Don’t blame VBScript

VBScript is a very popular automation language. Microsoft VBScript Engine, which is available for free, is one of the reasons, another - on the surface VBScript is extremely simple. But I admit that switching to VBScript from C was hard at times. It required unlearning approach.

And yet VBScript is powerful enough so that a seasoned programmer used to Object-Oriented paradigm can use the same approach with a little bit of creativity.

Class Constructor

Each VBScript class has a constructor which is automatically called when a new object instance is created.

```
Private Sub Class_Initialize()
' put your custom initialization code
  End Sub
```

Class Destructor

Each VBScript class has also a destructor which is automatically called when life span of the object comes to an end.

```
Private Sub Class_Terminate()
' put your custom initialization code
  End Sub
```

Virtual Properties

Special methods Property Get, Property Let, and Property Set allow using object properties that do not actually exist.

```
Class Point2D
  '' Properties

  Private p_X
  Private p_Y
  '-----------------------

  '' Methods
  'Constructor - called automatically
  Private Sub Class_Initialize()
    X = 0
    Y = 0
  End Sub
  '-----------------------
  'Destructor - called automatically
  Private Sub Class_Terminate()
  End Sub
  '-----------------------
  'A pair of methods to access private property X
  Property Get X
    X = p_X
  End Property
  Property Let X(ByVal in_X)
    p_X = in_X
  End Property
  '-----------------------
  'A pair of methods to access private property Y
  Property Get Y
    Y = p_Y
  End Property
  Property Let Y(ByVal in_Y)
    p_Y = in_Y
  End Property
  '-----------------------
  'A pair of methods to access virtual properties:
  'Point's Polar coordinates
  Property Get Polar_R
    Polar_R = Sqr(p_X*p_X + p_Y*p_Y)
  End Property
  Property Get Polar_Phi
    Polar_Phi = Atn(p_Y/p_X)
  End Property
  '-----------------------
End Class
```

Inheritance through Delegation

VBScript does not allow declaring a class through derivation. However, a programmer can declare a property (or multiple properties) and initialize it (them) with reference(s) of ancestor object(s). That will give the desired access to the properties and methods of ancestor objects.

```
Class ScreenPoint
  '' Properties
```

Off

Categories

Accessibility (3)
Automation (22)
Availability (1)
Bias (8)
Bug Reports (22)
Career Tips (8)
Compliance (1)
Database (3)
Disaster Recovery (2)
Documentation (16)
Estimation (1)
Exploratory Testing (24)
Free Tools (73)
Heuristics (51)
Internationalization (5)
Learning about the product (7)
Mind Mapping (11)
Performance Testing (34)
Playing Well With Others (40)
Practicing Testing (8)
Regression Testing (2)
Security Testing (13)
SEO (1)
Skilled Bug Investigation (13)
Stress Testing (1)
Technical Tricks (1)
Test Data (15)
Test Management (33)
Test Oracles (2)
Test Planning (32)
Test Theory (5)
Testability (1)
Testing mobile (6)
Testing Techniques (16)
Time Savers (29)
Tools under $100 (9)
Uncategorized (2)
Unit Testing (2)
Usability (13)
Web Testing (31)

Authors

Albert Gareev's website
Anne-Marie Charrett's blog
Jonathan Kohl’s blog
Julian Harty’s website
Karen Johnson’s blog

```
  'Ancestor Point2D
  Private P2D
  'Point color
  Private Color
  '----------------------

  '' Methods
  'Constructor - called automatically
  Private Sub Class_Initialize()
    Set P2D = new Point2D
  End Sub
  '----------------------
  'Destructor - called automatically
  Private Sub Class_Terminate()
    Set P2D = Nothing
  End Sub
  '----------------------
  'A pair of methods to access private property X
  Property Get X
    X = P2D.X
  End Property
  Property Let X(ByVal in_X)
    P2D.X = in_X
  End Property
  '----------------------
  'A pair of methods to access private property Y
  Property Get Y
    Y = P2D.Y
  End Property
  Property Let Y(ByVal in_Y)
    P2D.Y = in_Y
  End Property
  '----------------------
End Class
```

Dynamic Code and Callbacks

To demonstrate this feature, I first prompt you to read about Eval function and ExecuteGlobal statement in VBScript.

```
Public Function CTZ()
  sGlobalValue = "ExecuteGlobal successful"
End Function
Public sGlobalValue
Public Function TestExecuteGlobal()
  Dim sCallName, sLocalValue
Dim intRC, boolRC
  sGlobalValue = "Testing ExecuteGlobal"

  sCallName = "CTZ()"

 boolRC = True
 On Error Resume Next
  ExecuteGlobal sCallName
  intRC = Err.Number
 On Error GoTo 0
 If intRC <> 0 Then boolRC = False

  If Not boolRC Then
    sGlobalValue = "ExecuteGlobal failed"
  End If

  MessageBox(sGlobalValue)
End Function
```

Thus, you can add functions and any definitions (including class definitions) dynamically, during run-time - which allows referring undefined methods and objects!

There are also other magic tricks, as overloading and access to external class libraries. Hopefully this post has been a useful introduction to ignite your interest to automation tricks with VBScript.

Filed under: Automation | Comments Off

Comments (1) | Trackbacks (0) | (subscribe to comments on this post)

Eusebiu Blindu
November 1st, 2010 - 15:52

It reminds me of Rational Robot

« Bookmark Current Tab Set | Excel-based Test Reports (2) »

Go to top ☐

Mike Kelly's blog

Pages

About