Appendix A. Design Patterns

In this appendix, we will cover additional design patterns:

- Auxiliary classes and functions
- Action patterns
- Runtime data patterns

Auxiliary classes and functions

The following are auxiliary classes and functions' design patterns that provide additional functionality (not included within the main chapters):

- `AssertResult`: This design pattern checks whether the result triggers a predefined action:

```
Function ASSERT_RESULT(ByVal iResult)
    ' ------------------------------------------------------------
    ' Function    :    ASSERT_RESULT
    ' Purpose     :    Checks if the result triggers a predefin
    ' Args        :    ByVal iResult
    ' Returns     :    The value of iResult (unless the run session
    '                  terminated)
    ' ------------------------------------------------------------
    ASSERT_RESULT = CLng(iResult)
    If CLng(iResult) <> CLng(micPass) Then
        Reporter.ReportEvent micWarning, "ASSERT_RESULT", "The acti
        Execute(Environment("ON_FAILURE") & "(" & CStr(CLng(iResult
    End If
End Function
```

- `InfoClassInstance`: This design pattern prints a log message relating to the instance of an object:

```
Function InfoClassInstance(ByVal p, ByVal msg)
    '------------------------------------------------------------
    'Description:    Prints a log message relating to an object
    'Arguments    :
    '               p - a reference to the instance
    '               msg - a string
    'Usage        :    For example, in a Sub Class_Initialize withi
    '               InfoClassInstance(me, "Loaded successfully")
    'Changes Log:
    '------------------------------------------------------------
    Print C_OBJ_OF_CLASS_MSG & typename(p) & msg & " at " & Timesta
End Function
```

- `GetClassInstance`: The following design pattern returns an instance of a specific class:

```
Function GetClassInstance(oInst, ByVal sClass)
    ' ------------------------------------------------------------
    ' Function    :    GetClassInstance
    ' Purpose     :    Returns an instance of the specified Class
    ' Args        :    byRef oInst (output variable to return the i
    '                  ByVal sClass (name of requested Class)
    ' Returns     :    0 (success), 1 (failure)
    ' ------------------------------------------------------------
    GetClassInstance = 0
    On Error Resume Next
    Execute "Set oInst = new " & sClass
    If Err.Number <> 0 Then
        Set oInst = Nothing
        GetClassInstance = 1
        reporter.ReportEvent micFail, "GetClassInstance", "Failed t
    End If
End Function
```

- `GetIterations`: This design pattern returns with a list of iterations for an array:

```
Function GetIterations(ByVal sIterations)
    ' ------------------------------------------------------------
    ' Function    :    GetIterations
    ' Purpose     :    Get array with list of iterations
    ' Args        :     ByVal sIterations - A comma and hyphen sepa
    '                    string list with numbers of iterations to
    ' Returns     :     A System.Collections.ArrayList
    ' ------------------------------------------------------------
    ' Usage        :    Set DotNetArray = GetIterations("1,3,7-9,1
    '                  Print DotNetArray.Count
    '                  For each item in DotNetArray
    '                      Print item
    '                  Next
    ' ------------------------------------------------------------
    Dim arrRange, min, max, i, j
    Dim arrIterations    : Set arrIterations = CreateObject("System
    Dim arrTmp           : arrTmp = Split(sIterations, ",")

    'Parse array with iterations
    For i = 0 To Ubound(arrTmp)
        arrRange = Split(arrTmp(i), "-")

        If UBound(arrRange) = 1 Then '--- If is a Range
            min = arrRange(0)
            max = arrRange(1)
            If min > max Then
                Call SwapArgs(min, max)
            End If

            For j = min To max
                arrIterations.Add j
            Next
        Else '--- A single numeric value
            arrIterations.Add arrTmp(i)
        End If
        '--- Dispose of temporary range array
        Erase arrRange
    Next
    '--- Dispose of temporary array
    Erase arrTmp
    '--- Return DotNet array
    Set GetIterations = arrIterations
End Function
' ------------------------------------------------
```

- PadNumber: This design pattern pads a number string with zeros:

```
Function PadNumber(iNum, ByVal iMax)
    ' ------------------------------------------------------------
    ' Function    :    PadNumber
    ' Purpose     :    Pad a number with zeroes
    ' Args        :    ByRef iNum (the number to be padded)
    '                  ByVal iMax (the max value of the range)
    ' Usage       :    PadNumber(3, 100) will return the string "0
    ' Returns     :    String
    ' ------------------------------------------------------------
    'Validates the arguments - If invalid Then it returns the value
    If (Not IsNumeric(iNum) or Not IsNumeric(iMax)) Then
        PadNumber = iNum
        Exit Function
    End If
    If (Abs(iNum) >= Abs(iMax)) Then
        PadNumber = iNum
        Exit Function
    End If

    PadNumber = String(Len(CStr(Abs(iMax)))-Len(CStr(Abs(iNum))), '
End Function
```

- Timestamp: This design pattern returns a time stamped string:

```
Function Timestamp()
    ' ------------------------------------------------------------
    ' Function    :    Timestamp
    ' Purpose     :    Build a timestamp string
    ' Args        :    N/A
    ' Returns     :    String
    ' ------------------------------------------------------------
    dim sDate, sTime
    sDate=Date()
    sTime=Time()
    Timestamp = Year(sDate) & _
                    PadNumber(Month(sDate), 12) & _
                    PadNumber(Day(sDate), 31) & "_" & _
                    PadNumber(Hour(sTime),24) & _
                    PadNumber(Minute(sTime), 60) & _
                    PadNumber(Second(sTime), 60)
End Function
```

- CNum: This design pattern returns values based on coalescing operators:

```
Class CNum
    Private m_value

    Public Function [=](n)
        value = n
    End Function
    Public Function [++]()
        value = value+1
        [++] = value
    End Function

    Public Function [--]()
        value = value-1
        [--] = value
    End Function
    Public Function [+=](n)
        value = value+n
        [+=] = value
```

```
    End Function
    Public Function [-=](n)
        value = value-n
        [-=] = value
    End Function
    Public Function [*=](n)
        value = value*n
        [*=] = value
    End Function
    Public Function [/=](n)
        value = value/n
        [/=] = value
    End Function
    Public Function [\=](n)
        value = value\n
        [\=] = value
    End Function
    public default Property Get Value()
        Value = m_value
    End Property
    public Property Let Value(n)
        m_value = n
    End Property
    sub class_initialize()
        value = 0
    End sub
End Class
```

- `[As Num]`: This design pattern returns a string as a number:

```
Function [As Num](n)
    Set [As Num] = new CNum
    If isnumeric(n) Then [As Num].Value = n
End Function
```

`]`: This design pattern returns an incremented string number
ıe:

```
Function [++](n)
    Dim i
    Set i = [As Num](n)
    i.value = n
    i.[++]
    [++] = i
End Function
```

`]`: This design pattern returns a decremented string number
ıe:

```
Function[--](n)
    Dim i
    Set i = [As Num](n)
    i.value = n
    i.[--]
    [--] = i
End Function
```