Open2Test Test Automation Framework for OpenScript - Tips

Version 1.0

January 2010

TABLE OF CONTENTS

1. Introduction

1.1. Purpose

This "Tips & Tricks" document provides an overview of handling frequently encountered scripting problems and some valuable do's and don'ts to maximize the productivity of the Open2Test Test Automation Framework.

1.2. Scope

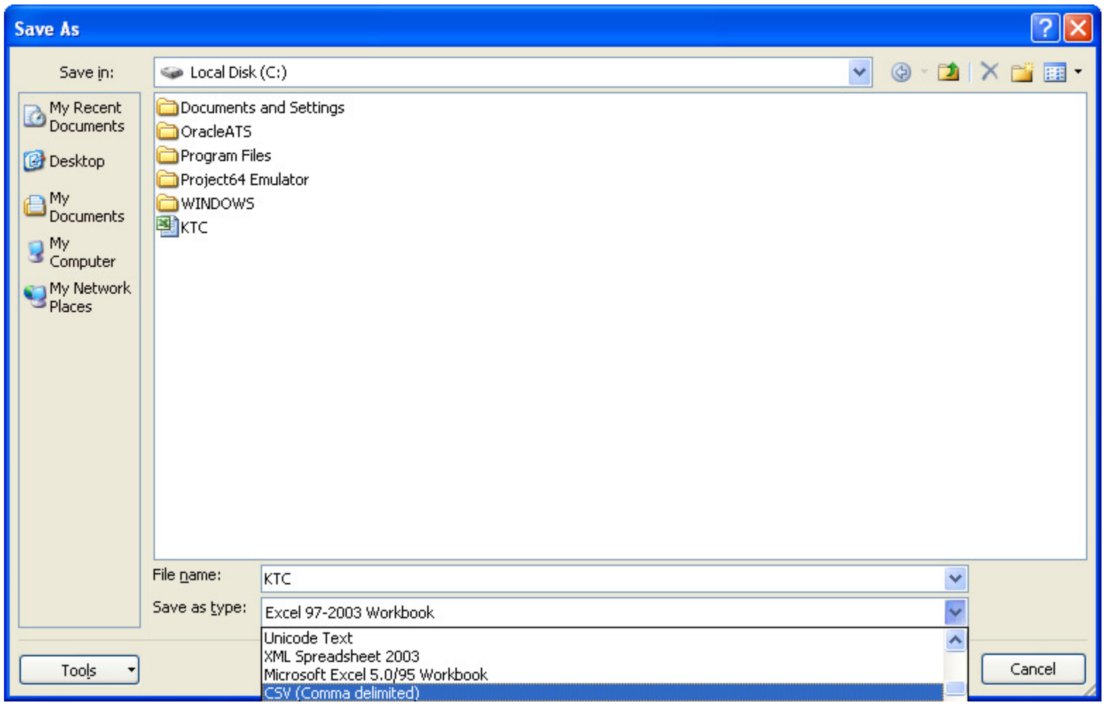
This document is just for understanding and reference. There might be a need to tweak the solutions provided before implementation, depending on the actual scenario in hand.

This document requires prior knowledge and working experience with the Open2Test Test Automation Framework. For understanding the keywords and syntaxes, please refer to the "Scripting Standards" document.

2. CSV file

The Open2Test Test automation framework supports csv format as the file type for the Keyword Test Case. A csv file can be accessed in Microsoft Excel, making the most use of the features of Microsoft Excel Workbook. Users might have difficulty creating the csv file and accessing it using Microsoft Excel. The steps below describe the steps involved:

1. Create and open a blank Microsoft Excel Workbook.
2. Select 'File → Save As' option from the menu bar.
3. From the 'Save as type' dropdown, select the option 'CSV (Comma delimited) and save the file with the required name.
4. Open the saved CSV file and create the Keyword Test Case and Databank file as required.



Note: In a csv file, users can have only one sheet. So save the details of Keyword Test Case and Databank file in the first sheet. If a user attempts to create a second sheet, details of the first sheet will be replaced with the details of the second sheet.

3. Using Variables

Open2Test Test Automation Framework supports the use of two kinds of variables in its implementation: Runtime variables, and databank variables. To use the values of runtime variables and databank variables in test execution, follow the steps below:

3.1. Runtime Variables

There might be a situation in test execution, where using the value generated in previous step in test steps is required. During those circumstances, the user can store the value of the respective parameter in a variable and then use it in the required step.

To retrieve the value of the run time variable from the dictionary object, the user needs to place the character '#' before the variable name, as in the example below:

Line	A	B	C	D	Remarks
10	r	Storevalue	Textbox;reftextbox	Value;var1	Captures the value present in the text box in a variable 'var1'.
11	r	Msgbox	#var1		Populates the value of the variable 'var1' in results view.

Notes:

1. Observe the presence of character '#' before the variable name while retrieving.
2. Variables are case-sensitive; 'var1' differs from 'Var1'.

3.2. Databank Variable

Open2Test Test Automation Framework supports parameterization, and the user can use the values of the databank variable in test execution, as illustrated below:

Line	A	B	C	D	Remarks
10	r	Perform	Textbox;refTextbox	Settext;dt_Val1	Val1 is the name of the configured databank variable.

Notes:

1. To use databank in OpenScript, the script should be configured with the respective databank file using 'Script -> Configure Databank' menu of the OpenScript.
2. Databank variables should place the character 'dt_' before the corresponding variable name.

4. Conditions

Conditions are basically implemented when we need a specific code to be executed only when a condition is satisfied. For example, we would want a set of steps to be executed if a value of the variable 'x' is negative.

Syntax – r |condition|<Var A>;<condition>|<Var B>

. . . (Set of test steps)

Endcondition

4.1. Problem Scenarios

At times you may encounter the following problem scenarios during the implementation of conditions using the Open2Test Test Automation Framework.

4.1.1. Handling nested conditions

At times the flow might not be restricted to just one condition but may instead require multiple conditions before the code is executed. In other words, at times nested conditions might be required. In such cases, the second condition should be placed after the first condition and should fall within the range of the first condition.

The following example in **Table 1** shows a simple way to create a nested condition in Open2Test Test Automation Framework.

Line	A	B	C	D	Remarks
10	r	Condition	VarA;equals	varB	Beginning of First condition
11	r	Condition	VarB;notequals	varC	Beginning of Second condition
12	r	Msgbox	varA		
13	r	endCondition			End of Second Condition
14	r	MsgBox	varB		
15	r	endCondition			End of First Condition

Table 1: Nested Conditions

5. BaseState

It is very important that the automation scripts developed should not have any dependency on the state the application is in. It should be able to execute from any screen in the application. The importance of this is that when these scripts run in a suite and a script fails, the following script might not find the application in the home screen. In such circumstances, the second script should not fail because it did not find the application in the required state.

The best way to resolve this is to have code in place that would take the application to the main screen before beginning the execution of any script. One way to do this is to use a Reusable Action and call this Reusable Action at the first and last line of each script. For best results, this code can also be included in the Error Handling functions.

The example in **Table 3** shows the call to BaseState in the first and last lines of the script. The BaseState function can be associated with certain parameters, if required, for additional flexibility.

Example:

Line	A	B	C	D	Remarks
1	r	Callaction	RA_BaseState	Param	**BaseState function call.**
2	r	Keyword	Keyword	Keyword	**Script Body**
10	r	Keyword	Keyword	Keyword	
15	r	Callaction	RA_BaseState	Param	**Base State function call.**

Table 3: BaseState

6. Message

Maintenance is an integral part of the automation lifecycle. Once we have developed the scripts, we must maintain them to keep the script updated and in sync with the application.

The message box functionality is handy for maintaining scripts. You can use the "MsgBox" keyword to display the value of the required variable in 'Results Pane' during the execution of the script. This, to some extent, helps in pinpointing the error points during execution.

In the example in '**Table 4**', at line number 10, 'Value' property of a 'textbox' object with the name 'TName' is stored in a variable 'strValue' for later use at line 12 where it is being checked to be equal to "Smith". During maintenance, we can actually see what is getting stored in the variable '#strValue' by inserting a simple 'message' keyword with '#strValue' as the argument.

Line	A	B	C	D	Remarks
10	r	Storevalue	Textbox;TName	value:strValu e	
11	r	Message	#strValue		Comment s Message will be display ed on Results pane

Table 4: Message Box 'msgbox' Keyword

7. Loops

Loops are a useful tool when we have to execute code repeatedly for a given set of data. This section describes how to use the looping keywords in the Open2Test Test Automation Framework to construct a loop.

Scenario: In this scenario, we have to create a set of users for the AUT. This is a repetitive job. The loop keyword of the Open2Test Test Automation Framework has been designed to handle such scenarios.

Consider a scenario in which it is required to loop over a certain set of test steps three times in a test case. This scenario can be implemented by using the following keyword test steps:

Line	A	B	C	D	Remarks
10	r	Loop	3		**Loop Keyword**
11	r	Keywords	…	…	
12	r	Keywords	…	…	
13	r	endLoop			

Table 6: Loops

If it is necessary to implement nested loops, this can be done by using the following keyword test steps:

Line	A	B	C	D	Remarks
10	r	Loop	3		**Beginning of first loop**
11	r	Loop	5		Beginning of second loop
12	r	Keywords			
13	r	Endloop			End of first loop
14	r	Keywords			
15	r	endLoop			End of second loop

8. Iterate

There might be a situation in which the test case needs to be iterated for a specified number of times with the values of the databank. Under those circumstances, the user can follow the steps below:

Line	A	B	C	D	Remarks
10	r	Loop	5		**Beginning of first loop**
11	r	Keywords			
12	r	Keywords			
13	r	Getnextdb			Navigates to next record in databank
14	r	Endloop			End of the loop

If the user needs to iterate over the test steps with all the available values of the databank, the following test steps can be followed:

Line	A	B	C	D	Remarks
10	r	Function	File;linecount	C:/DB.CSV;Dbcount	Finds the length of the databank.
11	r	Loop	#Dbcount		**Beginning of first loop**
12	r	Keywords			
13	r	Keywords			
14	r	Getnextdb	DB		Navigates to next record in databank ('DB' is the Alias name of the databank)
15	r	Endloop			End of the loop

9. Save Data to Databank

During test execution, there might be a situation in which the user is required to save the value of the parameter generated in one script for use in another script. Under those circumstances, the user can save the dynamically generated data of the first script in a .csv file and then use the saved csv file as a databank in the second script.

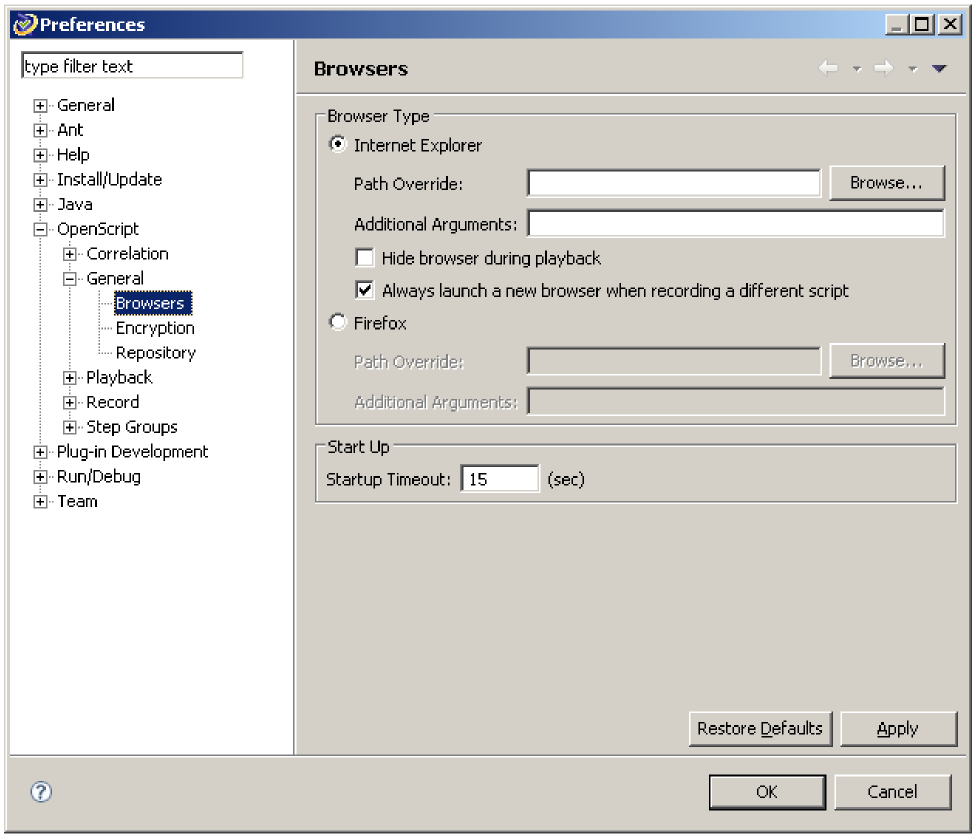
Line	A	B	C	D	Remarks
10	r	Storevalue	Textbox;reftextbox	Value;var1	Captures the value present in the text box in a variable 'var1'.
11	r	Function	File;append	C:\DB.csv;#var1	Saves the value of the variable var1 in a csv file DB which can be used as a data bank file in the required script.

10. Switching Browsers

Open2Test Test Automation Framework supports test execution in both Internet Explorer and Mozilla Firefox.

To switch from one browser to another, Browser preferences needs to be changed in OpenScript as mentioned below:

1. From OpenScript menu bar, choose 'Window → Preferences'.
2. From the 'Preferences' window, select 'OpenScript → General → Browser'.
3. From the 'Browser Type' section, choose either 'Internet Explorer' or 'Firefox' as required.



Browser Type
(●) Internet Explorer
Path Override:
Additional Arguments:
[ ] Hide browser during playback
[✓] Always launch a new browser when recording a different script
( ) Firefox
Path Override:
Additional Arguments:

Start Up
Startup Timeout: 15 (sec)

11. Manual Test Description

In the Keyword Test Case sheet, 'Manual Test Description' constitutes the fifth column, and is next to the 'Action Value' column. The user can describe the keyword action performed on the step in this column that would be reflected in the 'Results' report as a test step.

r	Perform	browser;*about:blank*	navigate;www.yahoomail.com	Load Yahoo.com
r	Perform	browser;*Yahoo!*	waitforpage;15	Wait until system loads the yahoo page
r	Perform	textbox;Uname	Settext;TestOne	Enter user name as 'TestOne'
r	Perform	textbox;Pwd	settext;Testing	Enter password as 'Testing'
r	Perform	Button;login	Click	Click on the 'Login' button

12. Cannot Get Connection From Helper Error

Sometimes the system will throw an error ‘Cannot get connection from Helper error’ in Oracle OpenScript while trying to record a flow or during playback. During those times, follow the steps below to fix the issue:

1. Exit out of OpenScript
2. Close all instances of Firefox and Internet Explorer.
3. Run ‘UninstallBrowserHelpers’ batch file from ‘C:\OracleATS\OpenScript’.
4. Delete the folder ‘Oracle IE ToolBar’ from the directory ‘C:\OracleATS\OpenScript’.
5. Run ‘InstallBrowserHelpers’ batch file from ‘C:\OracleATS\OpenScript’.

After performing the steps above, OpenScript should behave as expected without throwing any kind of error.

13. Label – Jump To

In test execution, there might be a situation in which the user needs to perform some set of operations when certain events occurs. For example, when the system displays a certain kind of error message, the user needs to jump to the required error handler section to properly handle the error. During those circumstances, the 'Label – Jump To' keyword comes in handy and serves the purpose.

Line	A	B	C	D	Remarks
10	r	Storevalue	Element;elemdompath	Innertext;check1	Stores the value of the inner text at the specified dom path.
11	r	Condition	Check1;like	Error 102	Checks whether the stored value has text 'Error 102' on it.
12	r	Assignvalue	Eflag	1	Assigns variable eFlag with the value 1.
13	r	JumpTo	Errhandler		Control jumps to the label Errhandler.
14	r	Endcondition			
		Keywords	…		
		Keywords	…		
22	r	Label	Errhandler		
23	r	Condition	Eflag;equal	1	
24	r	Keywords	…		
25	r	Keywords	…		
26	r	Endcondition			

Also, there might be a situation in which the user needs to redo certain test steps when a certain condition exists:

Line	A	B	C	D	Remarks
10	r	Label	TestOne		Defining a label in Test

					Case.
11	r	Keywords	…		
12	r	Keywords	…		
13	r	Condition	Var1;lessthan	50092	Verifying the condition
14	r	JumpTo	TestOne		If condition passes, control jumps to TestOne label.
15	r	Endcondition			
16	r	Keywords	…		
17	R	Keywords	…		

14. User-Defined Function

The keyword-driven framework will support the majority of automation tool functions. But there might be a situation when the user cannot use the keyword-driven framework, or when recorded script might be more appropriate or useful. Under those conditions, the user can make use of the UDF (user-defined function) class in the Open2Test Test Automation Framework.

To make use of user-defined functions in the Test Automation Framework, follow the steps below:

1. Record the required flow using OpenScript.

2. From the recorded script, isolate the required generated code in 'Java Code' view.

3. Download and open the UDF class from Open2Test.com.

4. Create a function (with the visibility 'private') with the required name in UDF and paste the isolated code from the recorded script.

5. If any parameters need to be passed for the function, define them as required.

6. Modify the name of the services as per the naming conventions of UDF class.

   ('UDF_Wds' for 'web', UDF_Br for 'Browser', 'UDF_fts' for 'ft' and 'UDF_Ut' for 'utilities'.)

7. If the function has a return value, store the return value in a string variable 'retval' of UDF class.

Line	A	B	C	D	Remarks
10	r	Callfunction	UserDefined;Par1:Par2	Retval	Calling the user-defined function of UDF class which has the parameters par1, par2 and whose return value is stored in a variable Retval
11	r	Message	#Retval		