|  |
| --- |
| UFT 12.02 |
| RMB UFT Automation Framework |
| **AUT: http://newtours.demoaut.com/index.php** |

|  |
| --- |
| Das, Abhijit (Consultant)  1/10/2017 |

# Framework Architecture:

## Common Object Repository

## Library

### Application library

'1.testCleanUp

'2.invokeBrowser

'3.intLogin

'4.intLogout

'5.clickandNavigate

'6.validateMaxLength

'7.validateEmailFieldMaxLength

'8.validateCommentFieldMaxLength

'9.validateDefaultValue

'10.validateCommentDefaultValue

'11.validateEmailDefaultValue

'12.validatePhoneDefaultValue

'13.validateFaxDefaultValue

'14.objectExistanceCheck

'15.emailFieldValidation

### Report Library

'1. CreateResultFile

'2. LogResult

'3. TestCaseExecutiveSummary

### Common Library

'1.strFormatNowDate

'2.Initialization

## Test Data

## Test Script

## Controller (.xls)

## Driver (.vb)

## Log (.txt)

# AUT: http://newtours.demoaut.com/index.php

# Automation Code Guidelines

# . Introduction

Coding conventions are suggestions designed to help write code/scripts to improve the readability and standardize the structure and coding style. Using good coding conventions result in clear, precise, and readable code that is consistent with other language conventions as well.

Camel case is a standard identifier naming convention for several programming languages and as a standard we’ll be using the CamelCase naming convention in the rest of this document.

# . Naming Conventions

## Object Repository Logical Names

As a good practice, prefix the Logical Name of the object with underscore “\_” followed by its class type.

For example:

*txtFirstName* (where FirstName is of the class WebEdit)

**Table: Recommended Prefixes for Logical Names in Object Repository**

| **Subtype** | **Prefix** | **Example** |
| --- | --- | --- |
| WebEdit | txt | txt\_FirstName |
| WebList | lst | lst\_State |
| WebCheckBox | cbx | cbx\_AgreeTerms |
| WebRadioGroup | rdo | rdo\_CustomerType |
| WebTable | tbl | tbl\_Address |
| Frame | frm | frm\_SignIn |
| Image | img | img\_Submit |
| Link | lnk | lnk\_Join |
| DialogBox | dbx | dbx\_ConfirmDelete |
| Push Button | pbn | pbn\_OK |

## Data Table/Action Parameter Names

Data Table names for shared actions to be prefixed with *act\_<<ActionName>>*.

For e.g. Member Enroll shared action DataTable could be named as *act\_MemberEnroll.xls*

## Variable names in Action/VBscript

The first part of the variable name (the prefix that indicates the data type) should begin with a lowercase letter. All subsequent words in the name then begin with an uppercase letter. Consequently, variables have names such as intUserID and strUserLastName rather than IntUserID or struserlastname.

Below tabular lists various data subtypes recognized by VBScript, as well as the recommended prefix for variables designed to store each kind of data.

**Table: Recommended Prefixes for VBScript Data Subtypes**

| **Subtype** | **Description** | **Prefix** | **Example** |
| --- | --- | --- | --- |
| Boolean | Contains either True or False. | bln | blnIsUSCititzen |
| Byte | Contains an integer in the range 0 to 255. | byt | bytDepartmentNumber |
| Integer | Contains an integer in the range -32,768 to 32,767. | int | intNumberOfDirectReports |
| String | Contains a variable-length string. Strings can be made up of any alphanumeric characters. | str | strUserLastName |
| Object | Contains an object reference. An object variable represents an Automation object. | obj | objExcelSpreadsheet |
| Error | Contains an error number. | err | errFindFile |
| Array | Contains an array of variables. Because arrays are designed to hold multiple objects, array names should always be plural. To maintain consistency with the other naming conventions, arrays should have two prefixes: arr to indicate the array, and a second prefix to indicate the data type. | arr | arrstrUserAccountNames |
| Dictionary Object | Contains an Dictionary object reference | dic | dicMemberEnroll |
| Collection | Technically, a collection is not a variable subtype. However, it is listed in this table because you should use the col prefix to indicate collections. Collections are used extensively in system administration scripting. | col | colInstalledApplications |

## Naming Constants

Constants provide meaningful names for values that cannot be changed while a script runs. User-defined constants are typically named using all uppercase letters, with underscores separating individual words in the constant (for example, MASTER\_TESTDATA\_FILE or BROWSER\_MAIN). This formatting convention indicates that MASTER\_TESTDATA\_FILE is a constant and thus cannot be changed at any point during the running of a script. Because constants are not the same thing as variables, it makes sense to format their names in a different manner.

Scripting languages, automation objects, type libraries and other similar entities typically include intrinsic constants as well. For those intrinsic constants, follow the naming convention used in the language or object's documentation. For example, VBScript includes such intrinsic constants as VbGeneralDate (used in formatting dates) and VbAbortRetryIgnore (used in constructing message boxes). You should use these names rather than names such as VB\_GENERAL\_DATE.

## File Names

**QTP action naming convention**

Action: <<FunctionalArea>> (e.g.: MemberEnroll)

# . Coding Standards

## Programmatic Statements

**If statements**

The if-else class of statements should have the following form:

The single-line form (first syntax) can be used for short, simple tests. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug.

**If** condition **Then** statements [**Else** elsestatements ]

**or**

**If** *condition* **Then**

   [*statements*]

[**ElseIf** *condition-n* **Then**

   [*elseifstatements*]] **. . .**

[**Else**

   [elsestatements]]

**End If**

**For statements**

Repeats a group of statements for each element in an array or collection.

**For Each** *element* **In** *group*

   [*statements*]

   [**Exit For**]

   [statements]

**Next** [*element*]

Repeats a group of statements a specified number of times.

**For** *counter* **=** *start* **To** *end* [**Step** *step*]

    [*statements*]

    [**Exit For**]

    [*statements*]

**Next**

**While statements**

**While** *condition*

   Version [*statements*]

**Wend**

**Do While statements**

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

**Do** [{**While** | **Until**} *condition*]

   [*statements*]

   [**Exit** **Do**]

   [*statements*]

**Loop**

Or, this syntax can also be used:

**Do**

   [*statements*]

   [**Exit** **Do**]

   [*statements*]

**Loop** [{**While** | **Until**} *condition*]

**Case statements**

**Select Case** *testexpression*

   [**Case** *expressionlist-n*

      [*statements-n*]] **. . .**

   [**Case Else** *expressionlist-n*

      [elsestatements-n]]

**End Select**

## Commenting Code

* Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself
* Programs can have different styles of implementation comments like block, single-line, trailing, and end-of-line.

Comments should not be enclosed in large boxes drawn with asterisks or other characters.

Comments should never include special characters such as form-feed and backspace.

* Comments should contain only information that is relevant to reading and understanding the program.
* Avoid duplicating information that is present in (and clear from) the code
* In general, avoid any comments that are likely to get out of date as the code evolves.

## Function / Procedure Header

All functions and procedures should begin with a brief comment describing what they do. This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse, erroneous comments. The code itself and any necessary inline comments describe the implementation.

' ======================================================================

‘ Function Name :

' Created Date :

' Author :

' Description :

' Arguments :

‘ Return Value :

‘ Change Log :

'

' ======================================================================

## Script / Action Header

All QTP scripts / Actions should begin with a brief comment describing what they do as shown below.

' ======================================================================

‘ Script / Action Name :

' Created Date :

' Author :

' Description :

' Parameters :

‘ Change Log :

‘

' ======================================================================

## Indentation need and usage

Indentation is commonly used with control structure statements such as If Then and For Next. For example, the syntax shown in the following snippet is valid, but it is difficult to understand what the script is doing without reading the code several times and mentally tracing the program flow.

For Each strEvent in objInstalledLogFiles

If (strEvent.EventCode >= "529") and (strEvent.EventCode <= "539") Then

If (strEvent.EventCode <> "538") Then

intEventTotal = intEventTotal +1

End If

End If

Next

Compare that with the revised code shown in the following script snippet. This is the same code, but each new control structure statement is indented four spaces (or Tab)

For Each Event in InstalledLogFiles

If (strEvent.EventCode >= "529") and (strEvent.EventCode <= "539") Then

If (strEvent.EventCode <> "538") Then

intEventTotal = intEventTotal +1

End If

End If

Next

## Usage of Blank Lines and White spaces

White space, achieved with blank lines, blank spaces, and character indentation, provides visual cues that help delineate and identify program flow and program sections.

For i = 1 to 95

intTotal = intTotal + i

Next

MsgBox intTotal

For example, you might want to use blank spaces:

* Before and after operators.

For Example:

* If intCheck>=-3 and intCheck<>7 then
* If (intCheck >= -3) and (intCheck <> 7) then
* In parameter lists.

For Example:

* objThing.Change("Large","Heavy","Red")
* objThing.Change("Large", "Heavy", "Red")

# . Coding Good Practices

Below are the some of the good coding practices that should be followed:

* Use switch case rather than complex if-else structure whenever possible
* Initialize all variables/objects to a default value
* Avoid recursion whenever possible
* Avoid global variables whenever possible
* Place all constants in a particular code file
* Read through config files/xml files all the data that are bound to change
* Modularity - breaking code in specific logical modules for better maintainability
* Create common libraries to create a set of re-usable code. Library functions can consists of utility functions, set of repeated user actions, Report Logging functions etc.
* Avoid using hard coded wait(x) statement. Wait statement waits for full x seconds, even if the event has already occurred. Instead use .sync or exist statement. While using exist statement always have a value inside it.

For example: Object*.Exist(10)* Here QTP will wait max till 10 seconds and if it finds the object at (say) 3 secs , it will resume the execution immediately thereby saving your precious time. On the other hand if you leave the parenthesis blank, QTP would wait for object synchronization timeoutyou have mentioned under File > Test Settings > Run Tab.

* When to use a Recovery Scenario and when to us on error resume next?

Recovery scenarios are used when you cannot predict at what step the error can occur or when you know that error won't occur in your QTP script but could occur in the world outside QTP, again the example would be "out of paper", as this error is caused by printer device driver. "On error resume next" should be used when you know if an error is expected and don’t want to raise it, you may want to have different actions depending upon the error that occurred. Use err.number & err.description to get more details about the error.

* Always use relative paths instead of absolute paths. This makes easy to port test suite to different environment / location.

# . Sample of Code adhering to standards

The following code adheres to the above mentioned coding conventions

' =========================================================================

‘ Function Name : intFindUser

' Created Date : dd-mmm-yyyy

' Author : Joe

' Description : Locates the first occurrence of a specified user in the UserList array.

' Arguments : strUserList()-the list of users to be searched.

strTargetUser: the name of the user to search for.

‘ Return Value : The index of the first occurrence of the strTargetUser in the strUserList array. If the target user is not found, return -1.

‘ Change Log :

' =========================================================================

Function **intFindUser** (strUserList(), strTargetUser)

Dim i ' Loop counter.

Dim blnFound ' Target found flag

intFindUser = -1

i = 0 ' Initialize loop counter

Do While i <= Ubound(strUserList) and Not blnFound

If strUserList(i) = strTargetUser Then

blnFound = True ' Set flag to True

intFindUser = i ' Set return value to loop count

End If

i = i + 1 ' Increment loop counter

Loop

End Function

# Unique Features of the Framework

* **Selective Script Run** 
  + Desired script is executed by changing the *Execute* column in controller excel file value to ***yes***.
* **Browser Compatibility** 
  + Tested in Chrome and IE.
* **Environment Based**
  + Individual script is executed by changing the *Environment* column value to ‘***DEV****’* , ‘***SIT***’ , ‘***UAT***’ accordingly.
* **Property Validation** 
  + Does not require code change; changing the values in property field sheet will suffice.
* **Executive Summary**
  + A summarized report of all the test case along with dynamically generated ***Pie chart*** visualization.
* **Auto E-Mailer** 
  + After the execution completes, the Executive summary is emailed to listed email id.
* **Screen Shot** 
  + Capture’s the screenshot of failed scenarios.

# Languages and Tools Used

## VB Scripting

## Batch Scripting

## HP-UFT 12.02

## Chrome UFT Add-On

## HTML

# Available Scripts

# Future Scope

* + 1. **Adding the individual result file in portal and sharing the link in mail**

# References

## Guru99.com