OBJECT ORIENTED TEST MODULES

This topic is going to demonstrate an alternate yet highly manageable, flexible and robust way of creating test modules that facilitate for quick and easy maintenance. Creation of these centralized objects is based upon executing tests with entry and exit conditions for each module, with customized CheckPoints, Output value objects and reporting in each component of the module.

DESIGN PATTERNS IN QTP

TABLE OF CONTENTS

INTRODUCTION

- What is a test module?
  - A test module is a logical grouping of one of more business processes
    - ie. A test module can represent one or more functionality
  - A test module can be an Action, Function, Subroutine, Business Component or even a Class
  - Can be used for data-driven frameworks – driving business processes with different data-sets
  - Examples:
    - Login
    - FindFlight
    - BookFlight
    - Logout
- Why use a Class object as a test module?
  - Keeps all methods of a functionality intact
    - This can simplify maintenance – you find all methods in a single place
    - Decreases code review time & time required to change/update code by team-members
  - When stored in a function library, it can be called by any script (similar to functions)
  - Methods and Variables can be Encapsulated
    - Only methods that need to be interacted with are revealed to other objects
    - Limited number of methods are called per object – less change required

COMPONENT: ENTRY CONDITIONS

- An entry condition determines if the module is 'ready to execute'
  - True/False [Boolean]
- Recommendation: Use an entry condition to verify mandatory fields
  - Mandatory fields are the ones that are pertinent for testing to move forward
  - Not to be confused with the required fields of the AUT!
- There can be multiple entry conditions for a module

```
Class Login
    'Entry Condition 1
    Private Function isPageLoaded() 'As Boolean
        'Verifies if the module is viewing
        'the correct page
    End Function

    'Entry Condition 2
    Private Function isGUIContextLoaded() 'As Boolean
        'Verifies if all the mandatory GUI
        'objects exist
    End Function

    'Entry Condition n
    Private Function anotherEntryCondition() 'As Boolean
        'Code
    End Function
End Class
```

COMPONENT: OUTPUT VALUE

- Output values to other classes or test components
  - It can be a string, integer, object etc.
  - Values can be used later for inputs at different points in the run session
  - Note: Output values are only available for the duration of the run session
- Creating output values with collections can enable creating a single point of accessing multiple output values (see below)

```
Class Login

    'Output Value Object
    Public Function [Output.Value]()
        Set [Output.Value] = CreateObject("Scripting.Dictionary")

        On Error Resume Next
            With [Output.Value]
                .Add "title", Browser("Idhasoft")_
                    .GetROProperty("title")
                .Add "bgColor", Page("Idhasoft")_
                    .Object.bgColor
            End With

            If Err.Number <> 0 Then Set [Output.Value] = Nothing
        On Error Goto 0
    End Function

End Class
```

COMPONENT: CORE ACTION

- The core action implements the main business process
  - True/False [Boolean]
- There can be multiple core actions in a module
  - Example: The login section has 2 login fields each: End-User & Admin
- Core action is executed only if the Entry Condition is satisfied

```
Class Login

    'Core Action: Login
    Private Function Login(user, password) 'As Boolean
        Login = False

        With Page("App")
            .WebEdit("user").Set user
            .WebEdit("password").SetSecure password
            .WebButton("login").Click
        End With

        If Page("NewPage").Exist(5) Then Login = True
    End Function

End Class
```

COMPONENT: MODULE RESULT

- Determines if the module passed
  - True/False [Boolean]
- It can contain the overall flow of the module
  - If flow changes for any module, changes are only required in a single place
  - Flow can also be distributed between other methods – requires changes in multiple areas

```
Class Login

    'Module Result
    Public Property Get Result 'As Boolean
        Result = False

        'If Page = Correct & all Mandatory Objects exist..
        If isPageLoaded And isGUIContextLoaded Then
            'If user logs in successfully..
            If Login("user", "password") Then
                'Result = Pass
                Result = True
            End If
        End If
    End Function

End Class
```

Implementing Try-Catch-Finally in VBScript

Using Functions

```
Function fnError()
    Err.Raise 1002
    Print "This line will not execute."
End Function

Function TryCatchFinally(ByRef fn)
    On Error Resume Next
        Try fn : CatchFinally
End Function

Function Try(ByRef fn)
        Set thisFn = GetRef(fn)
        Print "Entering Try.." & vbNewLine
        thisFn
End Function

Function CatchFinally()
        If Not Err.Number = 0 Then
            Print "Entering Catch.."
            Print "Error: " & Err.Description &_
                vbNewLine
        End If

        Print "Entering Finally.."
    On Error GoTo 0
End Function
```

Using Classes

```
On Error Resume Next

Class TryCatchFinally

    Private Sub Class_Initialize 'Try
        Print "Entering Try.." & vbNewLine

        Err.Raise 1002
        Print "This line will not execute if " & _
            "there is an error"
    End Sub

    Private Sub Catch 'Catch
        If Err.Number = 0 Then Exit Sub

        Print "Entering Catch.."
        Print "Error: " + Err.Description  &_
            vbNewLine
        Err.Clear
    End Sub

    Private Sub Class_Terminate : Catch 'Finally
        Print "Exiting.."
    End Sub

End Class
```

IMPLEMENTING HANDLER CLASSES

- Handler classes implement test modules (utility classes)
  - Contain Try-Catch-Finally mechanism
  - Each test module can have a unique or a shared handler object

```
Class Login
    'Entry/Exit Conditions, Output Values, Core Action, Result
End Class

'Instance of [Class] Login
Dim NewLogin : Set NewLogin = New Login

Class Handler
    Private Sub Class_Initialize
        Print "Entering Try.."
        Print NewLogin.Result 'Get Module Result
    End Sub

    Private Sub Catch
        Print "Entering Catch.."
        Print "Error caught: " & Err.Description
    End Sub

    Private Sub Class_Terminate : Catch
        Print "Entering Finally.."
    End Sub
End Class
```

THANK YOU!!