G22.3033-002 Scripting Languages
5/29/2008
Objects in VBA
Properties, Call-backs

© Martin Hirzel

Outline

- Using objects in VBA
- Application-embedded scripting
- Classes and properties
- Visual programming and callbacks

Using Objects

Dim a1 As Apple ' declare variables a1 and a2
Dim a2 As Apple ' of class Apple
Set a1 = New Apple ' allocate new objects, assign
Set a2 = New Apple ' references (need "Set" keyword)
a1.color = "green" ' set property, differently
a2.color = "red" ' for a1 and a2
a1.prepare("slice") ' call method, passing
a2.prepare("squeeze") ' string parameter

Classes and Objects

VBA

Abbreviated Member Access

With ActiveWindow.Selection.SlideRange.Shapes.Title ' call method .Flip (msoFlipVertical) ' assign to property .Rotation = 15 End With

Properties vs. Fields

- Both: dot notation look&feel
  - Writable: a1.color = "red"
  - Readable: Debug.Print a1.color
- Properties only: associated behavior
  - E.g., update graphical representation
- Properties only: may be indexed, like arrays
  - cake.ingredient("topping") = a1
- Other languages with properties:
  - E.g., PHP, Delphi, C#
Common Uses of Properties

### Simple (field-like)
- Visual update
- Invariant checking
  - Filter illegal values
  - Read-only
  - Copy on write
- Logging

### Indexed (array-like)
- Collections
  - Resizable array
  - Hash map
- Persistence
  - File
  - Database
  - Cookie

---

Collections

```vba
Dim col As Slides
Set col = ActivePresentationSlides
Dim i As Integer
Debug.Print "for-loop, indexed property access"
For i = 1 To col.Count
  Debug.Print col.Item(i).Name
Next i

Debug.Print "for-loop, default property access"
For i = 1 To col.Count
  Debug.Print col(i).Name
Next i

Dim s As Slide
Debug.Print "for-each loop"
For Each s In col
  Debug.Print s.Name
Next s
```

---

Progressive Disclosure

- We only looked at how to *use* classes, but not how to *define* classes
  - That's sufficient for most VBA tasks!
- Language design encourages this:
  - Learn small subset of language to do most important tasks
  - Learn a little more to do a little more
- *Progressive disclosure* user experience

---

Outline

- Using objects in VBA
- Application-embedded scripting
- Classes and properties
- Visual programming and callbacks

---

Structure of a VBA Application

- **Projects**
  - Files: ppt, xls, doc
  - Normal dot for Word
  - Personal Macro Workbook for Excel (Window→Unhide)
- **Modules**
  - Regular module
  - Class module
  - Form with code sheet
- **Subroutines**
  - Function or Sub (Macro)
Scopes and Visibility

- Project dependencies
  - Visual Basic Editor → Tools → References
  - Then, can call across projects: `project.module.subroutine`
  - Cyclic references are not allowed
- Visibility
  - “Instanting” property of class module
  - Public/Private modifiers of declarations

Powerpoint Object Model

- The complete object model is much larger
- See Visual Basic help in editor
- Also in MSDN library:
  - Office development
  - Microsoft Office 2003
  - Office 2003
  - VBA reference
  - Powerpoint help
  - Object model

Object Model

- Object-oriented API for embedded scripts
- Other examples:
  - Object models for other Microsoft apps
  - DOM = document object model for XML

Defining Classes

```vba
Public color As String
Public weight As Double

Public Function pluck() As String
    pluck = Me.color & " apple"
End Function

Public Sub prepare(how As String)
    Dim dish As String
    Select Case how
        Case "slice"
            dish = "salad"
        Case "squeeze"
            dish = "juice"
        Case Else
            dish = "dessert"
    End Select
    Debug.Print "yum, " & pluck() & " " & dish
End Sub
```

Documentation of OO in VBA

- MSDN library
  - Development tools and languages
  - Visual Studio 6.0
  - Visual Basic 6.0
  - Product documentation
  - Using Visual Basic
  - Programmer’s guide
  - Part 2: What can you do with Visual Basic
  - Programming with objects
Defining Properties

• Syntax
  - Property Get id([arg]) .. End Property
  - Property Let id([arg], arg) .. End Property
  - Property Set id([arg], arg) .. End Property

• Remarks
  - To return value from Get, assign to id
  - Let assigns regular value, Set assigns object
  - Extra arguments, if any, are for indexing
  - Each routine can be Private or Public
  - Public variable is just shortcut for routines

Inheritance

Class module "Fruit" (Interface)  
Would be abstract in other languages

"Implements/" in subclass identifies superclass
Override private
with mangled name
Class module "Apple" (subclass)

VBA

Inheritance in VBA

Public weight As Double
Public Fruit_weight() As String 'empty routine
Public Function pluck() As String 'empty routine
End Sub

Public color As String
Private wght As Double
Private Property Let Fruit_weight(ByVal RHS As Double)
    wght = RHS
End Property
Private Property Get Fruit_weight() As Double
End Property
Private Function Fruit_pluck() As String
    Fruit_pluck = color & " apple"
End Function
Private Sub Fruit_prepare(how As String)
    Debug.Print how & "d" & Fruit_pluck()
End Sub

Pure Interface Inheritance

• Interface inheritance (in VBA):
  - Subclass must provide own implementation for all methods in superclass
  - Can use object of subclass where object of superclass is expected
  - Implementation inheritance (not in VBA):
    - Subclass can leave some methods from superclass unchanged
    - Those method implementations are reused
  - To reuse code in VBA, must call it by hand

Subtyping Example

Dim someFruit As Fruit
If Then
    Set someFruit = New Apple
Else
    Set someFruit = New Banana
End If
' compiler knows that the method exists, 
' even if it doesn't know which version 
' will get called
someFruit.prepare("slice")

Missing OO Features in VBA

• Implementation inheritance
• Constructors
  - Can write Class_Initialize() method
  - Or write subroutine in separate module to allocate new instance and initialize it
• Static fields and static methods

⇒ OO features changed in VB.NET

© Martin Hirzel
Outline

- Using objects in VBA
- Application-embedded scripting
- Classes and properties
- Visual programming and callbacks

Using Dialogs

- Dialog = window to request user input
- User form = module defining dialog

1. Loading is optional
2. Name of module
3. Display dialog box, wait for user input, run event handlers

4. Unloading is optional
5. Retrieve user input

VBA Using Dialogs

VBA Using Dialogs

VBA Using Dialogs

VBA Using Dialogs

Hungarian Notation

- Convention: start identifier with indication of type

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chk</td>
<td>Check box</td>
</tr>
<tr>
<td>cmd</td>
<td>Command button</td>
</tr>
<tr>
<td>fmx</td>
<td>User form</td>
</tr>
<tr>
<td>fmx</td>
<td>Frame</td>
</tr>
<tr>
<td>lst</td>
<td>List box</td>
</tr>
<tr>
<td>cmb</td>
<td>Combo box</td>
</tr>
<tr>
<td>opt</td>
<td>Option button</td>
</tr>
<tr>
<td>lbl</td>
<td>Label</td>
</tr>
<tr>
<td>txt</td>
<td>Text box</td>
</tr>
</tbody>
</table>

Dialog Example

- Dialog = window to request user input
- User form = module defining dialog

VBA Dialog Example

VBA Dialog Example

VBA Drag&Drop Dialog Design

- Right-click on project → Insert → User form
- View → Properties Window
  - Change name, caption, maybe width/height
- View → Toolbox
  - Drag and drop controls onto user form
  - Rename, resize, set initial value
  - Set "default" or "cancel" property for buttons
- Right-click on form → Tab order
  - Order when user "tabs through" controls

VBA Drag&Drop Dialog Design

VBA Drag&Drop Dialog Design

Writing Event Handlers

Sub paintLemonStar(lines As Integer)
  Dim n As Integer, i As Integer
  For i = 0 To n
    Dim l As Shape
    Set l = ActiveWindow.Selection.SlideRange.Shapes.AddLine(
      _ BeginX:=300, BeginY:=200 + i * 100 / n, _
      EndX:=400, EndY:=300 - i * 100 / n)
    l.Line.ForeColor.RGB = RGB(i * 255 / n, i * 255 / n, 0)
  Next i
End Sub

Private Sub cmdPaint_Click()
  paintLemonStar CInt(txtNumberOfLines.Text)
  Hide
End Sub

Private Sub cmdCancel_Click()
  Exit Sub
End Sub

© Martin Hirzel
Code Sheets

- Right-click → View code
  - Just like other modules
  - Contains event handlers as well as regular subroutines
- Double-click on control
  - Goes to handler, creating it if necessary
  - Caution: handler name not changed automatically with control name!
- Dynamic dialog: assign properties at runtime
  - E.g., `Visible=True` reveals hidden controls
  - `UserForm_Initialize` sets default, e.g. for list box

Call-backs

![Call-backs Diagram](image_url)

- **VBA code**
  - Interpreter
  - User

- `frmLemonStar.Show()`
  - wait for user input
  - edit text box
  - click button

- `cmdPaint_Click()`
  - return from handler
  - return from "Show"

Common Controls and Events

- Controls:
  - Label, TextBox, ComboBox, ListBox, CheckBox, OptionButton, ToggleButton, Frame, CommandButton, TabStrip, MultiPage, ScrollBar, SpinButton, Image
- Events:
  - Change, Click, DblClick, Enter, Exit, Initialize, KeyPress, SpinDown, and many more
- In editor for user form code sheet:
  - Left drop-down list: control objects on this form
  - Right drop-down list: events on that object

Callback Mechanisms

<table>
<thead>
<tr>
<th>VBA form</th>
<th>Subroutine in form with mangled name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>WithEvent / RaiseEvent statements</td>
</tr>
<tr>
<td>Perl, Python, JavaScript</td>
<td>Pass anonymous function (lambda)</td>
</tr>
<tr>
<td>C, C++</td>
<td>Pass function pointer</td>
</tr>
<tr>
<td>SmallTalk</td>
<td>Pass object on which to call &quot;()&quot; operator</td>
</tr>
<tr>
<td>PHP</td>
<td>Pass code block</td>
</tr>
<tr>
<td></td>
<td>Pass name of function as string</td>
</tr>
</tbody>
</table>

Events on Classes

- In event source class `idsrc`:
  - `Public Event id_event(arg*)`
  - `RaiseEvent id_event(expr*)`
- In event sink module:
  - `WithEvents id_handler As idsrc`
  - `Sub id_handler_id_event(arg*)` — End Sub

Reusing Dialog Boxes

- Don’t write "open file" dialog from scratch!
- `Application.Dialogs(id).Show`
  - E.g., `id = xlDialogOptionsGeneral`
  - To find others: Help → Visual Basic → Search "build-in dialog argument"
- **Display** instead of **Show** prevents handlers
  - Return value: -2=Close, -1=OK, 0=Cancel,
  - >0 other command buttons
  - Retrieve user input from controls with `.Value`
How to Learn a Language

I. Use peers & gurus
II. Install tools
III. Read tutorial
IV. Find language & library reference
V. Read example programs
VI. Write example programs: I/O, types, control flow, libraries
VII. Understand error messages
VIII. Practice

Soap-box

Common VBA Mistakes

<table>
<thead>
<tr>
<th>Error</th>
<th>Message</th>
<th>Common cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>Object variable or With block variable not set</td>
<td>Missing &quot;Set&quot; before object assignment</td>
</tr>
<tr>
<td>Compile</td>
<td>Invalid character</td>
<td>Missing space before &quot;_&quot; at end of line</td>
</tr>
<tr>
<td>Compile</td>
<td>Expected: =</td>
<td>Parentheses around arguments, missing &quot;Call&quot;</td>
</tr>
</tbody>
</table>

And many more …

Suggestions for VBA Practice

- hw02 gets you points, but you may want to do more at your own leisure
- Powerpoint
  - Center shape on slide
  - Draw object model
  - Plot a polynomial
- Excel
  - Create graph, set fonts and grid preferences
  - Generate email from name+title+address sheet

Soap-box

Last Slide

- First homework due tomorrow at 4pm
- Quiz at end of next lecture at 7:30pm
- Please subscribe to class mailing list
- Today’s lecture
  - Putting the V and the A in VBA
  - Embedded scripts
  - Properties
  - Call-backs
- Next lecture
  - Associative arrays
  - Regular expressions
  - Basics of Perl

© Martin Hirzel