query> | <column\_name>:<key> | Storevalue |
| **Example** | TXTusername\_reg\_txt | Username\_register | Storevalue |
| STVadminuser | Username | Storevalue |
| COBCountry | country\_selected | Storevalue |
| XPH//div[@id='content'] | content\_text | Storevalue |
| LNK//a[@id='user\_id\_profile'] | user\_prof\_id | Storevalue |
| BTNrating\_stars | rating\_saved | Storevalue |
| TBL@id='dependants' | 2:1:dependant1 | Storevalue |
| DBVselect \* from emp where id=6 | Sal:sal\_id | Storevalue |

For **TBL**, any unique attribute of the locator must be given as input. The table is identified by xpath as shown below.

//table[<locator>]/tr[<row>]/td[<column>]

The stored values can be accessed from Hashmap using **HMV**<key>.

### Uncheck

This keyword is used to perform a uncheck operation unto a combo box. Use the following format to uncheck a combo box.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | CHK<locator> | **EMPTY** | Uncheck |
| **Example** | CHKremember\_me |  | Uncheck |

### Verify

This keyword is used to perform different types of verification operations as given below. Use the following format to verify title of page.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TTL | <title\_of\_the\_page> | Verify |
| **Example** | TTL | Home Page | Verify |

For title verification, data values can be given as shown below.

* Plain text

**Example: Home Page – Example.com**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

To following is the format to verify text in a text box

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TXT<locator> | <text\_expected> | Verify |
| **Example** | TXTusername | user@example.com | Verify |

To verify the URL's of a html link, then

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | LNK<locator>@href | <url\_expected> | Verify |
| **Example** | LNKLog in@href | https://www.example.com/login | Verify |

To verify the text present in a html element,

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<locator> | <text\_expected> | Verify |
| RDB<locator> | <text\_expected> | Verify |
| CHK<locator> | <text\_expected> | Verify |
| TBL<locator> | <row>:<col>:<text\_expected> | Verify |
| MSG | <text\_expected> | Verify |
| ALT | <text\_expected> | Verify |
| CNF | <text\_expected> | Verify |
| **Example** | XPH//div[@id='header']/h1 | Welcome to Home Page | Verify |
| RDBmale | m | Verify |
| CHKrem\_me | TRUE | Verify |
| TBL1 | 1:2:Name | Verify |
| MSG | It is a group policy | Verify |
| ALT | You are logged out!! | Verify |
| CNF | Are you sure? | Verify |

For text verification, data values can be given as shown below.

* Plain text

**Example: username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

* + For XPH, RDB and CHK value attribute is verified if text is mismatched.

To verify the presence of a html element,

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | BTN | <locator> | Verify |
| IMG | <locator> | Verify |
| LNK | <locator> | Verify |
| **Example** | BTN | Log In | Verify |
| IMG | //img[@class='logo'] | Verify |
| LNK | Home | Verify |

To verify the selected values in a dropdown

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | SLB<locator> | <value\_to\_be\_verified> | Verify |
| COB<locator> | <value1>|<value2>|.... | Verify |
| **Example** | SLBlanguage | English US | Verify |
| COBpref\_mode | Mail|Phone|Call | Verify |

For value verification, data values can be given as shown below.

* Plain text

**Example:** **Selected1**

* Value from the Data Sheet

**Example: dt:select\_pref#3**

* Value from the templatecase Data Sheet

**Example: key:Login::select\_pref#4**

Use the below format to verify existing values in a Hashmap, using GET.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | GET<hashmap\_key> | TXT<locator> | Verify |
| GET<hashmap\_key> | LNK<locator> | Verify |
| GET<hashmap\_key> | XPH<locator> | Verify |
| **Example** | GETuser\_id | TXTusername | Verify |
| GETpage\_header | XPH//div[@id='header']/h1 | Verify |
| GETuser\_profile\_link | LNKProfile | Verify |

### Verifyattribute

This keyword is used to verify the specified attribute with specified value.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <locator>**@**<attrribute> | <value\_to\_be\_used> | Verifyattribute |
| **Example** | //a[@name='user']@text | user\_admin | Verifyattribute |

For verifyattribute, data values can be given as shown below

* Plain value to be verified

**Example:** **user\_id\_342**

* Value already stored in a HashMap

**Example:** **HMVuserid**

* Value from the Data Sheet

**Example: dt:username#3**

* Value from the templatecase Data Sheet

**Example: key:Login::username#4**

### Verifycontinue

The same datavalues and datafields as of **verify** are used, except the step is marked as passed even on failure and a warning is generated.

### Verifynotpresent

This keyword is used to perform different types of verify operations which are used to test that the element doesn't exists. Use the below format to use verify title of page.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | MSG | <text\_not\_expected> | Verifynotpresent |
| **Example** | MSG | It is a group policy | Verifynotpresent |

For text verification, data values can be given as shown below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

Use the below format to verify that the html element does not exist.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | BTN | <locator> | Verifynotpresent |
| LNK | <locator> | Verifynotpresent |
| **Example** | BTN | save\_cancel | Verifynotpresent |
| LNK | Log In | Verifynotpresent |

Use the below format to verify that the specified element does not exist in a dropdown.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | CBS<locator> | <text\_not\_expected> | Verifynotpresent |
| **Example** | CBSCountry | New York | Verifynotpresent |

For value verification, data values can be given as shown below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

### VerifyObject

This keyword is used to verify object presence in the page. The format to be used is show below

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | BTN | <locator> | Verifyobject |
| TXT | <locator> | Verifyobject |
| LNK | <locator> | Verifyobject |
| **Example** | BTN | save\_cancel | Verifyobject |
| TXT | username | Verifyobject |
| LNK | Log In | Verifyobject |

### 

### Waitforelement

This keyword is used to wait for a specific amount of time until an element appears. The format to be used is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<locator> | <time\_to\_wait\_in\_ms> | waitforelement |
| LNK<locator> | <time\_to\_wait\_in\_ms> | waitforelement |
| TXT<locator> | <time\_to\_wait\_in\_ms> | waitforelement |
| IMG<locator> | <time\_to\_wait\_in\_ms> | waitforelement |
| COB<locator> | <time\_to\_wait\_in\_ms> | waitforelement |
| **Example** | XPH//div[@id='loc\_search'] | 45000 | waitforelement |
| LNKAdd More People | 45000 | waitforelement |
| TXTname\_pref | 45000 | waitforelement |
| IMG//img[@class='delete'] | 45000 | waitforelement |
| COBstate\_us | 45000 | waitforelement |

Use the following format for title and message verifications.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TTL<title\_to\_appear> | <time\_to\_wait\_in\_ms> | waitforelement |
| MSG<message\_to\_appear> | <time\_to\_wait\_in\_ms> | waitforelement |
| **Example** | TTLAdd Member | 45000 | waitforelement |
| MSGProfile Update Success | 45000 | waitforelement |

For title and message, **datavalues** can be given as listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

## Generic Keywords

Generic keywords are the keywords intended to make use of the framework in an efficient manner.

### Executetestcase

This keyword is designed to make reuse of the existing test steps and independent templatecases. The format to be used for executing steps from testcase is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | **EMPTY** | <tcid> : < start> - <end> | executetestcase |
| **Example** |  | 1:1-5 | executetestcase |

The format to be used for executing steps from templatecases

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | setup#<template\_row\_id> | <template\_case\_name> | executetestcase |
| **Example** | Setup#1 | Login | executetestcase |
| Setup#1 | Login:2-4 | executetestcase |

### Storecomments

This keyword is used to store comments in the form of

“<**datafield> :: <datavalue>“**

This is included in the last column of Excel report generated. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | HMV<hashmap\_key> | <comment\_type\_value> | Storecomments |
| TXT<locator> | <comment\_type\_value> | Storecomments |
| **Example** | HMVuser\_id | User ID in Hashmap | Storecomments |
| TXTemail | Email ID | Storecomments |

For HMV, a value present in the Hashmap is taken and is appended as comment.

For TXT, the text present in specified field is appended as comment.

### Waittime

This keyword is used to pause the execution for specified number of milliseconds. Use the below format to wait for a given amount of time.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | **EMPTY** | <time\_to\_wait\_in\_ms> | WaitTime |
| **Example** |  | 45000 | WaitTime |

For waittime, the datavalues can be given as shown below.

* Time in ms

**Example:** **35000**

* Value already stored in a HashMap

**Example:** **HMVwait\_time**

* Value from the Data Sheet

**Example: dt:wait\_time#3**

* Value from the templatecase Data Sheet

**Example: key:Login::wait\_time#4**

## Conditional and Looping Keywords

Conditional keyword is used to perform conditional operations within test steps. Looping keyword is used to perform a loop, i.e. Re-execute the required sequential steps for a required no. of times

### If

This keyword is used to perform a condition check along with **verify** keyword. It needs to be specified in the condition column of the **datasheet**.

* The default format to be specified is if **:{pass\_step}:{fail\_step}**.
* The execution is moved to the step number specified in **{pass\_step}**  if verify command returns a true value else to **{fail\_step}**

The user can make use of below two predefined values.

* **Next** – Move execution to next step
* **End** – Move execution to end of the test case

The format to be specified is given as below.

| **Condition** | **Description** |
| --- | --- |
| **If:Next:6** | If verify is pass, move to **next** step else **6th** step |
| **If:Next:End** | If verify is pass, move to **next** step else **last** step |
| **If:3:End** | If verify is pass, move to **3rd**  step else **last** step |
| **If:3:Next** | If verify is pass, move to **3rd**  step else **next** step |
| **If:End:6** | If verify is pass, move to **last** step else **6th**  step |
| **If:3:6** | If verify is pass, move to **3rd** step else **6th**  step |

### Loop

This keyword is used to perform a loop operation over a series of sequential steps, only if the action in the specified step returns true.

The default format to be specified is as follows.

loop:{start\_step}-{end\_step}:{no\_of\_iterations}

Once a loop is executed, the execution continues from next step where it has started looping. If there is any failure in loop, the complete testcase is treated as a failure. The format to be specified is given as below.

|  |  |
| --- | --- |
| **Condition** | **Description** |
| **loop:3-5:2** | Loops the test steps **3-5** for **2** times |
| **loop:3-3:7** | Loops the test steps **3** for **7** times |

## Webtable Keywords

The web table keywords are defined to deal with table contents in a web page. They include various verification contexts. But for each table keyword defined, row number in which the specified data is present, saved in Hashmap with the specified data.

### Columndatacheck

collaborating

This keyword is used to verify the contents of a table in one column with respect to another column, initiating search from a particular row as specified. The format to be used for TBL is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row\_start>:<column\_no>:  <column\_value>:  <column\_no\_target>: <column\_value\_target> | Columndatacheck |
| **Example** | TBL@id='prof' | 3:3:Dad:4:Male | Columndatacheck |

In this, the datavalue fields represent the following.

* <row\_start> - Initate search from specified row number
* <column\_no> - Column in which data is to be found
* <column\_value> - Column data to be matched
* <column\_no\_target> - Target column to be matched
* <column\_value\_target> - Target column value to be matched

For columndatacheck, data values for **<column\_value>** and **<column\_no\_target>** are listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

The format to be used for XPH is shown below.

|  |  |  |
| --- | --- | --- |
|  | **Data Fields** | **DataValues** |
| **Syntax** | XPH<Parent>::<Source>::<Target> | <div\_start>:  <source\_value>: <dest\_target> |
| **Example** | XPH//div[@id='ui']//div:://li[3]:://li[1] | 2:Build:Automation |

In this, the datafields represent the following.

* <Parent> - Parent tag, which has to be searched
* <Source> - Tag which matches the text to <source\_value>
* <Target> - Tag which matches the text to <dest\_target>
  + <Source> and <Target> are childs with respect to <Parent>.

In this, the datavalues represent the following

* <div\_start> - Initate search from specified element
* <source\_value> - Source Value to be matched
* <dest\_target> - Target element value to be matched

For columndatacheck, data values for **<source\_value>** and **<dest\_target>** are listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

* + The row at which the specified data is found is saved in Hashmap, whose key is the text give as part of <column\_value>.

### Verifysort

This verifies that the specified elements in a table like structure are in chronological order. The format to be used to verify sort order for **TBL** is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <start\_row\_no>:<col\_no>:  <order>:<format> | verifysort |
| **Example** | TBL@id='in\_time' | 3:4:Asc:DateHH:MM:SS | verifysort |

The format to be used to verify sort order for **XPH** is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<Parent>::<Child> | <start\_row\_no>:  <order>:<format> | verifysort |
| **Example** | XPH//div[@id='test']//div:://h2 | 4:Asc:DateHH:MM:SS | verifysort |

The values for order can be as listed below.

* **ASC –** Increasing / Ascending order
* **DSC –** Decreasing / Descending order

The values of format can be as listed below.

* **Num –** Numbers type of Date
* **STR –** String type of Data
* **Date –** Date type of Data

While using **Date,** date format used by the data must be specified. If nothing is specified default date format is **Mmddyy**.

### Verifytabledata

Verifies the cell value and clicks a particular column in that row, if needed. The format to be used to verify a particular cell value for **TBL** for **CHECK** is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row\_start>:<col\_no>:  <text\_to\_match>**:check** | Verifytabledata |
| **Example** | TBL@id='in\_time' | 3:2:Login:check | Verifytabledata |

The format to be used to verify a particular cell value for **TBL** for **CLICK** is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row\_start>:<col\_no>:  <text\_to\_match>:**click**:<col\_no\_click> | Verifytabledata |
| **Example** | TBL@id='in\_time' | 3:2:Login:click:9 | Verifytabledata |

For verifytabledata, data values for **<text\_to\_match>** can be given as listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

* + The row at which the specified data is found is saved in Hashmap, whose key is the text give as part of <text\_to\_match>.

The format to be used to verify a particular cell value for **XPH** for **CHECK** is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<Parent>::<Child> | <row\_start>:  <text\_to\_match>**:check** | Verifytabledata |
| **Example** | XPH//div[@id='test']//div:://li[3] | 3:Login:check | Verifytabledata |

The format to be used to verify a particular cell value for TBL for CLICK is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | XPH<Parent>::<Child>::<Child\_to\_click> | <row\_start>:  <text\_to\_match>:**click** | Verifytabledata |
| **Example** | XPH//div[@id='test']//div:://li[3]::li[5] | 3:Login:click | Verifytabledata |

### Verifytablerowexist

Verifies whether a particular cell has a specific value or not. The format to be used to verify that particular value exists in Cell is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row\_start>:<column>:<value>:**true** | Verifytablerowexist |
| **Example** | TBL@id='in\_time' | 3:4:Yes:true | Verifytablerowexist |

The format to be used is when to verify that particular value does not exist in Cell is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row\_start>:<column>:<value>:**false** | Verifytablerowexist |
| **Example** | TBL@id='in\_time' | 3:4:No:false | Verifytablerowexist |

In this, the datavalue fields represent the following details.

* <row\_start> - Row no. From which search is initiated
* <column> - Column no. in which data is to be found
* <value> - Value to be matched with

For verifytablerowexist, data values for **<value>** can be as listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

### Verifyvalueinrow

This verifies the cell value of a table with given value. The format to be used to verify a particular cell value is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <row>:<column>:<text\_expected> | Verifyvalueinrow |
| **Example** | TBL@id='in\_time' | 03:04:11 AM | Verifyvalueinrow |

For verifyvalueinrow, data values for **<text\_expected>** can be given as listed below.

* Plain text

**Example:** **username**

* Value from the Data Sheet

**Example: dt:title\_home#3**

* Value from the templatecase Data Sheet

**Example: key:Login::title\_home#4**

* Value from the Hashmap, which is already stored

**Example: HMVUsername**

* Value already stored in a HashMap appened with the user given data

**Example : <HMVusername>+<user defined data>+<HMVcountry**

## DB Keywords

### DB\_update

This keyword is used to perform an update query in a DB. An update query can be an **insert / delete / update query**. The format to be used is as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY<sql\_query> | **EMPTY** | db\_update |
| **Example** | QRYinsert into emp (ID,name) values (1,'Jhon') |  | db\_update |
| QRYupdate emp set ID=9 where name='Jhon' |  | db\_update |
| QRYdelete from emp where ID=9 |  | db\_update |

### DB\_verifycolumns

This keyword is used to verify the columns data obtained on executing the specific query. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY<sql\_query> | <column\_name1>=<column value>; <column\_name2>=<column value> | db\_verifycolumns |
| **Example** | QRYselect \* from emp; | name=Jhon | db\_verifycolumns |
| QRYselect \* from emp; | name=Jhon;id=65;sal=10000 | db\_verifycolumns |

For verification, data values can be given as shown below.

* Single Column

**Example:** **name=Jhon**

* Multiple Columns

**Example: name=Jhon;id=65;sal=10000**

* + Columns in DataValues should be seperated with semi colon (;).

### DB\_verifytablecontent

Verifies that specified columns against the table in the application. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | TBL<Locator> | <start\_row>:<end\_row>:<no of columns>:<query> | db\_verifytablecontent |
| **Example** | TBL@id='table1' | 1:4:3:select \* from test | db\_verifytablecontent |

* + Columns in DataValues should be seperated with colon (:).

### DB\_verifyduplicatevaluesonprimarykey

This keyword is used to verify that duplicate values are not allowed into the primary key. This keyword performs two insert operations as per query and checks that duplicate values are not allowed. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY | <sql\_query> | db\_verifyduplicatevaluesonprimarykey |
| **Example** | QRY | insert into emp (`ID`) values (1) | db\_verifyduplicatevaluesonprimarykey |

### DB\_verifynotnullconstraint

This keyword is used to verify that null values are not allowed into the specified columns. This keyword performs insert operation as per query and checks that null values are not allowed. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY | <sql\_query> | db\_verifynotnullconstraint |
| **Example** | QRY | insert into emp (`ID`) values (NULL) | db\_verifynotnullconstraint |

### DB\_verifyprimarykeys

This keyword is used to verify that specified columns are primary keys. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY<table\_name> | <column1>:<column2>:.... | db\_verifyprimarykeys |
| **Example** | QRYemp | id:ref\_no | db\_verifyprimarykeys |

For verification, data values can be given as listed below.

* Single Column

Example: **ID**

* Multiple Columns

Example: **id;ref\_no**

* + Columns in DataValues should be seperated with semi colon (;).

### DB\_Verifyrowscount

This keyword is used to verify the number of rows returned with the expected count on executing the specific query. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | QRY<sql\_query> | <expected rows count> | db\_verifyrowscount |
| **Example** | QRYselect \* from emp; | 5 | db\_verifyrowscount |

* + Columns in DataValues must be numeric.

### DB\_Verifyspexecutiontime

This keyword is used to execute the stored procedure and verify the execution time of a SP with the expected execution time passed by the user. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | SPH<sp\_name>::<expected\_execution\_time> | <set of i/o parameters>DB\_verifyspexecutiontime | DB\_verifyspexecutiontime |
| **Example** | SPHadd::10 | 10,20,? | DB\_verifyspexecutiontime |

* + Expected execution time in data values should be given in seconds.

### DB\_Verifystoredprocedure

This keyword is used to verify the execution of a stored procedure based on the procedure name and the input / output parameters. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | SPH<sp\_name> | <set of i/o parameters> | DB\_verifystoredprocedure |
| **Example** | SPHadd | 10,20,? | DB\_verifystoredprocedure |

Data values can be given as listed below.

* Set of input and output parameters

**Example: 10,20,?**

* + Output parameters should be mentioned as '?' in the data values column.

## REST Keywords

### RESTGET

This keyword is used to get the response from the REST web services. Performs the assertions (NULL and ! NULL). Stores the JSON Object value. The format to be used is given below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <Input Media Type>||<End Point>||<Cookie Flag> | <Status code to be verified>||NULL=<The value1 to be verified as equal to Null>,<The value2 to be verified as equal to Null>,..........||!NULL=<The value1 to be verified as equal to null.>,<The value2 to be verified as equal to null.>,.......||<Key1> =<Object field to be stored>, <Key2> =<Object field to be stored>, ............ | RESTGET |
| **Example** | application/json||http://metaapist.airtrade.com/lookup/SearchLocations/FlightHotel||NULL | 200||NULL=jsonObj.\_links.selectpackage.href,||!NULL=jsonObj.\_links.self.href,||DestinationLocationCode=jsonObj.Options[0].Code, | RESTGET |
|  |  |  |  |

### RESTPOST

This keyword is used to post the request to REST web services and retrieve the response. Performs the assertions (NULL and ! NULL). Stores the JSON Object value. The format to be used is given below

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <Input Media Type>||<End Point>||<Input Message>||<Cookie Flag> | <statuscode>::<Status code to verified>||<NULL>::<The value1 to be verified as equal to Null>,<The value2 to be verified as equal to Null>,..........||<!NULL>::<The value1 to be verified as equal to null.>,<The value2 to be verified as equal to null.>,.......||<store::Key::Value to be stored> | RESTPOST |
| **Example** | application/json||http://metaapist.airtrade.com/shoppingcart | <statuscode>::200||<!NULL>::jsonObj.\_links.self.href | RESTPOST |
|  |  |  |  |

### RESTPUT

This keyword is used to put the request to the REST web services and retrieve the response. Performs the assertions (NULL and ! NULL). Stores the JSON Object value. The format to be used is given below

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Fields** | **DataValues** | **Action** |
| **Syntax** | <Input Media Type>||<End Point>||<Input Message>||<Cookie Flag> | <statuscode>::<Status code to verified>||<NULL>::<The value1 to be verified as equal to Null>,<The value2 to be verified as equal to Null>,..........||<!NULL>::<The value1 to be verified as equal to null.>,<The value2 to be verified as equal to null.>,.......||<store::Key::Value to be stored> | RESTPUT |
| **Example** | application/json||http://metaapist.airtrade.com/HMVSessionIdentifier$/TravelInsuranceProduct/HMVTravelInsuranceBookingIdentifier$||{"SelectedTravelInsurancePolicyCode":"Base"}||Null | <statuscode>::200|| <store::ShoppingCartIdentifier::$Location.lastIndexof("/")+1> | RESTPUT |
|  |  |  |  |

## Using the Testdata Parameters

The testData parameters are used by ‘dt:’ prefix. To use a parameter defined in testdata, it is defined as

dt:<parameter\_name>#<index\_value>

If the testdata is specified as follows, the corresponding data is taken.

* dt:<parameter\_name>the value with index 1 is specified by default.
* dt:<parameter\_name>#RND**,** a random value is taken.
* dt:<parameter\_name>#<index>**,** the value with specified index is taken.

If the execution process is in a **loop**, then we can specify the data as given below.

dt:<parameter\_name>#auto

The value of the parameter is taken starting at index 1, for round 1 of the loop and is incremented by 1 for next rounds of looping. If the execution process is not in a loop, the index is treated as 1. For templatecase testdata, we can load the required testdata row by specifying its index as follows.

setup#1

The user can also load an arbitary random row by specifying the testdata parameter as

setup#RND

### Using Date as Parameter

The date parameters are prefixed by ‘**d:**’. To use a date parameter it must be defined as given below.

d:<date\_req>:<change\_by\_type>:<change\_by\_value>:format:<date\_format>

<date\_req> parameter takes the following values.

* **currentdate,** returns current date
* **effectivedate,** returns date for the month first
* **monthend,** returns date for month end

<change\_by\_type> parameter takes the following values.

* **M,** changes date by specified no. of months
* **d,** changes date by specified no. of days
* **y,** changes date by specified no. of years

Default format of date is **mm/dd/yyyy**. Some of the date formats are listed below with returning values.

|  |  |
| --- | --- |
| **Date Format specified** | **Returning value** |
| **d:currentdate** | Current Date |
| **d:currentdate:M:3** | Current date + 3 months |
| **d:currentdate:M:-3** | Current date - 3 months |
| **d:effectivedate:d:3** | 1st date of currentmonth + 3 days |
| **d:monthend:y:3** | Current month end date + 3 years |
| **d:monthend:d:3:format:EEE, MMM d, ''yy** | Current month end date + 3 days in specified format |

### Using RND Parameter

**RND** parameter is used to generate random digits as per requirement. Details of this parameter with examples are listed in the below table.

|  |  |  |
| --- | --- | --- |
| **Format specified** | **Example** | **Returning value** |
| **RND** | **RND** | Randomly generated 5 digit number |
| **RND:<no\_of\_digits>** | **RND:6** | Randomly generated number, with specified no. of digits |
| **RND<text>** | **RNDadmin** | Randomly generated 5 digit number is appended at the end of specified text |
| **RND:<no\_of\_digits>:<text>** | **RND:3:admin** | Randomly generated number, with specified no. of digits is appended at the end of specified text |

## Adding New Keywords

* Create a new Project in Eclipse IDE
* Copy the AppTestType.java file from Framework and paste it on the src folder in IDE
* Add all the available jar files in thr Framework folder to BuildPath of Project
* Open AppTestType.java and make sure that no errors exists in the Project

**Steps to add new keyword**

* Define the Name of the Keyword as given below.

public enum AppKeyWords{ LOGIN }

* Under switch statement, write a case as follows.

case LOGIN:

login(webdriver,fieldText,value,fieldName);

break;

* Define the login Function within a try-catch block, and write the code as follows

public void login(WebDriver webdriver, String fieldText,String value, String fieldName) {

try{

/\*\* Code for Login \*\*/

resultDetails.setFlag(true);

}catch(Exception e){

resultDetails.setErrorMessage(e.getMessage());

}

}

* The user can make use of stored variables by using the following code

String storedVal = tt.getValue(“<VALUE\_OF\_THE\_KEY>”);

* The parameters for perform Actions are defined as given below.

WebDriver webdriver:: The WebDriver object

String fieldText :: The Datafields parameter in TestData sheet

String value :: The Datavalues Parameter in TestData Sheet

String fieldName :: The fieldname based on action performed

* + Copy the library files, if newly added any into the Framework folder.

### Using Framework Methods in AppTestType

Following are the methods that we can use in AppTestType file

#### Enter:

Method to type or enter into a DataField

**Syntax:**

|  |
| --- |
| tt.enter(WebDriver webdriver, String fieldText, String value, String fieldName) |

**Examples:**1.tt.enter(webdriver,"XPH//input[@id='username']","admin","Username");

2.tt.enter(webdriver,"XPH//input[@id='username']","dt:username","Username");

3.tt.enter(webdriver,"XPHobj:username","admin","Username");

#### GetDate:

Method to get required Date

**Syntax:**

|  |
| --- |
| tt.driver.utils.getDate(String value) |

**Examples:**

tt.driver.utils.getDate("d:currentdate");

|  |  |
| --- | --- |
| Date Format specified | Returning value |
| ***d:currentdate*** | Current Date |
| ***d:currentdate:M:3*** | Current date + 3 months |
| ***d:currentdate:M:-3*** | Current date - 3 months |
| ***d:effectivedate:d:3*** | 1st date of currentmonth + 3 days |
| ***d:monthend:y:3*** | Current month end date + 3 years |
| ***d:monthend:d:3:format:EEE, MMM d, ''yy*** | Current month end date + 3 days in specified format |

#### GetFieldFromRepo:

Method to get the field form Repository

**Syntax:**

|  |
| --- |
| tt.driver.utils.getFieldFromRepo(String field) |

**Examples:**

tt.driver.utils.getFieldFromRepo("obj:username");

#### GetValue:

Method to get Values from HashMap / TestData sheet

**Syntax:**

|  |
| --- |
| tt.driver.utils.getValue(String value) |

**Examples:**

1.tt.driver.utils.getValue("HMVusername");

2.tt.driver.utils.getValue("RNDusername");

3.tt.driver.utils.getValue("dt:password");

#### Select:

Method to select a value from Dropdown

**Syntax:**

|  |
| --- |
| select(WebDriver webdriver, String fieldText, String value,String fieldName) |

**Example:**

tt.select(webdriver, "SLBlanguage", "English US", "Language Selection");

#### hMap Usage:

For Storing the variable in hMap and using the stored value in across the test cases.

**Syntax:**

|  |
| --- |
| driver.hMap.put(paramObject, paramV) |
| driver.hMap.get(paramObject) |

**Example:**

tt.driver.hMap.put("username", "admin");//stroing

tt.driver.hMap.get("username");//retrieving

#### waitForPageToLoad:

Waits for the page to load completely

**Syntax:**

|  |
| --- |
| WebDriverUtils.waitForPageToLoad(WebDriver webdriver, String timeOut) |

**Example:**

WebDriverUtils.*waitForPageToLoad*(webdriver, "2000");

where 2000 is time in milliseconds

*Note:* timeOut should be in milliseconds

#### selectWindow:Switches the focus to the window specified

***Syntax:***

|  |
| --- |
| *WebDriverUtils.selectWindow(WebDriver webdriver, String windowTitle)* |

***Example:***

WebDriverUtils.*selectWindow*(webdriver,"Title of the page");

#### waitForElementToPresent:

To wait for the element to be present

***Syntax:***

|  |
| --- |
| *WebDriverUtils.waitForElementToPresent(WebDriver webdriver,*By byLocator, int seconds) |

**Example:**

WebDriverUtils.*waitForElementToPresent*(webdriver, By.*id*("usename"), 1000);

#### SelectByIndex:

Selects the given option based on its index

**Syntax:**

|  |
| --- |
| WebDriverUtils.selectByIndex(WebDriver webdriver, String locator,int option) |

**Example:**

WebDriverUtils.*selectByIndex*(webdriver,"//select[@id='country]", 2);

#### mouseOver:

Places mouse pointer over an element

**Syntax:**

|  |
| --- |
| WebDriverUtils.mouseOver(WebDriver webdriver, String locator) |

**Example:**

WebDriverUtils.mouseOver(webdriver, "//select[@id='country]")

#### locatorToByObj:

Returns the locator type(By id, xpath, css, class, *linkText, cssSelector, name, className)*

**Syntax:**

|  |
| --- |
| WebDriverUtils.locatorToByObj(WebDriver webdriver, String locator) |

**Example:**

WebDriverUtils.locatorToByObj( webdriver, "//select[@id='country]")

#### WebElement:

this method makes the driver wait until the element is present **Syntax:**

|  |
| --- |
| WebDriverUtils.getWebElement(WebDriver webdriver,By byLocator, int seconds) |

**Example:**WebDriverUtils.isElementPresent(WebDriver webdriver,By.id("usename"), 2)

#### getSelectedOptions:

Returns the selected options of a combo box/list  
  
**Syntax:**

|  |
| --- |
| WebDriverUtils.getSelectedOptions(WebDriver webdriver,String locator) |

**Example:**  
WebDriverUtils.getSelectedOptions( webdriver, "//select[@id='country]")

#### getSelectedLabel:

Returns the selected label of a combo box/list **Syntax:**

|  |
| --- |
| WebDriverUtils.getSelectedLabel (WebDriver webdriver,String locator) |

**Example:**

WebDriverUtils.getSelectedLabel( webdriver, "//select[@id='country]")

#### getAlert:

Returns the message of the alert/confirmation box

**Syntax:**

|  |
| --- |
| WebDriverUtils.getAlert(WebDriver webdriver) |

**Example:**WebDriverUtils.getAlert(webdriver)

#### getAvailableOptions:

Returns all available options present in the list/combo box

**Syntax:**

|  |
| --- |
| WebDriverUtils.getAvailableOptions(WebDriver webdriver,String locator) |

**Example:**

WebDriverUtils.getAvailableOptions(webdriver,"//select[@id='country]")

#### isElementPresent:

**Method1:**  
waits for an element to be present and returns true if the element is

present, else returns false

**Syntax:**

|  |
| --- |
| *WebDriverUtils*isElementPresent(WebDriver webdriver,By byLocator, **int** seconds) |

**Example:**

WebDriverUtils.isElementPresent(WebDriver webdriver,By.id("usename"), 2)

**Method2:**  
returns true if the element is present, else returns false  
**Syntax:**

|  |
| --- |
| WebDriverUtils.isElementPresent(WebDriver webdriver, String locator) |

**Example:**WebDriverUtils.isElementPresent(WebDriver webdriver,By.id("usename"))

## On Fail Procedures

If the user wants to execute, a specific set of steps on a test case failure, he can provide the required set of steps as a function in **AppTestType.java,** whose method name is same as that of application name.

If the application name is **PHPScheduler,** the function in AppTestType, must be defined as follows.

public void PHPScheduler ( WebDriver webdriver) {

try {

// Actions to be performed on failure

} catch (Exception e) {

}

}

## Using Function Driven Framework

### Config File

**config** file is present in the **TestInputs** folder. It can be of two formats xml and xls format. It is used to provide the details of the test configurations to be run.

**Config.xml**

config file has **<testconfigurations>** as the root tag. The user can proved the required test configuration is the child tag **<testconfig>**.

Once the user specifies test configurations, user can run any of the test configurations using **run** attribute of **<testconfig>** by setting it to true as given below.

<testconfig run=”true”>.

The user can run the desired testconfigurations by setting the **run** attribute as required. Each **<testconfig>** tag has the following child tags.

* <Application>
* <testcases>
* <URL>
* <browser>
* <testDataSource>

#### Application

The Application tag is used to define the name of the application on which the tests are being run.

#### Testcases

**<testcases>** tag is to provide the required test cases to be run. It should provide the name of the functions to be executed seperated by **comma** (,).

User can specify the details of testcases to be executed by specifying name of functions as given below.

<testcases>test\_Login</testcases>

<testcases>test\_Login,test\_Feature</testcases>

The user can also make use of the regular expressions to match with test cases.

<testcases>.\*</testcases> – Execute all the test cases

<testcases> (test\_)(.\*) </testcases>

* + The names of the functions specified are case-sensitive.

#### URL

**<URL>** tag is used to specify the URL of application on which the tests are to be executed. URL is to be given completely, as shown below.

<URL> <http://www.example.com/> [</URL](http://www.example.com/%20%3c/URL)> -- Valid URL

<URL> [www.example.com](http://www.example.com/) </URL> -- invalid URL

#### Browser

**<browser>** tag is used to specify the browser on which the tests are executed. The framework currently supports different browsers such as **IE, Firefox, Chrome** and **android.** Theoretically it supports **Opera, Safari** and **HTMLunit** drivers also.

* IE – Internet Explorer
* FF – Firefox
* Chrome – Google chrome
* Safari – Safari
* Opera – Opera
* HTML – HTML Unit Driver
* Android – Execution on a native android browser

The prerequisites for using **android** browser are, the apk file android driver must be installed in the emulator / device and the respective device must be in an unlocked state. If nothing is specified, the default value is taken as ‘**firefox**’.

#### Testdatasource

Test datasource is used to specify the name of class in which the functions are present. Name of the datasource specified is case sensitive and name of the complete package must be specified while running the tests. If no package name is specified, the default package name ‘**tests**’is taken into account.

<testDataSource>org.scheduler.Login </testDataSource>

<testDataSource>Login</testDataSource>

<testDataSource>Login.java</testDataSource>

* + The class name specified is case-sensitive.

### Guidelines to Write the Function Driven Tests

* The class should extend the predefined com.java.GeneralLibrary.

**Example**

public class TestScheduler extends com.java.GeneralLibrary

* User should define the test methods with the prefix **test\_** as shown below

**Example**

public void test\_login( ) {

}

* The user can call **takescreenshot()** method to take a screen shot at any instance of the framework
* User should not perform quit() operation on webdriver object, which results to failing of the next tests
* The user has no acces to instantiate the Webdriver object across, the framework
* The default package to be used is **tests**
* The user can call **pass(“<msg>”),**  to mark the test case as pass, on which the **<msg>** is appended to the report.

**Example**

public void test\_login( ) {

/\*\* code for test \*\*/

pass(“ User is able to login “);

}

* The user can call **fail(“<msg>”),** and can return back to mark the test case as fail, on which the **<msg>** is appended to the report.

**Example**

public void test\_login( ) {

/\*\* code for test \*\*/

if(<condition>) {

fail(“ User is able to login with invalid details “);

return;

}

}

* User can use **getValue()**, method as shown below.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **key:<parameter>#<index>** | The parameter from the current method and index |
| **key:<parameter>#RND** | The parameter from the current method and random index |
| **key:<method>::<param>#<index>** | The parameter from the specified method and index |
| **key:<method>::<param>#RND** | The parameter from the specified method and random index |

* The name of the excel file to be used, for input test data must be present in the **TestInputs** folder and is same as the name of the application
* The name of the xml property file to be used, must be present in the **Properties** folder and is same as the name of the application
* The feature to execute **onFail** procedures is provided at class level. Each and every class can have a method, as shown below which executes on failure of the executed function.

**Example**

public void onFail( ) {

/\*\* code for onFail condition \*\*/

}

### The Properties File

The properies file is the same name as the application file, and the following properties are accepted, while remaining are ignored.

* <sendMailReport>
* <useGrid>
* <clearCookies>
* <closeBrowserSessions>
* <createpdf>
* <screenShotdir>

## Encrypted data creation

By executing the **PasswordEncryption** batch file in the Framework it will be asked to Enter string to be encrypted un on entering the data, it is going to generate the encrypted data in **decryptContent** text file of TestInputs folder.

In order to give the encrypted data in the DataSheet, user need to give as **DECPTencrypteddata** in Data Values column.