Programming Acrobat JavaScript Using Visual Basic

October 21, 2005
Acrobat 7.0 provides a rich set of JavaScript programming interfaces that are designed to be used from within the Acrobat environment. It also provides a mechanism (known as JSObject) that allows external clients to access the same functionality from environments such as Visual Basic.

This document gives you the information you need to get started using the extended functionality of JavaScript from a Visual Basic programming environment. It provides a set of examples to illustrate the key concepts.

What is JSObject?

In precise terms, JSObject is an interpretation layer between an OLE Automation client such as a Visual Basic application and the JavaScript functionality provided by Acrobat. From a programmer’s point of view, the end result is that programming JSObject from a Visual Basic environment is quite similar to programming in JavaScript using the Acrobat console.

Getting Started

The following steps get you set up to run the examples:

1. Install Acrobat 7.0 and Visual Basic .NET, since both are required for the examples in this document.

2. Open a new Visual Basic.NET project. That gets you started with a blank form and project workspace.

3. To access the Acrobat Automation APIs, including JSObject, you need to add a reference to Acrobat’s type library. From the UI, select Project > Add Reference, then the COM tab, and from the list of available references, click on the item labeled “Adobe Acrobat 7.0 Type Library.” Click Select. Click OK.
A Simple Example

This example describes the bare minimum required to display “Hello, Acrobat!” in Acrobat's JavaScript console.

1. Bring up the source code window for this form by selecting View > Code from the UI.

2. Select (Form1 Events) from the selection box in the upper left corner of that window.
   The selection box in the upper right shows all the functions available to the Form object.

3. Select Load from that box, which creates an empty function stub. The Form's Load function is called when the Form is first displayed, so it's a good place to add the initialization code.

This program uses some global variables for data that are required for its lifetime, and initializes them in the Form1_Load routine.
EXAMPLE 1  "Hello, Acrobat!"

Dim gApp As Acrobat.CAcroApp
Dim gPDDoc As Acrobat.CAcroPDDoc
Dim jso As Object
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    gApp = CreateObject("AcroExch.App")
    gPDDoc = CreateObject("AcroExch.PDDoc")
    If gPDDoc.Open("c:\adobe.pdf") Then
        jso = gPDDoc.GetJSObject
        jso.console.Show
        jso.console.Clear
        jso.console.println("Hello, Acrobat!")
        gApp.Show
    End If
End Sub

Note that you need a file called adobe.pdf at the root level of your C: drive.

With this code in place, the Visual Basic program attaches to Acrobat's Automation interface using the CreateObject call, then shows the main window using the App object's Show command.

You may have a few questions after studying the code fragment. For example, why is jso declared as an Object, while gApp and gPDDoc are declared as types found in the Acrobat type library? Is there a real type for JSObject?

The answer is no, JSObject does not appear in the type library, except in the context of the CAcroPDDoc.GetJSObject call. The COM interface used to export JavaScript functionality through JSObject is known as an IDispatch interface, which in Visual Basic is more commonly known simply as an "Object" type. The upshot of this is that the methods available to the programmer are not as well-defined as we would like. For example, you might be surprised to learn that if you replace the call to jso.console.clear with

    jso.ThisCanTPossiblyCompileCanIt("Yes it can!")

the compiler happily compiles the code, but fails rudely at run time. Since Visual Basic has no type information for JSObject, Visual Basic does not know if a particular call is even syntactically valid until runtime, and will compile any function call to a JSObject. For that reason, the programmer must rely on documentation to know what functionality is available through the JSObject interface. The Acrobat JavaScript Scripting Reference, which is available from http://partners.adobe.com/links/acrobat, is indispensable as you delve deeper into the mysteries of JSObject.

You may also wonder why it is necessary to open a PDDoc before creating a JSObject. Running the program shows that no document appeared onscreen, and showing the JavaScript console should be possible without a PDDoc in hand. The answer is that JSObject is designed to work closely with a particular document, since most of the available
features operate at the document level. There are some application-level features in JavaScript (and therefore in JSObject), but they are of secondary interest. In practice, a JSObject is always associated with a particular document. When working with a large number of documents, you must structure your code such that a new JSObject is acquired for each document, rather than creating a single JSObject to work on every document.

---

**Working with Annotations**

The next example is more interesting: a program that allows the user to select a PDF, add a pre-defined annotation, and save the file back to disk.

1. Create a new Visual Basic.NET project as you did in the first example; be sure to add the Adobe Acrobat 7.0 Type Library to this project as well.

2. For this example you use the Windows File > Open dialog, so select Project > Add Component from the UI. This allows you to open the Toolbox on the left side of the development environment. Select Windows Forms from the Toolbox and scroll down to and select the OpenFileDialog. Drag it to your form.

3. Use the same technique to add a Button to your form.
4. Now that a minimal user interface is set up, select **View > Code** from the main menu, and add the following source code:

**EXAMPLE 2 Working with annotations**

```vbnet
Dim gApp As Acrobat.CAcroApp
```
Private Sub Form_Load()
    gApp = CreateObject("AcroExch.App")
End Sub

Private Sub FormName.Closed(Cancel As Integer)
    If Not gApp Is Nothing Then
        gApp.Exit
    End If
    gApp = Nothing
End Sub

Private Sub Command1_Click()
    Dim pdDoc As Acrobat.CAcroPDDoc
    Dim page As Acrobat.CAcroPDPage
    Dim jso As Object
    Dim path As String
    Dim point(1) As Integer
    Dim popupRect(3) As Integer
    Dim pageRect As Object
    Dim annot As Object
    Dim props As Object
    OpenFileDialog1.ShowDialog()
    path = OpenFileDialog1.FileName

    pdDoc = CreateObject("AcroExch.PDDoc")
    If pdDoc.Open(path) Then
        jso = pdDoc.GetJSObject
        If Not jso Is Nothing Then
            ' Get size for page 0 and setup arrays
            page = pdDoc.AcquirePage(0)
            pageRect = page.GetSize
            point(0) = 0
            point(1) = pageRect.y
            popupRect(0) = 0
            popupRect(1) = pageRect.y - 100
            popupRect(2) = 200
            popupRect(3) = pageRect.y
            ' Create a new text annot
            annot = jso.AddAnnot
            props = annot.getProps
            props.Type = "Text"
            annot.setProps props
            ' Fill in a few fields
            props = annot.getProps
            props.page = 0
            props.point = point
            props.popupRect = popupRect
        End If
    End If
End Sub
props.author = "Rob McAfee"
props.noteIcon = "Comment"
props.strokeColor = jso.Color.red
props.Contents = "I added this comment from Visual Basic!"
annot.setProps props
End If
pdDoc.Close
MsgBox "Annotation added to " & path
Else
MsgBox "Failed to open " & path
End If
pdDoc = Nothing
End Sub

The code in the Form_Load and FormName.Closed routines simply initializes and shuts down the main Acrobat Automation interface. All the interesting work happens in the Command button’s click routine. The first few lines declare local variables and show the Windows Open dialog, which allows the user to select a file to annotate. At that point, the code opens the PDF’s PDDoc object, and obtains a JSObject interface to that document.

Some standard Acrobat Automation methods are used to determine the size of the first page in the document. These numbers are critical to achieving the correct layout, because the PDF coordinate system is based in the lower-left corner of the page, but the annotation will be anchored at the upper left corner of the page.

The lines following the "Create a new text annot" comment do exactly that, but this block of code bears additional explanation. First of all, addAnnot looks as if it were a method of JSObject, but the JavaScript reference shows that the method is associated with the doc object. In that case, you might expect the syntax to be jso.doc.addAnnot—however, jso is the Doc object, so jso.addAnnot is correct. All of the properties and methods in the Doc object are used in this manner.

The second item of note is the use of annot.getProps and annot.setProps. The Annot object is implemented with a separate properties object, meaning that you cannot set the properties directly. For example, you cannot do the following:

```
annot = jso.AddAnnot
annot.Type = "Text"
annot.page = 0
...
```

Instead, you must obtain the Annot’s properties object using annot.getProps, and use that object for read or write access. To save changes back to the original Annot, call annot.setProps with the modified properties object, as in the original example.

Finally, note the use of JSObject’s color property. This object defines several simple colors such as red, green, and blue. In working with colors, you may need a greater range of colors than is available through this object. Also, there is a performance hit associated with every call to JSObject. To set colors more efficiently, you can use code such as the following, which sets the annot’s strokeLineColor to red directly, bypassing the color object.

```
dim color(0 to 3) as Variant
```
Programming Acrobat JavaScript Using Visual Basic

Spell-Checking a Document

You can use this technique anywhere a color array is needed as a parameter to a JSObject routine. The example sets the colorspace to RGB, and specifies floating point values ranging from 0 to 1 for red, green, and blue. Note the use of the # character following the color values. These are required, since they tell Visual Basic that the array element should be set to a floating point value, rather than an integer. It is also important to declare the array as containing Variants, since it contains both strings and floating point values. The other color spaces ("T", "G", "CMYK") have varying requirements for array length. Refer to the `Color` object in the Acrobat JavaScript Scripting Reference for more details.

Spell-Checking a Document

Acrobat 7.0 includes a plug-in that can scan a document for spelling errors. This plug-in also provides JavaScript methods that can be accessed using a JSObject. In this example, you'll start with the source code from Example 2, and make the following changes:

1. Add a List View control to the main form. Keep the default name `ListView1` for the control.
2. Replace the code in the existing `Command1_Click` routine with the following:

```vba
Private Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim pdDoc As Acrobat.CAcroPDDoc
    Dim jso As Object
    Dim path As String
    Dim count As Integer
    Dim i As Integer, j As Integer
    Dim word As Variant
    Dim result As Variant
    Dim foundErr As Boolean

    OpenFileDialog1.ShowDialog()
    path = OpenFileDialog1.FileName
    foundErr = False
    pdDoc = CreateObject("AcroExch.PDDoc")
    If pdDoc.Open(path) Then
        jso = pdDoc.GetJSObject
        If Not jso Is Nothing Then
            count = jso.getPageNumWords(0)
```
For i = 0 To count - 1
    word = jso.getPageNthWord(0, i)
    If VarType(word) = vbString Then
        result = jso.spell.checkWord(word)
        If IsArray(result) Then
            foundErr = True
            ListView1.Items.Add (word & " is misspelled.")
            ListView1.Items.Add ("Suggestions:")
            For j = LBound(result) To UBound(result)
                ListView1.Items.Add (result(j))
            Next j
            ListView1.Items.Add ("")
        End If
    End If
Next i
jso = Nothing
pdDoc.Close

If Not foundErr Then
    ListView1.Items.Add ("No spelling errors found in " & path)
End If
Else
    MsgBox "Failed to open " & path
End If
pdDoc = Nothing
End Sub

In this example, note the use of the Spell object’s check method. According to the Acrobat JavaScript Scripting Reference, this method takes a word as input, and returns a null object if the word is found in the dictionary, or an array of suggested words if the word is not found. As always, the safest approach when storing the return value of a JSObject method call is to use a Variant. You can use the IsArray function to determine if the Variant is an array, and act accordingly. In this simple example, if the program sees an array of suggested words, it dumps them out to the ListView control.
Tips on Translating JavaScript to JSObject

Covering every method available to JSObject is beyond the scope of this document. However, the Acrobat JavaScript Scripting Reference covers the subject in detail, and much can be inferred from the reference by keeping a few basic facts in mind:

1. Most of the objects and methods in the reference are available in Visual Basic, but not all. In particular, any JavaScript object that requires the `new` operator for construction cannot be created in Visual Basic. This includes the Report object.

2. The Annots object is unusual in that it requires JSObject to set and get its properties as a separate object using the `getProps` and `setProps` methods.

3. If you are unsure what type to use to declare a variable, declare it as a Variant. This gives Visual Basic more flexibility for type conversion, and helps prevent runtime errors.

4. JSObject cannot add new properties, methods, or objects to JavaScript. Due to this limitation, the `global.setPersistent` property is not meaningful.

5. JSObject is case-insensitive. Visual Basic often capitalizes leading characters of an identifier and prevents you from changing its case. Don’t be concerned about this, since JSObject ignores case when matching the identifier to its JavaScript equivalent.

6. JSObject always returns values as Variants. This includes property gets as well as return values from method calls. An empty Variant is used when a null return value is expected. When JSObject returns an array, each element in the array is a Variant. To determine the actual data type of a Variant, use the utility functions `IsArray`, `IsNumeric`, `IsEmpty`, `IsObject`, and `VarType` from the Information module of the VBA library.

7. JSObject can process most elemental Visual Basic types for property puts and input parameters to method calls, including Variant, Array, Boolean, String, Date, Double, Long, Integer, and Byte. JSObject can accept Object parameters, but only when the Object was the result of a property get or method call to a JSObject. JSObject fails to accept values of type Error and Currency.