

Hybrid Test Automation Frameworks Implementation using QTP

Pallavi Patwa

"When developing our test strategy, we must minimize the impact caused by changes in the applications we are testing, and changes in the tools we use to test them."

--Carl J. Nagle

Abstract:

Various white papers have been published on Test Automation framework however how to implement the framework is the puzzle often faced by Test Automation Developer. This article provides overview of how the hybrid test automation framework can be implemented using QTP with example.

Need for Automation Framework

- With the onset and demand for rapidly developed and daily deployed build, test automation is crucial.
- Test Automation is kind of development activity. And for the most part, testers have been testers, not programmers.
- To have faster cycle time for development of test automation with less expertise, use of application-independent test automation framework becomes inevitable.

Automation Frameworks

- Various automation frameworks are available viz
 - Data Driven Automation Frameworks
 - Keyword Driven Automation Framework
 - Modular Framework
 - Hybrid Test Automation (or, "All of the Above")

Implementation of Hybrid Test Automation framework using QTP is discussed in this article

Hybrid Test Automation Framework Architecture

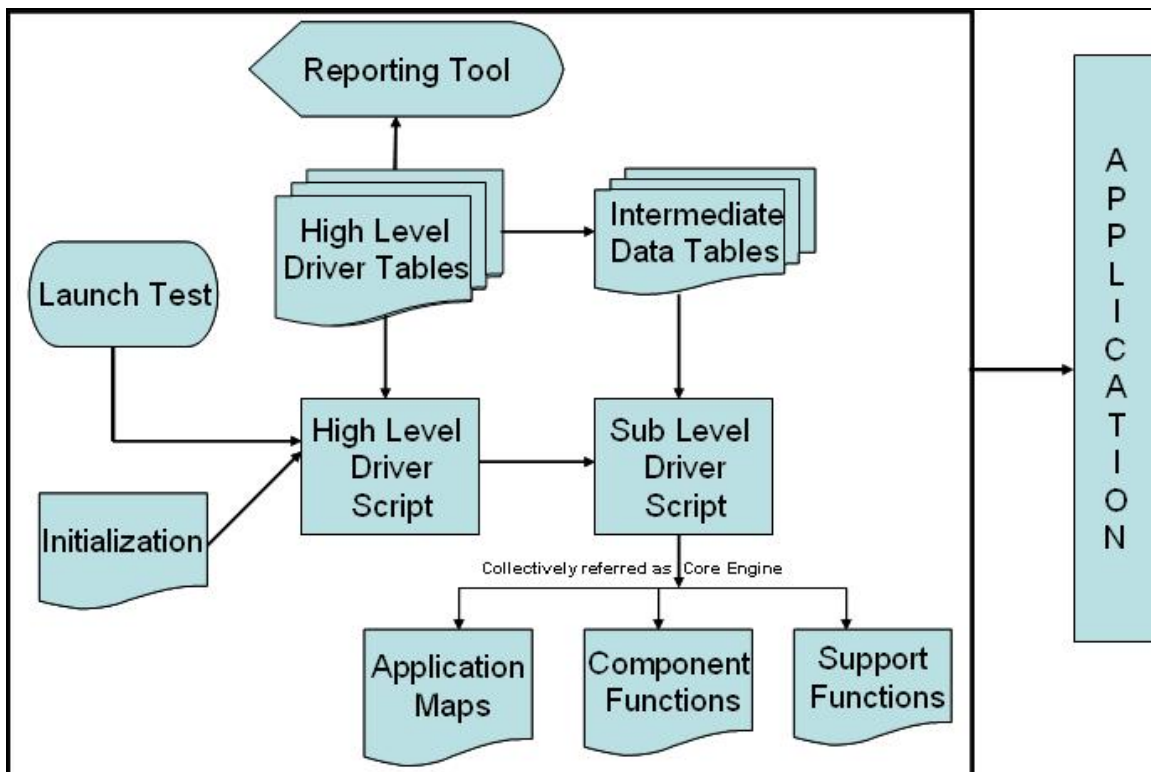


Diagram 1

Let us start with *Intermediate tables* which are based on keyword or test driven approach. Here the entire process is data driven including functionality. The keyword controls the processing.

Spreadsheet has been used as a mean for Intermediate data table. The records in data table contain the keywords that describe the actions we want to perform. They also provide any additional data needed as input to the application.

Consider following example

- Invoke browser, provide username / password & click sign on.
- Check if user have entered correct username /password, if not instruct to provide correct username/password.
- After providing correct username and password, click 'Yes' on the warning screen to proceed further.
- In end check if user has successfully logged in home page.

For above test case, consider following data table record as seen in Table 1

Component	Browser	Object	ObjectName	Action	Param	Expected	CheckPoint
Web		Browser		Invoke	http://www.t estautomati onframework .com/		
Web	Login	WebEdit	USER	Set	1231234		
Web	Login	WebEdit	PASSWORD	Set	#XZ77\$		
Web	Login	WebButton	Sign On	Click			
Web	Login	WebElement	html tag = TD, class = error- message	VerifyMinorError	innertext	Invalid Login	Invalid Login Please Provide Correct EIN & Password in DataTable
Web	Warning Screen	WebButton	Yes	Click			
Web	Home Page	WebElement	html tag = SPAN, class = x-tab-strip- text	VerifyProperty	innertext	Home Page	Login to Home Page

Table 1

Brief explanation of Table 1:

Based on keyword contained in Component column i.e. Web, the Web related functions will be called.

Any object in QTP is identified primarily by its parent child relationship. So Browser column contains the name of Parent Browser in which object reside.

Object column contains child object in the Browser on which action needs to be performed. Object name is the name of the object or unique identity of the object E.g. for object like Web Element; it may not have its name as unique identity so html tag & class have been taken as unique properties to identify the object. The unique identity of object in object name column is required for creation of run time object from Application Map.

Action column contains action to be performed on the object. Here apart from generic action i.e. set or click, this column also has keyword as 'VerifyMinorError'. Based on this keyword, the functions to check if user has entered correct username /password will be called.

As name suggested Param column contains inputs for action to be performed.

Expected column contains expected value that needed to be compared with actual while checkpoint column contains the user defined messages to be displayed in pass/fail report.

How this will be processed further

The *Intermediate Tables* are handled by the *Sub Driver* which passes each *Step* to the *Core Engine* (consist of application map, component function, support function.vbs) for further processing.

About Core Engine

The Application Map:

The *Application Map* is one of the most critical items in this framework. Since the application GUI is not stable enough, so as suggested 'Application Map file' concept has been implemented.

Application Map is a .vbs file consisting subroutines that create runtime object by using Descriptive programming. With the use of application map file, creation of object repository step has been omitted.

Consider the below snippet of code for implementing a Application Map file concept

```
Start Page Test ApplicationMap.vbs
1: Option Explicit
2: Public WebBrowserDesc, WebPageDesc, WebEditDesc, WebButtonDesc, WebObjDesc
3: Dim arrObjProperty, arrObjValue, strObjProperty, strObjValue
4:
5: ' Subroutine which accepts title or name of the object & create runtime objects
6: Public Function fnApplicationMap(strBrowser, strObject, strObjName)
7:
8:     ' Creation of Browser Object
9:     Set WebBrowserDesc = Description.Create
10:    WebBrowserDesc("application version").value="internet explorer 6"
11:    If strBrowser <> "" Then
12:        WebBrowserDesc("name").value= strBrowser
13:    End If
14:    ' Creation of Browser Page Object
15:    Set WebPageDesc = Description.Create
16:    WebPageDesc ("title").value=strBrowser
17:
18:    ' Based on keyword in object column of Intermediate Tables , creation of runtime object
19:    Select Case strObject
20:        Case "WebEdit"
21:            Set WebEditDesc = Description.Create
22:            WebEditDesc("html tag").Value = "INPUT"
23:            WebEditDesc("name").Value = strObjName
24:        Case "WebButton"
25:            Set WebButtonDesc = Description.Create
26:            WebButtonDesc("html tag").Value = "INPUT"
27:            WebButtonDesc("name").Value = strObjName
28:        Case "WebElement"
29:            Set WebObjDesc = Description.Create
30:            ' splitting the webelement object name i.e html tag = TD, class = error-message by comma
31:            arrObjProperty = Split (strObjName, ",")
32:            For each strObjProperty in arrObjProperty
33:                ' For every combined property & value in object name , splitting again to separate property & value
34:                arrObjValue = Split (strObjProperty, "=")
35:                strObjValue = trim (arrObjValue(0))
36:                ' This is for creation of object description i.e. WebObjDesc("html tag").value = "TD"
37:                WebObjDesc(strObjValue).Value = trim (arrObjValue(1))
38:            Next
39:    End Select
40: End Function
```

Snippet 1

In above snippet 1 of code, Subroutine which accepts Object & Object name has been declared. Initially Parent Objects i.e. Browser & Browser Page are created in this subroutine. Afterward based on keyword in object column of Intermediate table, other child objects are created.

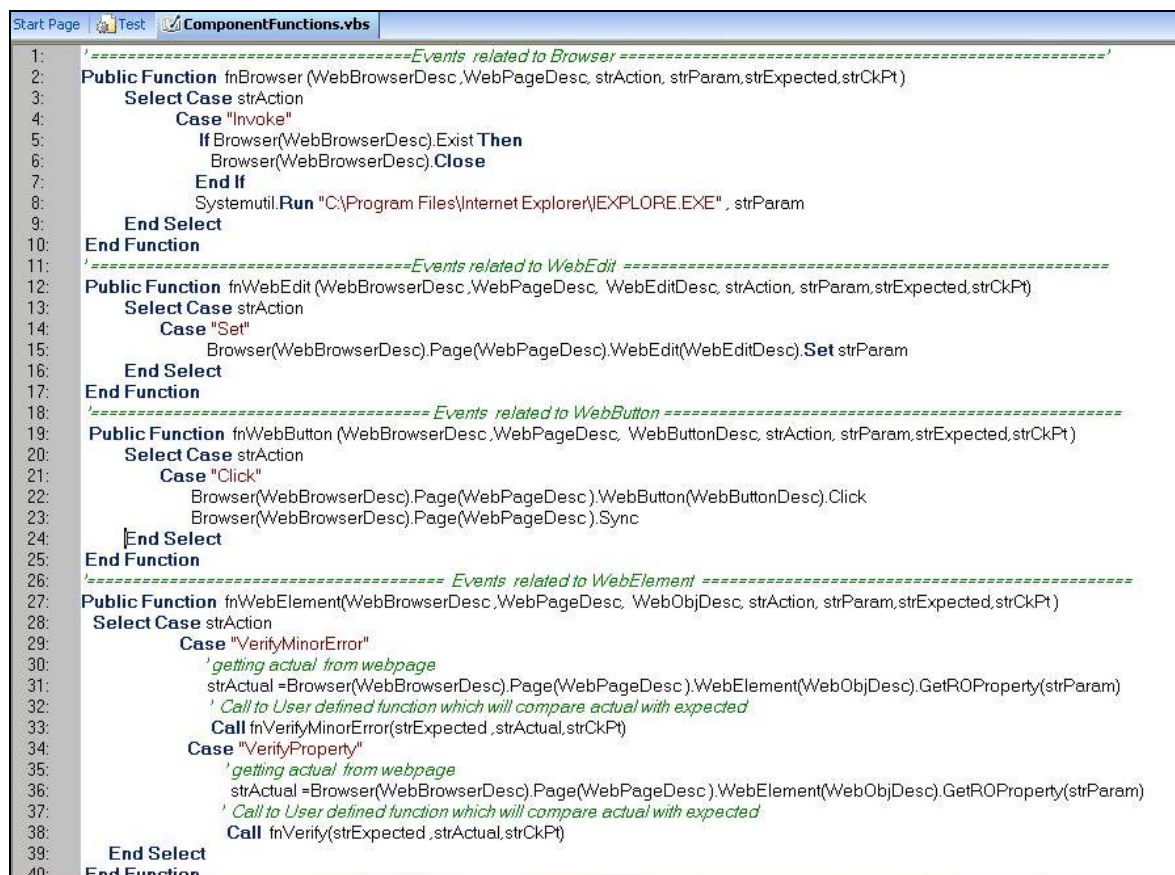
The Component Functions:

Once the objects has been created using Application Map, next step is to perform action on the object to drive the test script forward.

Component Functions are those functions that perform specific tasks (e.g. Invoke application action, press a key or button, etc), also call “User Defined Functions” if required.

In this automation framework various *Component Functions* for each type of object (e.g. WebEdit, WebButton, WebElement, WebLink, etc...) have been clubbed in a single .vbs file.

Consider the below snippet of code for Component Functions.



```
1: '-----Events related to Browser-----'
2: Public Function fnBrowser (WebBrowserDesc ,WebPageDesc, strAction, strParam,strExpected,strCkPt)
3:     Select Case strAction
4:         Case "Invoke"
5:             If Browser(WebBrowserDesc).Exist Then
6:                 Browser(WebBrowserDesc).Close
7:             End If
8:             Systemutil.Run "C:\Program Files\Internet Explorer\EXPLORE.EXE", strParam
9:         End Select
10:    End Function
11: '-----Events related to WebEdit -----'
12: Public Function fnWebEdit (WebBrowserDesc ,WebPageDesc, WebEditDesc, strAction, strParam,strExpected,strCkPt)
13:     Select Case strAction
14:         Case "Set"
15:             Browser(WebBrowserDesc).Page(WebPageDesc).WebEdit(WebEditDesc).Set strParam
16:         End Select
17:    End Function
18: '----- Events related to WebButton -----'
19: Public Function fnWebButton (WebBrowserDesc ,WebPageDesc, WebButtonDesc, strAction, strParam,strExpected,strCkPt)
20:     Select Case strAction
21:         Case "Click"
22:             Browser(WebBrowserDesc).Page(WebPageDesc).WebButton(WebButtonDesc).Click
23:             Browser(WebBrowserDesc).Page(WebPageDesc).Sync
24:         End Select
25:    End Function
26: '----- Events related to WebElement -----'
27: Public Function fnWebElement (WebBrowserDesc ,WebPageDesc, WebObjDesc, strAction, strParam,strExpected,strCkPt)
28:     Select Case strAction
29:         Case "VerifyMinorError"
30:             'getting actual from webpage
31:             strActual =Browser(WebBrowserDesc).Page(WebPageDesc).WebElement(WebObjDesc).GetROProperty(strParam)
32:             ' Call to User defined function which will compare actual with expected
33:             Call fnVerifyMinorError(strExpected ,strActual,strCkPt)
34:         Case "VerifyProperty"
35:             'getting actual from webpage
36:             strActual =Browser(WebBrowserDesc).Page(WebPageDesc).WebElement(WebObjDesc).GetROProperty(strParam)
37:             ' Call to User defined function which will compare actual with expected
38:             Call fnVerify(strExpected ,strActual,strCkPt)
39:         End Select
40:    End Function
```

Snippet 2

In above snippet 2 of code, the Browser, Page & Child Object descriptions (derived from Application Map) have been passed as parameter to the Component function. Apart from Object descriptions, the action which needs to be performed on these object along with parameter required to perform the action has also been passed.

Expected value & user defined checkpoint message passed to these functions are in turn passed to “User defined Function” for further comparison with actual value derived in component function. E.g. refer fnWebElement Function in above snippet 2 of code.

Here, we can also add extra code to help for initial conditions; synchronizations apart from those provided by the tool. E.g. refer fnBrowser Function in above snippet 2 of code. In this function before invoking the application, open browser (if any) has been closed.

The Support Functions:

The *Support Function* consists of generic functions with core logic. These functions are independent of Test Automation Framework & can be also useful outside the context of the framework. In this framework the Support Functions.vbs contains the general-purpose routines and utilities that let the overall automation framework do what it needs to do. They are the modules that provide things like

- File Handling
- String Handling
- Database Access
- Logging Utilities
- Error Handling Utilities

Consider the below snippet of code for Support Functions

```
Start Page Test SupportFunctions.vbs
1: 'Function Name : fnExecuteSql(strSchema, strPassword, strServer, strSQL)
2: 'Input Parameter : Schema name, Schema password, Oracle server name, sql to be executed
3: 'Return Parameter :
4: 'Description : This function connect to application database & execute SQL passed to the function as a string
5: Public Function fnExecuteSql(strSchema, strSchPass, strServer, strSQL)
6:
7:     Dim strConnection, objConnection, objRs
8:
9:     strConnection = "DRIVER={Microsoft ODBC for Oracle};UID=" & strSchema & ";PWD=" & strSchPass & ";SERVER=" & strServer & ";"
10:    'Creation of database object
11:    Set objConnection = CreateObject("ADODB.Connection")
12:    'open connection session
13:    objConnection.Open strConnection
14:    'error Handling
15:    If err.number <> 0 Then
16:        Call fnErrorHandling()
17:    End If
18:    'Execute SQL
19:    objConnection.Execute strSQL
20:    'error Handling
21:    If err.number <> 0 Then
22:        Call fnErrorHandling()
23:    End If
24:    'Calling function to close db session
25:    Call fnbdisconnect(objConnection)
26:
27: End Function
28:
29: '-----
30: 'Function Name : fnbdisconnect (byRef curSession)
31: 'Input Parameter : Sthe session name (string)
32: 'Return Parameter :
33: 'Description : The function disconnects from the database and deletes the session.
34: Function fnbdisconnect (byRef curSession)
35:
36:     curSession.close
37:     set curSession = Nothing
38: End Function
```

Snippet 3

```
Test SupportFunctions.vbs
Function Name : fnVerify(strExp,strAct,strCkPt)
Input Parameter : Expected vs Actual
Return Parameter :
Description : The function verify the functionality by comparing Expected vs Actual & log checkpoint messages in Main Driver tables accordingly.
Public Function fnVerify(strExp,strAct,strCkPt)

    Dim strSQL,strTime
    strTime = Now
    'Compare Expected with actual & log checkpoint message in Main Driver tables
    If strExp = strAct Then
        strSQL = "INSERT INTO TBL_TEST_AUDIT_TRAIL (execution_id,msg_type,audit_msg,audit_desc,CHECK_STATUS,audit_ts,TC_ID) values ("&intExp
        strRetSt = "Pass"
    Else
        strSQL = "INSERT INTO TBL_TEST_AUDIT_TRAIL (execution_id,msg_type,audit_msg,audit_desc,CHECK_STATUS,audit_ts,TC_ID) values ("&intExp
        strRetSt = "Fail"
    End If
    'Calling function to execute SQL
    Call fnExecuteSql(strSchema,strSchPass,strServer,strSQL)
End Function

-----
Function Name : fnVerifyMinorError(strExp,strAct,strCkPt)
Input Parameter : Expected vs Actual, Warning Message
Return Parameter :
Description : The function verify the functionality by comparing Expected vs Actual & log error correction messages in Main Driver tables accordingly
Public Function fnVerifyMinorError(strExp,strAct,strCkPt)

    Dim strSQL,strTime
    strTime = Now
    'Compare Expected with actual & log error correction message in Main Driver tables
    If strExp = strAct Then
        strSQL = "INSERT INTO TBL_TEST_AUDIT_TRAIL (execution_id,msg_type,audit_msg,audit_desc,CHECK_STATUS,audit_ts,TC_ID) values ("&intExp
        strRetSt = "MinErr"
    End If
    'Calling function to execute SQL
    Call fnExecuteSql(strSchema,strSchPass,strServer,strSQL)
    'Stop execution since the error occurred
    ExitAction(0)
End Function
```

Snippet 4

While in Snippet 3 generic database functions to Execute SQL & Disconnect database session have been provided, Snippet 4 provides the user defined functions which compares actual with expected & log pass/fail messages or error correction messages in Main Driver tables (For more details on Main Driver tables ref session Role of Main Driver in Test Automation Framework).

Role of Sub Driver in driving the Core Engine

After understanding the primary purpose of Core Engine (i.e. Application Map, Component Function, Support Function) let us now understand how the Drivers (which consist of Main Driver & Sub Driver) drives the core Engine. Let us first take look at the role of Sub driver in driving the Core Engine.

Sub Driver in this framework is the QTP Script. *Sub Driver* script processes intermediate-level tables listing *Steps* to execute. *Sub Driver* reads each record from the intermediate-level tables and passes each *Step* to *Core Engine* it finds during this process.

Consider the below snippet of code for Sub Driver

```

Start Page SubDriver
SubDriver
1: Option Explicit
2: On error resume next
3: 'Declaration of Variable
4: Public intExecID, strRetSt, strActual
5: Dim strComponent, strBrowser, strObject, strObjName, strAction, strParam, strExpected, strCkPt
6: Dim intRowCount, intIteration
7: Dim strLibParam, strDataFileParam, arrLibraryPath, strLibraryPath
8:
9: 'getting parameter value to this script
10: strDataFileParam = Parameter("strDataLoc")
11: strLibParam = Parameter("strLibrary")
12:
13: 'To include library files in the script
14: arrLibraryPath = split(strLibParam, ",")
15: For each strLibraryPath in arrLibraryPath
16:     If strLibraryPath <> "" Then
17:         ExecuteFile strLibraryPath
18:     End If
19: Next
20:
21: 'Import datafile in the script
22: DataTable.Import strDataFileParam
23:
24: 'Getting Row count & iteration till end of the file
25: intRowCount = DataTable.GetSheet(1).GetRowCount
26: For intIteration = 1 to intRowCount step 1
27:     'Initialising the variable from data from Intermediate datatable
28:     strComponent = DataTable.GetSheet("Global").GetParameter("Component").ValueByRow(intIteration)
29:     strBrowser = DataTable.GetSheet("Global").GetParameter("Browser").ValueByRow(intIteration)
30:     strObject = DataTable.GetSheet("Global").GetParameter("Object").ValueByRow(intIteration)
31:     strObjName = DataTable.GetSheet("Global").GetParameter("ObjectName").ValueByRow(intIteration)
32:     strAction = DataTable.GetSheet("Global").GetParameter("Action").ValueByRow(intIteration)
33:     strParam = DataTable.GetSheet("Global").GetParameter("Param").ValueByRow(intIteration)
34:     strExpected = DataTable.GetSheet("Global").GetParameter("Expected").ValueByRow(intIteration)
35:     strCkPt = DataTable.GetSheet("Global").GetParameter("CheckPoint").ValueByRow(intIteration)
36:

```

Snippet 5

```

SubDriver
37: 'Processing the individual steps
38: Select Case strComponent
39:
40:     'For Events related to Web
41:     Case "Web"
42:         'Creating Run time objects by calling subroutine
43:         Call fnApplicationMap(strBrowser, strObject, strObjName)
44:
45:         Select Case strObject
46:             Case "Browser"
47:                 'Call to Component functions related to Browser Object
48:                 Call fnBrowser(WebBrowserDesc, WebPageDesc, strAction, strParam, strExpected, strCkPt)
49:                 'error Handling
50:                 If err.number <> 0 Then
51:                     Call fnErrorHandling()
52:                 End If
53:             Case "WebEdit"
54:                 'Call to Component functions related to WebEdit Object
55:                 Call fnWebEdit(WebBrowserDesc, WebPageDesc, WebEditDesc, strAction, strParam, strExpected, strCkPt)
56:                 'error Handling
57:                 If err.number <> 0 Then
58:                     Call fnErrorHandling()
59:                 End If
60:             Case "WebElement"
61:                 'Call to Component functions related to WebElement Object
62:                 Call fnWebElement(WebBrowserDesc, WebPageDesc, WebObjDesc, strAction, strParam, strExpected, strCkPt)
63:                 'error Handling
64:                 If err.number <> 0 Then
65:                     Call fnErrorHandling()
66:                 End If
67:         End Select
68:
69:         'For Events related to database
70:         'Case "Database"
71:     End Select
72: Next

```

Snippet 6

In Snippet 5, Intermediate data tables & the Library Files are passed as the parameter to Sub Driver. After processing, the Library Files are included in the script, while intermediate table are read line by line until it reaches end of file. While reading every row of intermediate data table, Sub Driver stores value/Keyword of each column in the variable list.

In Snippet 6, When Sub Driver encounters the Component Keyword i.e. “Web” from intermediate data table; it starts further processing for web related events. Firstly it passes param list as the inputs to Application Map for creation of runtime objects. Afterward based on object keyword, it invokes the corresponding *Component Function* module to handle the task

Role of Main Driver in Test Automation Framework

Main Driver mainly deals with the entire set up required before running the individual test scripts The Driver Script consists of the following components, which are plug and play units.

Driver tables: - The Main Driver tables plays crucial role in this framework. Relational database concept has been implemented while designing main driver tables. Main Driver tables are ported in a separate schema created in Oracle Database.

Various tables are designed to hold data like the Core Engine File names that requires to be included in Sub Driver Script, the intermediate data table i.e. spreadsheet names, the sequence in which the intermediate data tables need to be sent to Sub Drivers (E.g. for batch processing, after login.xls the Functionality Test 1.xls, the Functionality Test 2.xls required to be sent in sequential manner to Sub Driver for further processing), etc

Apart from these tables, a special table named Audit Trail Table is designed to hold pass / fail results logged after comparing expected vs. actual while processing the test script. The idea being the development of the reporting tool that can represent data in Audit Trail Table (*For more details on this refer session Reporting Tool*).

Library Functions: - For reading data from Driver tables, Support Library Functions (that consist database handling functions) are included in the Main Driver script.

Initialization.vbs:- The initialization.vbs in this framework is used to declare and define the variables which are not going to changes frequently e.g. schema name/ password for the main driver tables, the sequence list for the intermediate tables, etc. The idea being at any point of time, the script executor should not require to interfere the Main driver code. Initialization.vbs is also included in the Main Driver Script

How Main Driver will work?

The main driver script connects to driver tables, reads data like Core Engine file/ Intermediates data file name, creates absolute path for these file names & sends to Sub

driver as parameter. After Sub Driver completes processing of first intermediate data table, the control returns back to main driver which is now ready to send the next intermediate data table for further execution.

Reporting Tool

Reporting Tool is the interesting plug & play feature for this Test Automation Framework. Here reporting tool are simple .jsp pages that represent data logged in driver table (viz: pass/fail results with user defined check point, start & end execution time of the script, etc) in html format. The advantage being the viewer can get online status of script execution, pass/ fail result on internet. Special feature like export to excel can be also added to this reporting tool.

In Nutshell

We have seen the features of implementation of Hybrid Test Automation framework using QTP; let us now recapture implementation of Automation Framework work flow from start.

The flow starts with the Main Test Driver Script. The Main Driver Script first loads the support functions & variables available in Initialisation.vbs file e.g. schema name/ password for the main driver tables, etc

Main Driver Tables are repository built in Oracle database that consists the information such as Core Engine File names, the intermediate data table i.e. spreadsheet names, the sequence for processing of the intermediate data tables, etc.

Main Driver Script connects to Driver tables with the help of support functions & variable list available from Initialisation.vbs, reads data from Driver tables & sends the same as parameter to Sub Driver Script.

The interaction logic with AUT has been depicted in Intermediate Data tables i.e. spreadsheet in the form of keyword. Sub Driver Script after including Core Engine files (consist of Application Map, Component Functions, Support Function), reads each record from intermediate data tables and store the same in variable list.

Application Map is a .vbs file consisting subroutines that create runtime object by using Descriptive programming. Sub Driver passes the variable list i.e. each record read from Intermediate data table to Application Map for creation of run time object. Based on keyword/ value available in variable list the Parent as well as child object required to execute the test steps are created.

Once the objects has been created using Application Map, next step is to perform action on the object to drive the test script forwards. Component functions consist of low level events (i.e. press button, set param in input boxes) to be performed on the object.

Sub Driver passes the object description (derived from Application Map) along with action to be performed & parameter required for performing action as inputs to Component Functions. Based on object name & action to be performed the corresponding component functions are invoked to handle the task.

The *Support Functions* are functions with core logic which are independent of any automation framework. These functions can be called from any module of framework if required. Apart from providing generic utilities (e.g. Database Access utilities, Error handling utilities, etc.), support functions also contain user defined functions viz sub routine for comparing actual vs. expected & logging pass and fail result in Main Driver Tables. The Pass/ Fail results logged in main driver tables are then represented in the Form of Reporting Tool.

References

Test Automation Frameworks by Carl J. Nagle

About the Author

Pallavi Patwa is a Senior Technical Consultant at TechMahindra Ltd. She is a CSTE from QAI and ISTQB Certified tester. She has about ten years of experience in the area of software validation and verification and test automation