Using ArcObjects in ArcReader

Mark Cederholm
Unisource Energy Services

Why ArcReader?

- Free deployment for field applications
- Fully disconnected apps: no need for ArcGIS Server or Mobile
- ArcMap-quality cartography
- But: out-of-the-box functionality is limited
- Workaround: create a custom object that resides in the map and can be persisted to the PMF

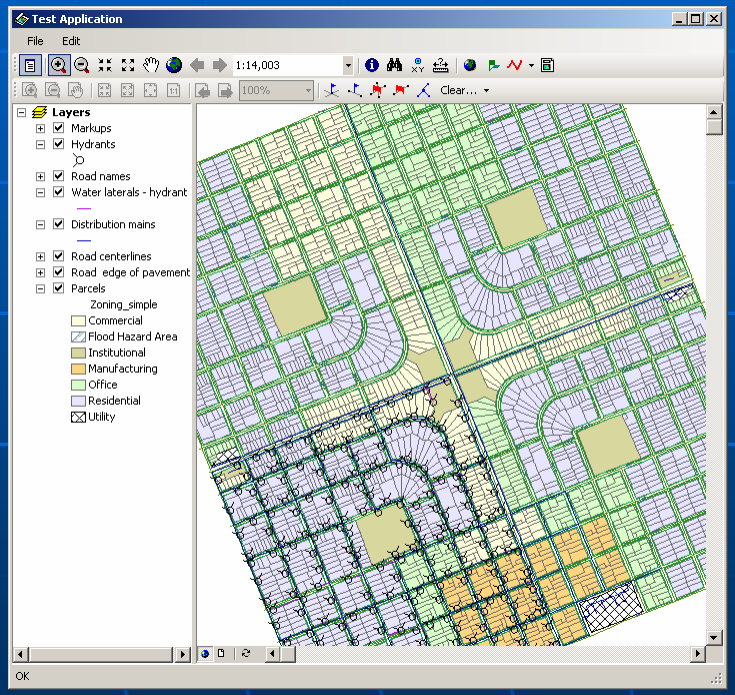
A custom ArcReader object:

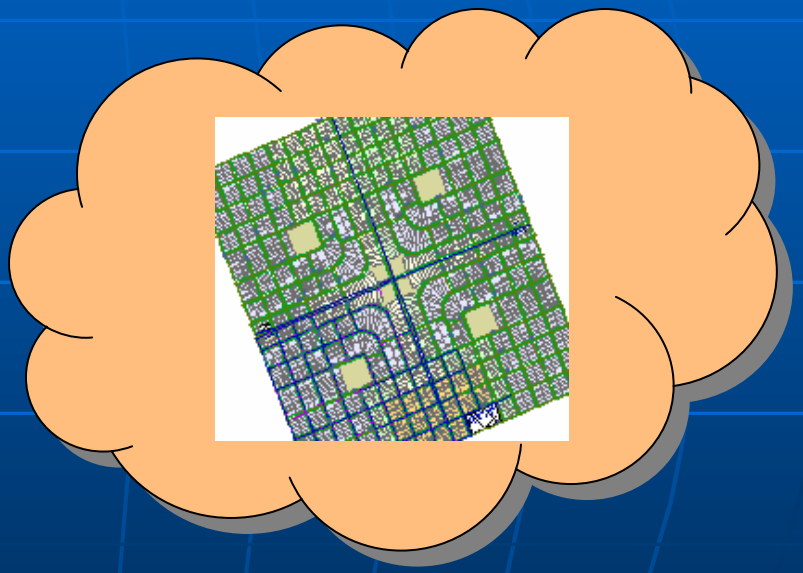
- Create a COM object that implements IPersistVariant
- In ArcMap, assign the PageLayout as a member variable which is saved and loaded in IPersistVariant
- Attach the object to some point of persistence in the map, such as a custom layer or the CustomProperty of a graphic element
- Problem: how to communicate with the custom object?

Application architecture

The ArcReader control resides in the main application

The actual map resides in a separate process called ArcReaderHost





The two processes can communicate via window messages or some other form of IPC

Why window messages?

- Simple, fast, effective
- Serializable data such as strings may be sent synchronously using SendMessage
- Integer command codes may be sent asynchronously using PostMessage
- Strategy: first send the data to be cached by the target, then send the command code for processing

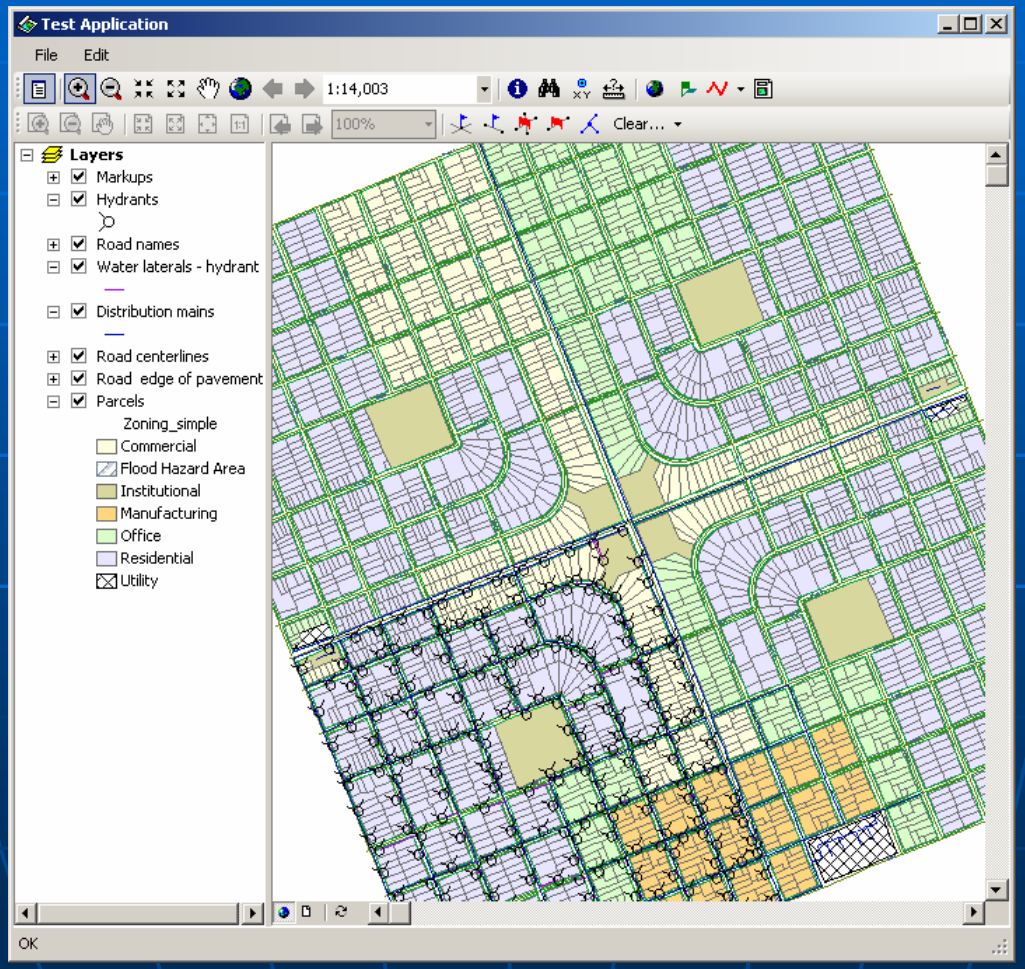
ArcObjects functionality in ArcReader

- You can manipulate just about any objects associated with the PageLayout
- Many fine-grained ArcObjects are available
- Many coarse-grained objects are not available or will not work
- Experimentation is the key to uncovering capabilities

Some things you can do

- GPS support
- Projections and transformations
- Topological operations (union, intersect, etc.)
- Network tracing through ForwardStar
- Simple, non-versioned feature edits (no edit operations)
- Custom plotting and PDF export

Demo: a sample application



Message handling, part 1

```
<StructLayout(LayoutKind.Sequential)> _
Private Structure COPYDATASTRUCT
        Public dwData As IntPtr
        Public cbData As Integer
        Public lpData As IntPtr
End Structure

Private Const WM_COPYDATA As Integer = &H4A
Private Const WM_USER As Integer = &H400
Private Const WS_POPUP As Integer = &H80000000

<DllImport("user32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> _
Private Shared Function FindWindow( _
 ByVal lpClassName As String, _
 ByVal lpWindowName As String) As IntPtr
End Function
```

Message handling, part 2

```
<DllImport("user32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> _
Private Shared Function SendMessage( _
 ByVal hWnd As IntPtr, _
 ByVal Msg As Integer, _
 ByVal wParam As Integer, _
 ByRef lParam As COPYDATASTRUCT) As Integer
End Function

<DllImport("user32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> _
Private Shared Function PostMessage( _
 ByVal hWnd As IntPtr, _
 ByVal Msg As UInteger, _
 ByVal wParam As IntPtr, _
 ByVal lParam As IntPtr) As Boolean
End Function
```

The Communicator class

```
Public Interface ICommunicator
   Sub Startup(ByVal Name As String)
   Function SetTargetName(ByVal Name As String) As Boolean
   Function SendData(ByVal DataString As String) As Boolean
   Function SendCommand(ByVal CommandCode As Integer) As Boolean
   Sub Shutdown()
End Interface

Public Interface ICommunicatorEvents
   Event DataReceived(ByVal DataString As String)
   Event CommandReceived(ByVal CommandCode As Integer)
End Interface

Public Class Communicator
   Inherits NativeWindow
   Implements ICommunicator
   Implements ICommunicatorEvents
   . . .
End Class
```

Creating, finding, and destroying a window

```
Public Sub Startup(ByVal Name As String) . . .
        Dim cp As New CreateParams
        cp.X = 0
        cp.Y = 0
        cp.Width = 0
        cp.Height = 0
        cp.Caption = Name
        cp.Style = WS_POPUP
        Me.CreateHandle(cp)
        . . .

Public Function SetTargetName(ByVal Name As String) As Boolean . . .
        Dim hWnd As IntPtr
        . . .
        hWnd = FindWindow(vbNullString, Name)
        . . .

Public Sub Shutdown() . . .
        Me.DestroyHandle()
```

Sending messages

```
Public Function SendData(ByVal DataString As String) As Boolean . . .
        . . .
        ' Serialize data
        . . .
        iResult = SendMessage(m_hTargetWnd, WM_COPYDATA, 0, cds)
        Return True
End Function

Public Function SendCommand(ByVal Code As Integer) As Boolean . . .
        . . .
        Return PostMessage(m_hTargetWnd, WM_USER, _
                New IntPtr(0), New IntPtr(Code))
End Function
```

Receiving messages

```
Protected Overrides Sub WndProc(ByRef m as Message)
        If m.Msg = WM_USER Then
                RaiseEvent CommandReceived(m.LParam.ToInt32)
        ElseIf m.Msg = WM_COPYDATA Then
                . . .
                ' Deserialize data
                . . .
                RaiseEvent DataReceived(sData)
        End If
        MyBase.WndProc(m)
End Sub
```

Serializing data

```
Dim b As New BinaryFormatter
Dim stream As New MemoryStream
Dim iDataSize, iResult As Integer
Dim bytes() As Byte
Dim pData As IntPtr
Dim cds As COPYDATASTRUCT

b.Serialize(stream, DataString)
stream.Flush()
iDataSize = stream.Length
ReDim bytes(iDataSize - 1)
stream.Seek(0, SeekOrigin.Begin)
stream.Read(bytes, 0, iDataSize)
stream.Close()
pData = Marshal.AllocCoTaskMem(iDataSize)
Marshal.Copy(bytes, 0, pData, iDataSize)
cds.lpData = pData
cds.cbData = iDataSize
cds.dwData = New IntPtr(100)
```

Deserializing data

```
Dim cds As New COPYDATASTRUCT
Dim cdsType As Type
Dim iDataSize As Integer
Dim bytes() As Byte
Dim b As New BinaryFormatter
Dim stream As MemoryStream
Dim sData As String

cdsType = cds.GetType
cds = CType(m.GetLParam(cdsType), COPYDATASTRUCT)
iDataSize = cds.cbData
ReDim bytes(iDataSize)
Marshal.Copy(cds.lpData, bytes, 0, iDataSize)
stream = New MemoryStream(bytes)
sData = b.Deserialize(stream)
```

TestHostClass

```
Public Interface ITestHost
        Function Init(ByVal pPageLayout As IPageLayout) As Boolean
        Sub SendCommand(ByVal sData As String)
End Interface

<ComClass(TestHostClass.ClassId, TestHostClass.InterfaceId, _
TestHostClass.EventsId), ProgId("TestHost.TestHostClass")> _
Public Class TestHostClass
        Implements ITestHost
        Implements IPersistVariant
        . . .
        Private m_pPageLayout As IPageLayout
        Private WithEvents m_Communicator As Communicator
        Private m_sData As String
        . . .
End Class
```

Initializing (in ArcMap)

```
Public Function Init(ByVal pPageLayout As IPageLayout) As Boolean . . .
        Dim pAV As IActiveView = pPageLayout
        Dim pMap As IMap = pAV.FocusMap
        Dim pGC As IGraphicsContainer = pPageLayout
        Dim pFrame As IFrameElement = pGC.FindFrame(pMap)
        Dim pProps As IElementProperties = pFrame
        Dim oProp As Object = pProps.CustomProperty
        If Not oProp Is Nothing Then
                Return False
        End If
        pProps.CustomProperty = Me
        m_pPageLayout = pPageLayout
        . . .
        Return True
End Function
```

Implementing IPersistVariant

```
Public Sub Load(ByVal Stream As IVariantStream) . . .
        Dim pPageLayout As IPageLayout = Nothing
        Dim bSuccess As Boolean = True
        Try
                pPageLayout = Stream.Read
                Marshal.ReleaseComObject(Stream)
        Catch ex As Exception
                bSuccess = False
        End Try
        If bSuccess Then
                m_pPageLayout = pPageLayout
        End If
End Sub

Public Sub Save(ByVal Stream As IVariantStream) . . .
        Stream.Write(m_pPageLayout)
        Marshal.ReleaseComObject(Stream)
End Sub
```

Sending and receiving commands

```
Public Sub SendCommand(ByVal sData As String) . . .
        Dim bResult As Boolean = m_Communicator.SendData(sData)
        bResult = m_Communicator.SendCommand(CommandCode)
End Sub

Private Sub m_Communicator_DataReceived(ByVal sData As String) . . .
        m_sData = sData
End Sub

Private Sub m_Communicator_CodeReceived(ByVal Code As Integer) . . .
        . . .
        ' Process command
        . . .
End Sub
```

AddTestHost.py

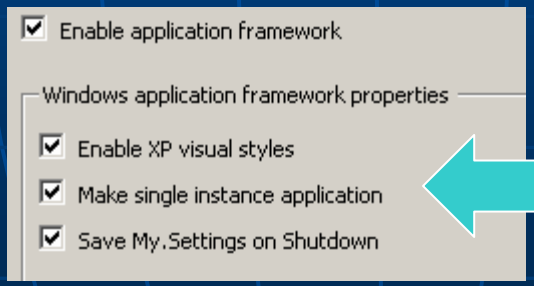
```
def AddTestHost():
    . . .
    # Get application and install TestHost
    pApp = GetApp()
    pFact = CType(pApp, esriFramework.IObjectFactory)
    pDoc = pApp.Document
    pMxDoc = CType(pDoc, esriArcMapUI.IMxDocument)
    pLayout = pMxDoc.PageLayout
    pUnk = pFact.Create(CLSID(TestHost.TestHostClass))
    pTestHost = CType(pUnk, TestHost.ITestHost)
    bResult = pTestHost.Init(pLayout)
    print bResult
```

Main Application (TestApp)

TIP: Creating the ArcReader control programmatically avoids consuming a Publisher license at design time

```
' Be sure to reference both PublisherControls and AxPublisherControls
Imports ESRI.ArcGIS.PublisherControls
Imports TestComm

Public Class TestMain
        . . .
        Private WithEvents m_Communicator As Communicator
        Private WithEvents m_ARControl As ArcReaderControl
        . . .
End Class
```



Set to single instance to avoid conflicts

Create the ArcReader control…

```
Private Sub TestMain_Load . . .

        Dim AxArcReaderControl1 As AxArcReaderControl
        Dim Init As System.ComponentModel.ISupportInitialize

        AxArcReaderControl1 = New AxArcReaderControl
        Init = AxArcReaderControl1
        Init.BeginInit()
        AxArcReaderControl1.Location = PictureBox1.Location
        AxArcReaderControl1.Size = PictureBox1.Size
        AxArcReaderControl1.Name = "AxArcReaderControl1"
        AxArcReaderControl1.Dock = DockStyle.Fill
        ToolStripContainer1.ContentPanel.Controls.Add(AxArcReaderControl1)
        Init.EndInit()
        AxArcReaderControl1.BringToFront()
        PictureBox1.Visible = False
        m_ARControl = AxArcReaderControl1.GetOcx
```

… and establish communication

```
Dim pARControl As IARControl = m_ARControl
m_Communicator = New Communicator
m_Communicator.Startup(AppName)
pARControl.LoadDocument(DefaultPMF)
. . .
Dim bResult As Boolean = m_Communicator.SetTargetName(HostName)
If Not bResult Then
        MsgBox("Could not find host communicator." . . .
        . . .
End If
SendCommand("host:target=" & AppName)
```

Tool actions are started by the main app…

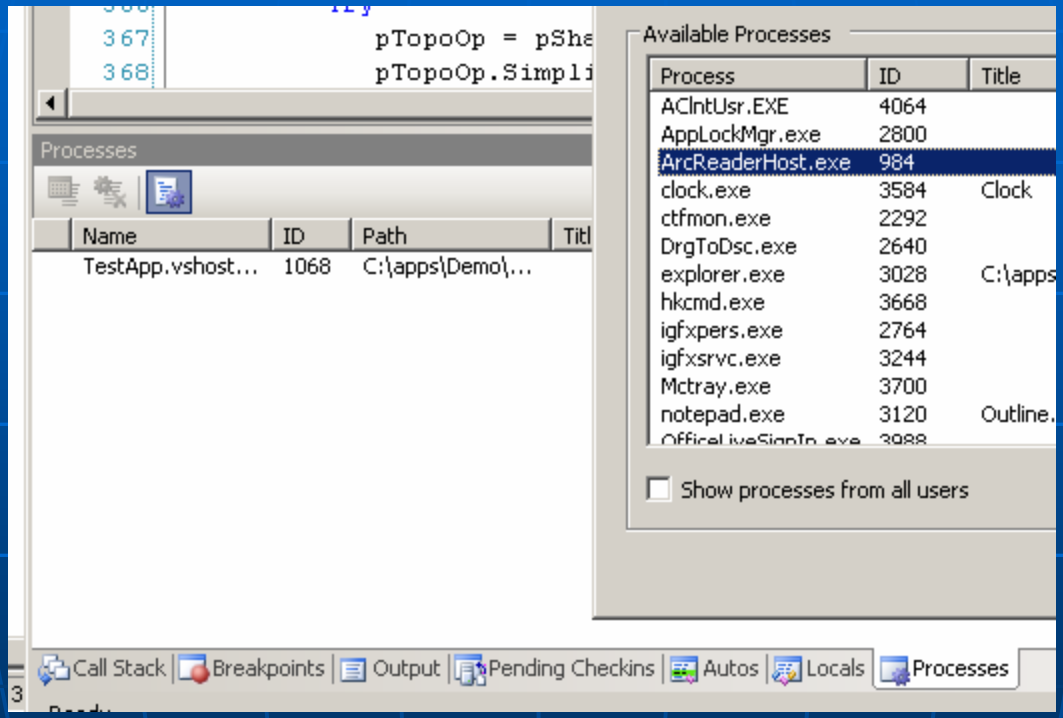
```
Private Sub m_ARControl_OnMouseDown(ByVal button As Integer . . .
        Dim sCommand As String = ""
        If m_ToolAction = ToolAction.None Then
                Exit Sub
        End If
        If button = 1 Then
                Select Case m_ToolAction
                        Case ToolAction.RedLine
                                sCommand = "markup:add_line"
                . . .
                End Select
                SendCommand(sCommand)
        End If
End Sub
```

… and finished by the host class

```
Private Function CaptureShape(ByVal opt As FeatureType) As IGeometry
        Dim pRubberBand As IRubberBand = Nothing
        Dim pShape As IGeometry = Nothing
        If opt = FeatureType.Point Then
                pRubberBand = New RubberPointClass
        ElseIf opt = FeatureType.Line Then
                pRubberBand = New RubberLineClass
        ElseIf opt = FeatureType.Poly Then
                pRubberBand = New RubberPolygonClass
        End If
        pShape = pRubberBand.TrackNew(m_pScreenDisp, Nothing)
        . . .
```

Debugging the application

- Attach the ArcReaderHost process:



- Or, debug the host class in ArcReader, using a separate testing app to send commands

Some final tips:

- While handling a synchronous window message, do as little as possible – avoid window operations
- IMap.UpdateContents will not update the TOC in the ArcReader control – adding and removing layers will lead to a disconnect
- Avoid intensive use of fine-grained ArcObjects in .NET
- For best performance, use C++ to create coarse-grained COM objects

Questions?

- Mark Cederholm
  mcederholm@uesaz.com
- This presentation and sample code may be downloaded at:

http://www.pierssen.com/arcgis/misc.htm