Chapter No. 4
"Understanding and Creating Frameworks"
In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.4 "Understanding and Creating Frameworks"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Ashish Bhargava has over 10 years' experience in Information Technology. Working as a test lead, tester, and Java developer, he delivered many successful applications. He has experience in designing frameworks using various tools and languages. He has worked with HCL Technologies and Microsoft. He worked on QTP, Coded UI, and RFT, and has experience in VBScript, JavaScript, Java, and C#. He has experience in technically mentoring the junior team members to learn automation and test automation tools. He has experience in teaching automation and designing courses for test automation and manual testing. The author is the key contributor at http://testautomationlabs.com/.

Ashish has the following professional certifications:

- HP Certified QTP Professional
- ISTQB Certified
- Certified Software Test Manager (CSTM)

For More Information:

Designing and Implementing Test Automation Frameworks with QTP

HP QuickTest Professional is a test automation tool for functional and regression testing for software applications and environments. HP QuickTest Professional supports keyword and scripting interfaces and features a graphical user interface. It uses the Visual Basic Scripting Edition (VBScript) to specify a test procedure and to manipulate the objects and controls of the application under testing.

This book is a great platform for sharing my knowledge of designing and implementing the test automation framework that I have acquired over the years, while working in Information Technology and teaching QTP to professionals. If any professional or student finds it difficult to implement frameworks, this is the right book—one which teaches approaches and concepts with simple examples. This book covers the managerial, technical, and design concepts, and is uniquely designed to deliver the knowledge needed to create an effective framework. It demonstrates easy ways to implement and learn concepts, along with code and suggestions for creating a portable framework across various versions of QTP.

This book also covers the new features for creating test automation using XPath in QTP and JavaScript for automated web applications.

What This Book Covers

Chapter 1, Automation Life Cycle and Automation Goals, explains the automation life cycle and tool selection which helps to implement better processes for test automation and tool selection. This chapter also explains the overall automation and design goals that help develop a better test automation framework and guides you towards implementing better practices to achieve maximum output.

Chapter 2, Essentials of Automation, explains the key elements for designing test automation using QTP along with examples on how to use each of them. These elements serve as the base to create a complete test automation script.

Chapter 3, Basic Building Blocks for Creating Frameworks, explains the basic code-building blocks (with examples) for designing the implementation of test automation frameworks, all of which are used in implementing the framework(s).

For More Information:
Chapter 4, *Understanding and Creating Frameworks*, explains what we mean by test automation frameworks and what types of frameworks exist. It guides you towards understanding these frameworks before we design and implement them. The chapter will also highlight the differences and commonalities among them. It explains the basics of test automation frameworks and builds the user's maturity in implementing test automation framework(s) end-to-end.

Chapter 5, *Deploying and Maintaining Frameworks*, explains, in the beginning of the design phase, how and what needs to be done to maintain and enhance these frameworks with ease.

Chapter 6, *DOM- and XPath-based Framework in Web Applications*, explains the various web-based technologies and builds the maturity to implement these technologies for automated web-based applications. It explains HTML, XPath, DOM, JavaScript, and how to use them effectively for creating test automation scripts.

Chapter 7, *Capturing the Lessons Learned*, explains what should be captured for lessons to be learned and how to make sure that these lessons help in future projects or serve as input in other projects.

For More Information:
This chapter provides a detailed approach and guidance for creating a framework for test automation.

Creating frameworks in QTP requires knowledge of the following:

- VBScript basics, control structures, loops, and built-in functions
- Basic concepts and approaches for designing various frameworks
- Programming constructs that help to create the utilities for the framework

**Programming with VBScript**

Knowledge of Visual Basic script is essential to create frameworks. The knowledge of creating simple statements, controls structure, loops, and built-in functions in VBScript allows us to enhance the test scripts and make them more robust; it also allows us to interact with external resources. This section describes the VBScript features that are useful throughout the designing of the frameworks.

**Using VBScript**

QTP uses VBScript as a scripting language. VBScript allows scripts to deal with resources that are not a part of the QTP itself, for example, filesystem and external data sources (Excel, databases, and so on). VBScript allows us to define the function procedures; it provides a lot of built-in functions that help to create the reusable functions. VBScript supports the regular expression that assists in verification and validation. Since it's a lightweight language, the framework designer is benefited.

For More Information:
VBScript datatypes

Variant is the only datatype in VBScript. A variant can contain different kinds of information depending on its declaration and use. It can contain numeric, constant, date, string, and Boolean values.

The following is a list of the subtypes of a variant:

<table>
<thead>
<tr>
<th>Subtype</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>Variant is uninitialized</td>
<td>0 for the numeric variables or Zero-length for the string variables</td>
</tr>
<tr>
<td>Null</td>
<td>Variant initially contains no valid data</td>
<td>Does not contain data</td>
</tr>
<tr>
<td>Boolean</td>
<td>True or False</td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>Small integer</td>
<td>0-255</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>From -32,768 to 32,767</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>From -922,337,203,685,477.5808 to 922,337,203,685,477.5807</td>
</tr>
<tr>
<td>Long</td>
<td>Long integer</td>
<td>From -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>Single</td>
<td>Single-precision floating-point number</td>
<td>From -3.402823E38 to -1.401298E-45 for negative values, and from 1.401298E-45 to 3.402823E38 for positive values</td>
</tr>
<tr>
<td>Double</td>
<td>Double-precision floating-point number</td>
<td>1.79769313486232E308 to 4.94065645841247E-324 for negative values, and 4.94065645841247E-324 to 1.79769313486232E308 for positive values</td>
</tr>
<tr>
<td>Date (Time)</td>
<td>A number that represents a date</td>
<td>Dates from January 1, 100 to December 31, 9999</td>
</tr>
<tr>
<td>String</td>
<td>Represents the characters</td>
<td>Contains characters of variable length string, approximately 2 billion in length</td>
</tr>
<tr>
<td>Object</td>
<td>Contains an object</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Contains an Error object</td>
<td></td>
</tr>
</tbody>
</table>

For More Information:
Operators in VBScript

Operators are useful in creating the expressions; these expressions are created by arithmetic, comparison, and logical operators. In VBScript, we can divide operators into four major categories:

- Mathematical
- Comparison
- Concatenation
- Logical

<table>
<thead>
<tr>
<th>Mathematical</th>
<th>Symbol</th>
<th>Comparison</th>
<th>Symbol</th>
<th>Logical</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>^</td>
<td>Equality</td>
<td>=</td>
<td>Logical negation</td>
<td>Not</td>
</tr>
<tr>
<td>Urinary</td>
<td>-</td>
<td>Inequality</td>
<td>&lt;&gt;</td>
<td>Logical conjunction</td>
<td>And</td>
</tr>
<tr>
<td>negation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>Less than</td>
<td>&lt;</td>
<td>Logical disjunction</td>
<td>OR</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>Greater than</td>
<td>&gt;</td>
<td>Logical exclusion</td>
<td>XOR</td>
</tr>
<tr>
<td>Integer</td>
<td>\</td>
<td>Less than</td>
<td>&lt;=</td>
<td>Logical equivalence</td>
<td>EQV</td>
</tr>
<tr>
<td>division</td>
<td></td>
<td>equal to</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modulus</td>
<td>MOD</td>
<td>Greater than</td>
<td>&gt;=</td>
<td>Logical implication</td>
<td>IMP</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td>Object</td>
<td>Is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>equivalence**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Integer division divides two numbers and returns an integer result, for example, 5/2 will result in 2 not 2.5.

The modulus or remainder operator divides operand 1 by operand 2 (rounding the floating-point number to an integer) and returns a remainder, for example, 5 MOD 2 will result in 1.

Object equivalence compares two object reference variables. If object1 and object2 both refer to the same object, the result is True; if they don't, the result is False.
Adding examples of expressions
An expression is a combination of explicit values, constants, variables, operators, and functions that are interpreted according to particular rules of precedence and association for a particular programming/scripting language.

Control structures and loops
A control structure, conditionally, executes a group of statements, depending on the value of an expression. There are four constructs that control the flow of the execution, which are shown in the following table:

| If (condition) statement1 | If condition Then statements [Else else statements ] End If | If condition Then [statements] [ElseIf condition Then elseif statements] [Else else statements] End If | Select Case expression [Case expression list [statements]] [Case Else else statements]] End Select |

An example of the If and else control statement is given as follows:

```
If True Then
Else
    Reporter.ReportEvent micFail, "Name of the step", "Details..."
End If
If elseif ... and end if
    color ="b"

    If color ="r" Then
        messagebox "red"
    elseif color ="g" then
        messagebox "green"
    elseif color="b" then
        messagebox "blue"
    else
        messagebox "invalid color..."
End If
```

For More Information:
The following is an example of the Select case:

```
Dim Color, bgcolor
Select Case Color
    Case "red"     bgColor = "red"
    Case "green"   bgColor = "green"
    Case "blue"    bgColor = "blue"
    Case Else      MsgBox "pick another color"
End select
Msgbox "You have Selected the " & bgcolor
```

Running the preceding lines of code executes a series of statements as long as the given condition is True.

### Using loops

A loop executes a sequence of statements that are specified once but may be carried out several times in succession. VBScript allows four looping constructs as shown in the following table:

<table>
<thead>
<tr>
<th>While ... Wend</th>
<th>Do... Loop</th>
<th>For... Next</th>
<th>For each... Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>While condition [statements]</td>
<td>Do [[While</td>
<td>Until] condition] [statements] [Exit Do] [statements] Loop</td>
<td>For counter = start To end [Step step] [statements] [Exit For] [statements] Next</td>
</tr>
<tr>
<td>Wend</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following is an example of the usage of While ... Wend:

```
'Print 10 random numbers
counter = 0 'Initialize the counter
While counter < 10
    print Cint( rnd() * 100 )
    counter = counter + 1 'Increment the counter
Wend
```

For More Information:

One of the drawbacks of the While...Wend loop is that it does not have the exit statement to terminate from the loop but other looping constructs allow us to exit from the loop.

The Do Loop

The Do...Loop runs statements for an indefinite number of times. The statements are repeated either while a condition is true or until it becomes true.

The following is an example of the usage of Do Loops:

'Execute the loop until the response until CANCEL button is clicked
Usages 1
Do Until DefResp = vbNo
    MyNum = Int (6 * Rnd + 1)
    'Generate a random number between 1 and 6.
    DefResp = MsgBox (MyNum & "Do you want another number?", vbYesNo)
    'The msgbox displays generated random number and OK and Cancel button
Loop
Usages 2
Do while not DefResp = vbNo
    MyNum = Int (6 * Rnd + 1)
    DefResp = MsgBox (MyNum & "Do you want another number?", vbYesNo)
Loop
Usages 3
Do
    MyNum = Int (6 * Rnd + 1)
    DefResp = MsgBox (MyNum & "Do you want another number?", vbYesNo)
Loop until DefResp = vbNo

There are two variants of Do Loop; we can use the while or until conditions with Do or Loop. When we use the while or until condition with Do, the execution starts with validating the condition first and then enters into the loop; refer to example 1. In the latter case, example 3, the first loop is executed once and the condition is checked later.

The For Next loop iterates statements over a predefined number of times. Refer to the following code.

The ToProperties method returns a collection of the properties of the test object.

    set Pcoll =
    Dialog("DialogLogin").WinEdit("EditAgentName:").GetTOProperties()
For i = 0 to coll.count() – 1  
'Count method return the numbers of properties  
    -print "Item is " & Pcoll.item(i)  
Next

The result of the preceding set of code is as follows:

Item is Edit
Item is Agent Name:
Refer the above results and compare the properties of the object
are same in the following image.

Refer to the following screenshot:

The For each ... Next loop iterates statements for each element in an array
or collection.

Dim dicObj creates a dictionary variable, which is shown as follows:

Set city = CreateObject("Scripting.Dictionary") 'Creates
  Dictionary Object
'Dictionary stores data key, item (key - value) pairs.
'Add keys and items.
dicObj.Add "a", "Atlanta"
dicObj.Add "b", "Paris"
dicObj.Add "c", "New Delhi"
'Keys is a method that returns an array or collection containing
    all existing keys in a Dictionary object.
For each keyitem in d.Keys
    print keyitem
Next

For More Information:
Introducing frameworks

A framework is defined as a broad overview, guideline, or skeleton of the interlinked components, which supports a particular approach to a specific objective, and serves as a guide that can be enhanced as required by adding or deleting components. A framework is a working or conceptual model that supports or provides guidelines for creating or expanding the test scripts to achieve the test automation, ensuring lower maintenance and easy expandability.

A test automation framework is a layered structure and provides the mechanism to interrelate and interact with each other to achieve the common goals. Frameworks also include actual programs and interfaces or offer utility tools for using the frameworks. A framework facilitates a standard way for modifying, adding, and deleting the scripts and functions. It's a comprehensive structure that provides scalability and reliability with less efforts.

Automation goals can be achieved by selecting the right framework that is suitable for test automation. The cost of test automation includes both development and maintenance efforts. Selecting the suitable framework and techniques helps in maintaining the lower cost and high impact solution.

The automation framework and types of framework

In general, various structures and techniques are used to design the framework. Broadly based on these techniques and structures, we can classify the framework as follows:

- **Linear**: In this, the script is created in a linear fashion, usually generated by recording and replaying without or with only slight modification.
- **Data-driven**: Parameterizes the test and fetches data from a persistent data source. The data source could be internal or external to the test.
- **Modular**: A modular framework is designed to achieve modularity at test as well as script levels. The modular framework can be of the following combinations:
  - Test script modular framework
  - Test library modular framework
- **Keyword-driven**: Keyword-driven framework is designed for reducing the maintenance cost by separating test cases from their execution.
- **Hybrid**: In this, two or more of the previous patterns are used.

For More Information:
Record and replay

Recording is the process of capturing an object and its properties, creating the test objects, and storing them in the Object Repository with hierarchy. Assign a logical name to the test object and create the scripts by capturing the operation performed on the GUI.

![Screenhots showing AUT, recorded script, and OR. The AUT contains the test objects. OR stores the test objects in a hierarchy and with their identification properties.]

The preceding screenshot shows the AUT, recorded script, and OR. The AUT contains the test objects. OR stores the test objects in an hierarchy and with their identification properties:

The recorded scripts take us through the following three test steps:

1. Click on the Login dialog.
2. Enter ashish in the Agent Name textbox.
3. Enter the password.

A test step contains test objects with hierarchy, operation, and data value (if applicable) for the test object. While recording the user action on AUT, OR creation and scripts generation almost happen simultaneously. This approach is also called the QTP linear framework approach. In the linear framework approach, scripts are recorded in a step-by-step fashion without focusing on reusability. Consider an example where you have a test case to log in to an application, search for some data and then log out. In the linear framework, the code would look like something shown in the following example:

Steps for creating the QTP linear framework:

1. Enter username.
2. Enter password.

For More Information:

3. Click on the **OK** button.
4. Enter the **Flight** date.
5. Select **Fly From** (source).
6. Select **Fly To** (destination).
7. Click on the **FLIGHT** button.
8. Click on the **OK** button.
9. Enter the name.
10. Click on the **Insert Order** button.
11. Close the **Flight Reservation** window.

```
'Login
Dialog("DialogLogin").Activate
Dialog("DialogLogin").WinEdit("EditAgentName:").Set "ashish"   'Enter the agent name
Dialog("DialogLogin").WinEdit("EditPassword:").SetSecure
"S1e691da82a99155f73b209eeaeac51a66ef0883"   ' Enter password
'Create Order
Window("WinFlight").Activate
Window("WinFlight").ActiveX("MaskEdBox").Type "111114"   'Enter Flight Date
Window("WinFlight").WinComboBox("FlyFrom").Select "London"   'Select Source
Window("WinFlight").WinComboBox("Fly To:").Select "Frankfurt"   'Select destination
Window("WinFlight").WinButton("btnFLIGHT").Click   'Click on flight button
Window("WinFlight").Dialog("FlightTable").WinButton("btnOK").Click   'Click on the OK button
Window("WinFlight").WinEdit("EditName:").Set "Mr. James Anderson"   'Enter the passenger name
Window("WinFlight").WinButton("btnInsertOrder").Click   'Click on the Insert Order button
Window("WinFlight").Activate
Window("WinFlight").Close   'Click on the close button
```

This is the simplest way to create scripts and use and implement them if the AUT is small, but do not expect frequent changes in AUT. The normal record and playback falls under this framework.

For More Information:

Advantages and disadvantages of linear a framework

The advantages of linear framework are stated as follows:

- No expertise or experience of programming or designing is required. Basic knowledge of QTP is required for creating scripts.
- Automatic test object creation and storage in the Object Repository.
- This is the fastest way to create test automation scripts.
- This is the simplest framework, and it is easy to understand.
- Helps to learn the objects and create the scripts manually.
- Can add checkpoints easily.

The disadvantages of linear framework are stated as follows:

- Does not allow reusability since scripts are created in a linear fashion and do not have functions.
- Data is bound with the scripts; hence, the test is noniterative, and this approach is inefficient for executing the test for multiple sets of data. Change the data manually for each run or create multiple copies of scripts; unfortunately, both of these techniques are inefficient.
- The maintenance cost is very high, and it is tedious, and error-prone since it is required to make changes in all the affected areas.
- Need to add comments on our own.

Introduction to the data-driven framework

A data-driven framework is the collection of test scripts that run with related multiple datasets. This framework provides reusable scripts for different sets of data and it improves the test coverage. Input and result (test criteria) data values can be stored in one or more central data sources or databases; the actual format and organization can be implementation specific.

To understand the data-driven framework implementation we should know three terms:

- Variable
- Parameter
- Iteration

For More Information:

A variable is a storage location and an associated with an identifier, which contains some known or unknown quantity or information, a value.

A parameter is a variable that is provided as an input to the scripts. Parameterization in QTP can be achieved by the *datatable* object. The *datatable* object represents the QuickTest design-time data table and its associated sheets and parameters. The *datatable* object has various methods and properties to access data from the runtime data table object.

Iteration is a process where a parameterized script executes the test for a predefined number of times from a data source.

There are four major steps in creating a data-driven framework:

1. Create a script.
2. Define the parameters that hold data.
3. Add code to get data from the data source and assign a value.
4. Modify the settings or add code to execute the test for all the rows or subset of rows.

There are two major ways to access data in QTP:

- Using *datatable* object
- Using external data source

Creating data-driven scripts using the *datatable* object:

1. Create the script by recording or manually.
2. Define parameters in an external Excel sheet as shown in the following screenshot. Make sure that the first row contains the parameter name. Enter data in the subsequent rows as shown in the following screenshot:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User:Name</td>
<td>Password</td>
<td>Flight</td>
<td>FlyFrom</td>
<td>FlyTo</td>
<td>Agent</td>
</tr>
<tr>
<td>2</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
<td>harsha</td>
</tr>
<tr>
<td>3</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
<td>ashish</td>
</tr>
<tr>
<td>4</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
<td>dinesh</td>
</tr>
</tbody>
</table>

For More Information:
To fetch the data from the external excel sheet, use the `import` method of `datatable`:

1. Define parameters using the `datatable` object
2. To add the data, rename the column name by double-clicking on the column name. Rename the column and add data to the subsequent rows as shown in the following screenshot:

<table>
<thead>
<tr>
<th>UserName</th>
<th>Password</th>
<th>Flight</th>
<th>FlyFrom</th>
<th>FlyTo</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
</tr>
<tr>
<td>3</td>
<td>ashish</td>
<td>mercury</td>
<td>111114</td>
<td>London</td>
<td>Paris</td>
</tr>
</tbody>
</table>

3. Use the data-driver tool or `datatable` object parameterization. Add the test data to `datatable`, add the column name to the header, and double-click on `Add` or change the column name to parameter in the data table.

4. Make the test scripts iterative.

There are two ways to make scripts iterative. First, under **Test Settings** select **Run one iteration only**, **Run on all rows**, and then **Run from row n to row m**:

Second, you can also programmatically create the scripts for running the test script:

```python
' Set the current row to retrieve the value from datatable
datatable.SetCurrentRow startnum
For i = startnum to endnum 'run the script from row (n) to row (m)
  .
  .
  .
' Move to next row of datatable
datatable.SetNextRow
Next
```

*For More Information:*

To run for one iteration, startnum and endnum should be 1, startnum=endnum=1.

To run test for all rows, startnum = 1 and endnum = datatable.datatable.GetRowCount.

Let’s club all the concepts together to parameterize the test. Creating the test data in datatable.datatable takes two arguments: parameter name and sheet type. datatable has two types of sheets: local sheet (local to action) and global sheet. To access the data from the global sheet, use dtGlobalSheet, and to access the data from the local sheet, use dtLocalSheet. Refer to the following example:

SystemUtil.Run PathToFlightApp
Dialog("DialogLogin").Activate
Dialog("DialogLogin").WinEdit("EditAgentName:").Set DataTable("Agent", dtGlobalSheet)
Dialog("DialogLogin").WinEdit("EditPassword:").SetSecureDataTable("Password", dtGlobalSheet)
Dialog("DialogLogin").WinButton("btnOK").Click
Window("WinFlight").ActiveX("MaskEdBox").Type DataTable("FlightDate", dtGlobalSheet)
Window("WinFlight").WinComboBox("FlyFrom").Select DataTable("FlyFrom", dtGlobalSheet)
Window("WinFlight").WinComboBox("Fly To:").Select DataTable("Flyto", dtGlobalSheet)
Window("WinFlight").WinButton("btnFLIGHT").Click
Window("WinFlight").Dialog("FlightTable").WinButton("btnOK").Click
Window("WinFlight").WinEdit("EditName:").Set DataTable("Name", dtGlobalSheet)
Window("WinFlight").WinButton("btnInsertOrder").Click
Window("WinFlight").WinButton("btnButton").Click
Window("WinFlight").Close

The following code shows how to import and iterate the test using datatable:

datatable.Import "c:\FlightData.xls"
' Import the excel to the datatable.
rc = datatable.GetRowCount
'get the row count
For i = 1 to rc
    systemutil.Run PathToFlightApp
    Dialog("DialogLogin").Activate
    Dialog("DialogLogin").WinEdit("EditAgentName:").Set DataTable("Agent").Value("Agent")
'Value is DataTable default property. Retrieves or sets the value of the cell in the specified parameter and the current row of the run-time Data Table.
'To find the value use DataTable.Value(ParameterID [, SheetID])
.SheetID Optional. Identifies the sheet to be returned. The SheetID can be the sheet name, index or dtLocalSheet, or dtGlobalSheet.
Dialog("DialogLogin").WinEdit("EditPassword: ").SetSecure datatable.
Value("Password")
Dialog("DialogLogin").WinButton("btnOK").Click
Window("WinFlight").ActiveX("MaskEdBox").Type datatable.
Value("FlightDate")
Window("WinFlight").WinComboBox("FlyFrom").Select datatable.
Value("FlyFrom")
Window("WinFlight").WinComboBox("Fly To:").Select datatable.
Value("Flyto")
Window("WinFlight").WinButton("btnFLIGHT").Click
Window("WinFlight").Dialog("FlightTable").WinButton("btnOK").Click
Window("WinFlight").WinEdit("EditName: ").Set datatable.Value("Name")
Window("WinFlight").WinButton("btnInsertOrder").Click
Window("WinFlight").WinButton("btnButton").Click
Window("WinFlight").Close
datatable.SetNextRow
Next

Using the Excel application with VBScript:

Set excelfile = createobject("excel.application")

'Create the excel first before executing script.
'Ensure that excel file is in Closed state.
excelfile.Workbooks.Open "D:\parameter.xls"
set sheet = excelfile.ActiveWorkbook.Worksheets("Sheet1")
'Get the max row occupied in the excel file
Row = sheet.UsedRange.Rows.Count

'Read the data from the excel file
For i = 2 to Row
Username = sheet.cells(i,1).value
Password = sheet.cells(i,2).value
wait 1
Next
'Close the Workbook
excelfile.ActiveWorkbook.Close

'Close Excel
excelfile.Application.Quit

'Release the objects
Set sheet = nothing
Set excelfile = nothing

The FileSystemObject object model

VBScript allows you to process drives, folders, and files using the FileSystemObject (FSO) object model, which is explained in the following section that describes how you can use FileSystemObject to manipulate files.

There are two ways for file manipulation:

- Creating and appending files, removing data from the files, and reading from the files
- Copying, moving, and deleting files

The following are the steps for reading and writing data from text files:

1. Create a text file.
2. Write data to it.
3. Close it.
4. Open the text file again.
5. Read the file.
6. Close it.

The following is an example of reading and writing to the text file:

    Dim filefso, file1, readfile, s
    Const ForReading = 1
    'create the object of a File system Object
    Set filefso = CreateObject("Scripting.FileSystemObject")
    Set file1 = filefso.CreateTextFile("c:\logfile.txt", True)
    'Write a line.
    file1.WriteLine "Testing FSO"
    file1.WriteBlankLines(2)
    file1.Close

For More Information:
'Read contents of the text file.
'Reading file
Set readFile = filefso.OpenTextFile("c:\logfile.txt", ForReading)
's = readFile.ReadLine
'iterate the file until end of the file
Do While readFile.AtEndOfStream <> True
  retstring = readFile.ReadLine
Loop
ts.Close

An example of using a text file for data-driven testing is given as follows:

Dim fso, f1, textfile, s
Const ForReading = 1
Set textfile = fso.OpenTextFile("c:\logfile.txt", ForReading)
's = textfile.ReadLine
'iterate the file until end of the file
Do While textfile.AtEndOfStream <> True
  retstring = textfile.ReadLine
Dialog("DialogLogin").Activate
Dialog("DialogLogin").WinEdit("EditAgentName:").Set datatable.Value("Agent")
'Value is DataTable default property. Retrieves or sets the value of the cell in the specified parameter and the current row of the runtime Data Table.
'To find the value use DataTable.Value(ParameterID [, SheetID])
  .SheetID Optional. Identifies the sheet to be returned. The SheetID can be the sheet name, index or dtLocalSheet, or dtGlobalSheet.
Dialog("DialogLogin").WinEdit("EditPassword:").SetSecure datatable.Value("Password")
Dialog("DialogLogin").WinButton("btnOK").Click
Window("WinFlight").ActiveX("MaskEdBox").Type datatable.Value("FlightDate")
Window("WinFlight").WinComboBox("FlyFrom").Select datatable.Value("FlyFrom")
Window("WinFlight").WinComboBox("FlyTo:").Select datatable.Value("Flyto")
Window("WinFlight").WinButton("btnFLIGHT").Click
Window("WinFlight").Dialog("FlightTable").WinButton("btnOK").Click
Window("WinFlight").WinEdit("EditName:").Set datatable.Value("Name")
Window("WinFlight").WinButton("btnInsertOrder").Click
Window("WinFlight").WinButton("btnButton").Click
Window("WinFlight").Close
Loop
textfile.Close

For More Information:
Understanding and Creating Frameworks

The methods used for performing the read and write operation on the test files are listed as follows:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Write to an opened text file</td>
</tr>
<tr>
<td>WriteLine</td>
<td>Write to an open text file and add a newline character to it</td>
</tr>
<tr>
<td>WriteBlankLines</td>
<td>Write blank line(s) to an opened text file</td>
</tr>
<tr>
<td>Read</td>
<td>Read characters from the text file, which are specified as an argument</td>
</tr>
<tr>
<td>ReadLine</td>
<td>Read the entire line, excluding the newline character</td>
</tr>
<tr>
<td>ReadAll</td>
<td>Read the entire data from a text file</td>
</tr>
</tbody>
</table>

The following example demonstrates the file manipulation operations:

```vbscript
Dim filefso, txtfile1, txtfile2, txtfile3
Set filefso = CreateObject("Scripting.FileSystemObject")
Set txtfile1 = filefso.CreateTextFile("d:\tmp\testfile.txt", True)
'Write a line.
txtfile1.Write ("Writing a text")
'Close the file to writing.
txtfile1.Close
'Moving the file to d:\tmp
'Get a handle of the file in root of d:\.
Set txtfile2 = filefso.GetFile("d:\tmp\testfile.txt")
'Moving the file to tmp directory.
txtfile2.Move ("d:\tmp\testfile.txt")

'Copying the file to temp.
txtfile2.Copy ("d:\temp\testfile.txt")

Set txtfile2 = filefso.GetFile("d:\tmp\testfile.txt")
Set txtfile3 = filefso.GetFile("d:\temp\testfile.txt")
'Deleting the files
txtfile2.Delete
txtfile3.Delete
```

For More Information:
The following is an example of using the ADODB to get data from the database:

```vbscript
Set Conn = CreateObject("ADODB.Connection")
' Set the Connection String.
Conn.ConnectionString = "DSN=QT_Flight32;DBQ=C:\Program Files\Mercury Interactive\QuickTest Professional\samples\flight\app\flight32.mdb;Driver=C:\WINDOWS\system32\odbcjt32.dll;DriverId=281;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;"
' ADODB.Connection
' RecordSet
Conn.Open("DSN=QT_Flight32")
Set rcRecordSet= Conn.Execute("SELECT order_number from Orders order by order_number desc")
rcRecordSet.MoveFirst
var_order_num = rcRecordSet.fields("Order_Number")
rcRecordSet.close
Conn.close
set rcRecordSet = nothing
set Conn = nothing
```

**Introduction to the modular framework**

Modularity allows decomposing the components and/or functionality and recombines them. This approach is a design technique that emphasizes on separating the functionality of an AUT into independent, interconnected modules such that each module contains everything necessary to execute only one aspect of the desired functionality.

Achieving the modularity requires modularity at two different layers; one layer is test and the other layer is script. To create a modular framework, we need to decompose the test layer into manageable pieces based on their objectives. For example, common test libraries are separate from function libraries. At test level, we decompose the key libraries and resources into a structure to achieve the automation goals using appropriate design.
There are four distinct parts of the modular framework:

- Script-level modularity
- Test-level modularity
- Resource structure
- Framework design

Let's take the example of the Flight application; the entire functionality can be divided into small independent functions as shown in the following figure. This requires to achieve script-level modularity as shown in the following figure:

![Script level modularity diagram](Diagram)

We can omit a few functions as part of the automation that are least used and do not contribute towards ROI, such as graph, edit, and help in the Flight application.

Apart from decomposing the entire functionality into small, independent functions, we need to ensure that other components should also be decomposed. After decomposing, the entire test component look likes the following figure:

![Test Level Modularity diagram](Diagram)

The preceding list contains the most frequently used components, but implementation of the framework may require having a few more or less components based on the automation goals and requirements. Following is the description of these components:

- **Function library**: This is a collection of scripts that perform a particular task. Usually one task that allows to perform on the set of statements on AUT.

- **Test data**: This is stored in datatable, or an external data source provides the input for the tests.

- **Log files**: They capture the log message that is used to see the outcome of the test scripts.

For More Information:

- **Master scripts**: These drive the flow of the test and allow coordination with the other components to ensure that the test runs successfully.

- **Error handlers**: These allow to exit the test gracefully when an error occurs, and reveal information about the error.

- **OR**: This is a part of the test that contains the test object; usually the OR is a shared OR.

- **Environment**: This is a component used to provide test-level environment variables or values that allow configuration of the tests that have to be run on various environments.

**Structure**

After the test is decomposed, resources are required to be arranged in a structured way, which means there is a centralized repository for these resources. Managing the test requires the folder structure or test management tool (for example, Quality Center) to store the test resources. We need to ensure that the test component stays as defined by the guidelines and structure. Structure allows organizing the resources for achieving portability and consistency. The framework's folder structure is as shown in the following screenshot. The structure may vary from project to project but it's important for achieving consistency.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>File folder</td>
<td>Environment</td>
</tr>
<tr>
<td>File folder</td>
<td>FunctionLibrary</td>
</tr>
<tr>
<td>File folder</td>
<td>Log</td>
</tr>
<tr>
<td>File folder</td>
<td>MasterScript</td>
</tr>
<tr>
<td>File folder</td>
<td>OR</td>
</tr>
<tr>
<td>File folder</td>
<td>Results</td>
</tr>
<tr>
<td>File folder</td>
<td>TestData</td>
</tr>
<tr>
<td>File folder</td>
<td>UtilityFunction</td>
</tr>
</tbody>
</table>

For More Information:

Advantages
Structure has the following advantages:

- **Facilitates specialization**: Structure designs the components to perform the specific tasks and provide abstraction from complexity.
- **Control over resources**: Resource structure simplifies control over resources because they are at centralized locations and governed by policies for accessing, creating, and updating the resources.
- **Easier communication**: Resource structures clearly state the flow of information in a controlled way among team members. The modifications are done at a centralized location and utilized by all the other team members.
- **Better performance**: Each component is specialized to perform its task and is tested thoroughly to improve the performance and reliability of the test.

Always use the relative path; do not hardcode the resource location in scripts.

Design
Design is the most important aspect of frameworks. The design of a framework is driven by its key automation goals. The following is a comprehensive but not complete list of goals, where a few more goals can be added if required:

- **Maintainability**: It allows decomposing of resources and scripts; this make it easier to maintain and debug, and to fix the issues faster and reliably.
- **Readability**: It is a well-designed and smaller function that improves the readability of the script and makes it easy to understand as well as enhances it.

Comments and script/function headers are important to improve the readability and understandability.

- **Extensibility**: When small functions are created and resources are structured, extending the existing functionality becomes easy. This allows us to add and enhance functionality, debug, and test with ease faster.

For More Information:
• **Reusability**: Functions that allow reusability to function make the framework robust and reliable.

Before we start designing the framework, we should think of functions that can be used across scripts in the test to increase the reusability. Test these common functions thoroughly, and make sure that they have the necessary error handling capabilities.

• **Concise scripts**: Make sure that scripts are short and do a single task at a time; these scripts should be a part of the appropriate libraries or modules. Ensure that the scripts are tightly cohesive and loosely coupled.

**High cohesion and coupling**

Coupling refers to that part of the code which can be reused and can be separated from the code but is part of it directly. Whereas cohesion is a measure of how closely related are all the responsibilities, data, and methods of a class/code to each other.

In particular, I would like to achieve a minimum of three specific things with the design and architecture of a framework:

• Keep things that have to change together in the code, as close together as possible.
• Allow unrelated things in the code to change independently (also known as orthogonality).

Orthogonality is a concept that allows combining the small components to get effective results. This eases the read/write programs. A more orthogonal design allows for fewer exceptions, symmetry, and consistency.

• Minimize duplication in the code.

The goals behind achieving loose coupling between modules are:

• Make the code easier to read
• Make our codes easier to consume by other developers by hiding the complexity
• Isolate potential changes to a small area of code
• Reuse code in completely new contexts

For More Information:
Decreasing coupling and increasing cohesion

We generally consider high cohesion to be a good thing for designing framework libraries like a key design considerations, but why?

Look at the following code. We need a connection string to connect to the database, and the connection string and database manipulation code is tightly coupled. Alright, what is the issue here? Now if we want to change the DSN, we need to change the code for each DSN:

```vba
Set Conn = CreateObject("ADODB.Connection")
Conn.Open("DSN=QT_Flight32")
Set rcRecordSet= Conn.Execute("SELECT order_number from Orders
    order by order_number desc")
rcRecordSet.MoveFirst
var_order_num = rcRecordSet.fields("Order_Number")
rcRecordSet.close

Conn.close
set rcRecordSet = nothing
set Conn = nothing
```

Let's rewrite this code to make it less coupled and highly cohesive. I have divided the code into two parts. The first part is a function that returns the DSN, and the second part uses that DSN and manipulates the database. Now if the user wants to access another database, the DSN will be different and the user just needs to change the parameter. The previous code is coupled and the following code is cohesive:

```vba
Function getDSN(database)
    If database = "access"
        DSN = "DSN=QT_Flight32"
    ELSEIF ...
End Function

Set Conn = CreateObject("ADODB.Connection")
Conn.Open(getDSN("access") )
Set rcRecordSet= Conn.Execute("SELECT order_number from Orders
    order by order_number desc")
rcRecordSet.MoveFirst
var_order_num = rcRecordSet.fields("Order_Number")
rcRecordSet.close
Conn.close
set rcRecordSet = nothing
set Conn = nothing
```

For More Information:
There is one more design consideration in the context of the QTP, that is, whether we should use actions or functions.

From my personal experience, we should use functions. Actions have many advantages and should be preferred in implementing any framework. Functions allow us to achieve the goals without functional decomposition. To create the framework, the generic library is required; for example, Logger, Error handler, and configuration utilities are required to achieve the following:

The usability can be enhanced by dividing the entire flow into small manageable pieces using actions or functions.

Key steps for designing the modular framework are as follows:

- Analyze the application
- Create the design
- Create Automation Test Repository
- Create the test-level components
- Create the script-level functions
- Integrate
- Test It

**Analyzing the application**

Analyzing the application defines the automation goals into the defined functions and operations of the intended application. The key considerations for the analysis are:

- What are the key functionalities of the application and how are they related to each other?
- What are key flows?
- Which is the least used functionality? Will it contribute to ROI?
- What are the goals that we want to achieve through automation?

**Creating a design**

Creating a design describes the desired features and operations in detail, including layouts, modules, rules, verification, and interaction among modules along with the process diagrams, pseudo code, and other documentation.
The key to achieving modularity is by decomposing the functionality and recombining the modules:

- **Decomposition**: The interaction of the user with the application is broken down into libraries, such as Functional library (login, createorder, and so on), Common library, OR, Test data, and environment.

- **Recombination**: The basic elements of interaction are recombined to follow a formal test plan using several levels of aggregation. The steps are aggregated to make sequences of the basic steps, and the sequences of basic steps are combined to make a scenario. The scenarios are aggregated to make a test suite.

After decomposition and recombination, the overall modular framework design will look like the following diagram:

### Setting up an environment

Perform the following steps to set up the test environment:

1. Create the structure, which is the same as creating the infrastructure for the test.
2. Create resources, including installation.
3. Create the folder structures or resources in the Test Management tool.
4. Ensure version control (good practice).
5. Create the configuration file.
Creating Object Repository
Create the OR by adding objects (Add objects to local repository or record the test step that automatically creates the test objects).

Creating test-level components
Perform the following steps to create test-level components:

1. Prepare the test data.
2. Create common functions/libraries.
3. Create a functional library.
4. Create test flows.
5. Add common components by loading the libraries.

Creating script-level functions
Perform the following steps to create script-level components:

1. Create steps manually.
2. Add the required programming logic.
3. Call the common functions (verification points).
4. Call the err handlers.

Integrating design elements
Bring all the pieces together into a testing environment, and ensure that the test automation is working end-to-end.

Test the framework design
Finally, we can perform the test to checks for errors, bugs, and interoperability.

Report the bugs, fix them, and retest.

Advantages of modular frameworks

• Functional decomposition allows us to divide and conquer the complexities
• Modularity eases designing, implementing, and debugging of the frameworks
• Provides standard interfaces for communicating with each other and allows the plugging in and removal of the new modules

For More Information:
Challenges for modular frameworks

The key challenges for creating modular frameworks are as follows:

- Require more technical know-how and effort to create generic modules
- For very specific modules, the cost of making interfaces is high
- For assemblers (integrators), it can be difficult to assess the quality and interaction of different modules
- It can be difficult to assemble (integrate) the modules
- The design creativity of a module designer can be limited because he needs to conform to the interface
- Less variation in products because of overuse of the same modules
- Total system performance may be suboptimal

The keyword-driven framework

Keyword-driven testing is also called table-driven testing or action-based testing. It is a software testing methodology.

Keyword-driven testing uses a spreadsheet to specify test cases in a specific format, usually in a table format. The functions are designed for each keyword. This keyword is stored in the column on a row of the table. For example, in the keyword-driven approach, each action has its corresponding function stored in the functional library. Driver scripts drive the entire flow, fetch the action, and call the corresponding function.

The keyword-driven testing approach

The keyword-driven approach is when the interaction of a user with an application is abstracted in the form of actions, and data is maintained in the external source.

Decomposition

The interaction of the user with the application is broken down to the basic elements (login, select flight …). In the keyword-driven approach, a keyword represents the business scenario that performs many actions on AUT or small actions repeatedly. The basic elements of interaction are referred to as steps.

For More Information:

Recombination

The basic elements of interaction are recombined to follow a formal test plan using several levels of aggregation. The steps are aggregated to make sequence(s) of the basic steps, and sequences of the basic steps are combined to make sequence groups.

Refer to the following screenshot that shows mapping of the actions from its library function using driver script. The interaction of a user with an application is abstracted in the form of actions and data maintained in the external source.

Separate the test cases from the scripts. The test cases are kept out of the scripts. These are fetched by the driver script and the keyword function is called as shown in the following screenshot:

<table>
<thead>
<tr>
<th>Action</th>
<th>MR</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>No</td>
<td>UserName=ashish;Password=mercury</td>
</tr>
<tr>
<td>CloseDolog</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>SelectFlightOptions</td>
<td>No</td>
<td>Passenger=p2;Trip=sneeway;from=Frankfurt;to=London;TravelClass=p1;Preference=p3;Blue Skies Airlines</td>
</tr>
<tr>
<td>SelectFlight</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>VerBookingDetails</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>SetNameMeal</td>
<td>Yes</td>
<td>FirstName=p1;LastName=p2;meal=Hindi,FirstName=p3;LastName=p4;meal=Hindi</td>
</tr>
<tr>
<td>GetCreditCardDetails</td>
<td>No</td>
<td>creditCard=American Express;cc_num=1234567890;exp_mn=12;exp_year=2010;firstname=p1;midname=p2;lastname=p3</td>
</tr>
<tr>
<td>GetDeliveryAddress</td>
<td>No</td>
<td>address1=aliax;address2=aliax;city=newyork;state=newyork;zip=11111;country=UNITED STATES</td>
</tr>
<tr>
<td>GetTicketLess</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>GetCopyAddress</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>BookFlight</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>GetConfirmationNumber</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>GetTotalTaxPrice</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>Home</td>
<td>No</td>
<td>NA</td>
</tr>
</tbody>
</table>

```
Select Case [Action]
Case "Login"
   If (trim(Data) <> "NA") Then
      ActLogin Data NA
   Else
      ActLogin
   End if
```

For More Information:
Understanding and Creating Frameworks

Once the keyword is found, the corresponding keyword library function is called as shown in the following screenshot:

```plaintext
Public Function ActLogin (ldata)
    'Author
    'Description
    'Create Date
    SynoPage "LoginPage","LoginPage"

    userdata = Split(ldata,";")
    name = Mid(userdata(0), InStr(userdata(0),"=") + 1 )
    'msgbox name
    password =Mid(userdata(1), Instr(userdata(1),"=") + 1 )

    Browser("LoginPage").Page("LoginPage").WebEdit("username").Set name
    Browser("LoginPage").Page("LoginPage").WebEdit("password").Set password
    Browser("LoginPage").Page("LoginPage").Image("Sign-In").Click
    WriteToLog logtilepath, "Pass","Login","username ashish and password mercury"
End Function
```

Development of the keyword-driven framework requires achieving the modularity first; apart from that, we need the keyword function library, test cases, and driver script. The keyword-driven framework is a modular framework plus test cases stored in an external source and driver scripts with a specific functional library, also called the keyword library.

Function decomposition is an important activity for designing the keyword-driven framework, and the decomposition guide to identify the keywords. Keywords are the reusable functions that perform one of the key action(s) on AUT, do a specific task, and complete it on their own terms. Another important aspect of the keyword-driven framework is test cases are separated from the test scripts.

The keyword-driven or test-plan driven method

This approach leverages the advantages of functional decomposition and separates the execution from the scripts. The test cases are defined in the spreadsheet containing the keywords. Each keyword has its corresponding function library that performs the action on the AUT.

For More Information:
In this method, the entire process is keyword driven, including the functionality. The keywords control the processing.

**Keyword-driven testing (KDT)** was created to simplify the creation of automated test cases and make them as much as manual test cases. KDT creates QTP-automated test scripts **On the Fly** based on information entered into a spreadsheet.

Test case representation will be in rows and columns. In the test scenario spreadsheet, each test case will perform an action.

![Note: In automated testing, the lowest level of granularity is the test case step. The step level is where each action or verification occurs.]

As per KDT, we need to prepare data spreadsheets for each locale because the functionality may differ based upon the different locales. Some locales have a specific coverage for commercial groups, individual businesses, and so on. It is difficult to handle these in the test script. Test suite is prepared to specify which locale/locales need to be executed, and the script is made generic. The script is driven based on the Excel sheet, which will take Excel data from the locale and generate the **On the Fly** script.

Key steps for designing the keyword-driven framework:

1. Analyze the application.
2. Create the design.
3. Create the Automation Test Repository.
4. Create the test-level components.
5. Create the scripts-level functions.
6. Integrate.
7. Test it.

We have already seen the key steps in creating the modular framework. The approach remains the same for the keyword-driven framework as well.

Let's find out about the automation repository in the keyword-driven approach. Overall, it contains data related to documents, test suite, and created test cases and libraries.

Test cases are also referred to as test scenarios or test groups, which contain the spreadsheet that specifies the test steps for execution.

For More Information:  
Understanding and Creating Frameworks

Common libraries contain the QTP library files and (.vbs, .qf1) files that control the entire KDT control flow, for example, driver scripts and library files (.qf1, .vbs).

Common libraries can be viewed as:

- Utility functions
- Navigation functions
- Support functions

**Environment libraries** contain .xml and .qf1 files that are used to set the environment.

**Error libraries** contain files that are used to catch the errors during execution and perform the necessary actions (.qf1 or .vbs file).

**Framework documents** contain all the data that is created for the framework, for example, enhancements to the initial framework or help files created for the framework.

**Object Repository** contains the .tsr object files created using QTP.

The following diagram shows how various layers and components work together to achieve the overall design:

![Diagram](image-url)
Generic flow in keyword-driven approach

The key steps and their sequence of execution in the keyword-driven approach are as follows:

1. Fetch the keyword as mentioned in the spreadsheet.
2. Build a list of the parameters from the step that follows.
3. Call the utility functions; the utility scripts will do the following:
   1. Call utility scripts with the input parameter-list received from the utility script.
   2. Call the driver script to perform specific tasks (for example, log in, select flight, and so on), calling user-defined functions if required.
   3. Report any errors to a test report for the test case.
   4. Exception handler scripts.
   5. Return to the driver script.
4. Repeat the steps from 1 to 3.

Advantages of the keyword-driven approach

The key advantages offered by the keyword-driven approach are as explained in the following sections.

Cost effectiveness

The keyword-driven automation framework reduces cost and time of the test design, automation, and execution. Keywords are highly reusable functions that represent the business scenarios or actions performed on the GUI. Each of these well-designed and tested functions provide good return on investment over a period of time.

Separating test cases allows executing test cases without modifying the scripts. When the flow of the application changes, just change the test cases and not the scripts.

Reusability

Keywords, utilities, and functions are built to achieve reusability. The entire framework is divided into layers that are integrated with each other.

For More Information:
Ease of maintenance

- The keyword-driven approach provides abstraction from the complexities and technical challenges; it is easy to maintain
- The robustness of the keyword-driven framework allows adapting the changes in GUI and test flows
- Allows focusing on the development of test cases without or with minimal changes to the scripts, functions, and utilities

Ease of execution

The keyword-driven approach allows executing and creating automated test cases for non-technical testers, business analysts, and SMEs (Subject Matter Expert) to write automated tests.

Test cases are separated from the scripts, and it is easy to prepare the test cases using the used keywords without knowledge of how they have been implemented.

Key challenges

Knowledge of designing is required.

New team members have to put in efforts to understand the framework and its design. Good documentation and knowledge sharing is required to overcome this issue.

The hybrid framework

The hybrid framework allows combining the two frameworks to leverage their strengths and remove their weaknesses. Most frameworks, which are developed, fall into this category that uses the function decomposition like modular frameworks and the data-driven approach.

Key steps in designing the hybrid framework are as follows:

1. Creating the folder structure.
2. Creating and storing automation resources.
3. Organizing and managing resources.
4. Integrating the frameworks.
5. Dry run.

For More Information:

Advantages of the hybrid framework
Hybrid framework allows leverages, and pulls the strengths of the other frameworks and eliminates their shortcomings, which suits the automation. Practically, most of the test automation solutions fall in this category.

Key challenges
Though it mitigates the weaknesses of the other approaches, it loses its generosity and is very specific to the test automation solution AUT, which reduces the reusability of its components across the multiple AUTs.

Business Process Testing
Business Process Testing (BPT) approach allows dividing the business processes into smaller reusable components that can used many times in the same or different test scripts; for example, the business process of buying a product is split into components such as log in, select product, add to cart, place order, and log out, which can be re-used in the same business process or different processes. The key advantage is it facilitates SMEs, Bas, and automation engineers to work and collaborate effectively. Some people call it a framework, but this is an approach rather than a framework. BPT is similar to the modular approach of creating test automation solutions using QTP and Quality Center.

Application-independent framework
The application-independent framework is a specific keyword-driven testing or table-driven testing. It identifies the keywords that are independent of the AUT, which can perform specific actions on the components of the AUT directly. The key difference between the keyword-driven framework and application-independent keyword-driven framework lies in the library. In the application-independent keyword-driven framework, the functional library is more generic or works directly on the generic components for AUT. The following is the data table:

<table>
<thead>
<tr>
<th>Window</th>
<th>WinObject</th>
<th>Action</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculator</td>
<td>button</td>
<td>Click</td>
<td>1</td>
</tr>
<tr>
<td>Calculator</td>
<td>button</td>
<td>Click</td>
<td>+</td>
</tr>
<tr>
<td>Calculator</td>
<td>button</td>
<td>Click</td>
<td>3</td>
</tr>
<tr>
<td>Calculator</td>
<td>button</td>
<td>Click</td>
<td>=</td>
</tr>
<tr>
<td>Calculator</td>
<td>Text</td>
<td>VerResult</td>
<td>4</td>
</tr>
</tbody>
</table>

For More Information:
The functionality of the AUT is specified in the data table. The preceding table allows calculating some operations on the calculator window, which is 1 + 3 = and verify the result. The instructions are as follows:

1. Click on the button (1) in the calculator window.
2. Click on the button (+) in the calculator window.
3. Click on the button (3) in the calculator window.
4. Click on the button (=) in the calculator window.
5. Verify the result from text in the calculator.

The action column lists the actions done with the mouse, keyboard, or specific functions. The data table should be mapped to generate the test step; for example, mouse click on the button, and the button is identified by the argument (1). The control name is given in the WinObject column, and the Window column contains the name of the application.

In QTP, the Object Repository stores the test objects and provides the logical name to it. We can use this logical name as a parameter to create the script; for example, we can store the object hierarchy in Excel with data. That is, utilizing the code to create the application-independent frameworks. The following diagram is an example of how we can achieve the application-independent implementation using excel and convert it into scripts. All the objects' names are provided in the column; in runtime, it fetches the object information and creates the scripts as shown in the following diagram:

In the preceding diagram, we can observe that the script creates the objects at runtime rather than the hardcoded scripts. This gives the application independence from the other applications.
Advantages of the application-independent framework

Application-independent keyword-driven framework inherits all the advantages of the keyword-driven framework, apart from the one that allows us to work on different applications without much changes.

Key challenges of the application-independent framework

Creating the application-independent framework requires more expertise to deal with the complexities in creating generic libraries.

The application-independent keyword-driven approach is usually designed for specific technologies and not for multiple technologies.

The most basic framework that is provided by QTP is the replay mechanism and easy steps to create the test steps. These steps are put into reusable functions and become the functional library in the modular and keyword-driven frameworks. Parameterizing the step and allowing it to iterate becomes the data-driven approach. Refer to the following diagram that shows the steps we performed in developing the one framework that became the base for developing the next one, for example, record and replay become the base for data-driven frameworks. One framework becomes the base or partial base to the next framework. Refer to the following diagram demonstrating how they are related:

<table>
<thead>
<tr>
<th>Framework Type</th>
<th>Key Steps</th>
<th>Modular</th>
<th>Data Driven</th>
<th>Record &amp; Replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Steps</td>
<td>Create Driver Scripts Create Test Structure Create Utility Functions Split scripts into keyword library Recording or Manual Creates scripts Replay the scripts</td>
<td>Create master scripts Recombine all modular components Create Utility Functions Create Test Structure Split script into functional library Recording or Manual Creates scripts Replay the scripts</td>
<td>Parameterize the test and iterate it Recording or Manual Creates scripts Replay the scripts</td>
<td>Record script Replay script</td>
</tr>
<tr>
<td>Key Components</td>
<td>Keyword Library Function library Utility Functions Driver scripts Test Structure</td>
<td>Master Scripts Function Library Utility Functions Test Structure</td>
<td>Scripts Data Source</td>
<td>Linear Scripts QTP Engine</td>
</tr>
</tbody>
</table>

For More Information:

Summary
This chapter describes the various concepts and approaches to build a framework and its components, and a structure that allows us to keep the resources uniformly accessible. This helps to achieve the automation goals and lower the maintenance cost of the test automation suite.

In the next chapter, we are going to discuss various ways to debug scripts, custom logging deployment, and maintaining the framework.
Where to buy this book
Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.