Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

A Quick Test Window for COM

Using any new COM object inevitably requires a little experimentation. This article shows how you can use a very simple HTA (HTML Application) to create a window that lets you interactively try out COM objects by typing in JScript.

March 01, 2001
URL:http://www.drdobbs.com/a-quick-test-window-for-com/184416298

I recently attended a lecture by Don Box where he used Visual Basic's immediate mode feature to explore various COM objects. The process seemed very cumbersome to me. Don fired up the VB IDE, created a dummy form, and put a break statement in its **FormLoad()** method. Then he ran the dummy application, and when it hit the breakpoint, he was in VB's immediate mode, which is sort of a command prompt with access to all the registered COM objects (among other things).

I saw the benefit of exploring COM objects interactively, but being a hard core C++ kind of guy, I didn't want to install the entire VB IDE and then spend way too many keystrokes just to call a method of some COM object. What I had in mind was a quick testing environment for COM objects I write. Such a testing environment would let me code a method, build the COM object (unless it's a WSC, but that's another article), immediately call it several times with different parameters, and watch the results. The solution I came up with, called WinPrompt, is based on an HTA application hosting the MS Script control. The scriplet control allows adding COM automation objects for testing and the full power of a scripting language of your choice. (I use JavaScript.) The user interface is actually a DHTML application, courtesy of the IE 5.5 WebBrowser control, which is the heart of an HTA (HTML Application).

The source for WinPrompt is in **winprmpt.hta** (Listing 1). You can also download it from this month's source code archive.

Windows and COM Objects

COM objects are the heart and soul of modern Windows applications. In the name of component-based development, everybody and his sister is doing COM these days. Just fire up the OLE/COM Object Viewer and check out your computer if you don't want to take my word for it. The Object Viewer is one of the Visual Studio tools, and you should find it in

```
<Your visual studio root>\Common\Tools\OleView.exe
```

along with some other interesting utilities. I will not delve into COM in this article; I'll just say that COM objects allow true reuse of functionality.

Raw COM objects may be instantiated and called from any language that supports pointers and virtual function tables. In order to use raw COM objects, you must know what interfaces they implement and what methods to call on each interface. COM automation objects, on the other hand, implement a special interface called **IDispatch** and carry with them type information, so dynamic discovery of their methods is possible. Many languages and tools that don't support raw COM objects do support COM automation objects. Automation objects are not as efficient as raw COM objects and are very cumbersome to use from C++. Other environments like VB, VJ++, Delphi, and scripting languages offer strong and friendly support for COM objects in general and automation objects in particular.

MS ScriptControl

The Script Control is an ActiveX control that lets developers add scripting capabilities to their applications. The application hosts the Script control, which provides several methods to add objects (automation objects), add code, and execute statements. The application developer is supposed to add his application's objects to the Script control and provide the user some mechanism (an editor window or a plain text file) to write scripts against this object model. The Script control supports any scripting language installed. WinPrompt uses JScript.

HTA

HTAs (HTML Application) are actually regular Windows applications that are built around the WebBrowser control. HTA became available with IE 5.0, so you must have IE 5.0 (or higher) installed to use it. The user interface of an HTA is DHTML, and its logic is script. At this point, the reasonable reader will ask what is the difference between an HTA and just a local HTML file. Well, it turns out that they are pretty much the same except that HTAs don't impose all the nagging security warnings that a regular browser displays if the HTML file uses ActiveX controls. Another minor difference is that HTAs have a special application tag that lets the HTA developer control several parameters, such as application icon, title, and whether or not it will appear in the taskbar. I find HTAs a great tool for rapid proptotyping and even serious utilities development, especially when using automation objects such as WSH, RegEx, and the FileSystemObject.

WinPrompt's code is really trivial, and the trick is using existing technologies in an innovative way. The result is both simple and powerful. I will first describe the user interface of WinPrompt and then the code.

The WinPrompt User Interface

The WinPrompt user interface is very minimalistic. However, I took advantage of WinPrompt to explore the wonderful world of DHTML and CSS (Cascasing Style Sheets) in IE 5. WinPrompt has a simple window with a single-line input element for entering commands and a multi-line **div** element for displaying previous commands and command results, very much like a regular command prompt. The window is resizable and the elements stretch and shrink dynamically to occupy exactly its entire area. I will explain how it's done shortly, just note that both elements have a style clause which specify "style='position:absolute'".

I wanted the command line to get the focus when WinPrompt starts, but I couldn't make it happen although I called the **focus()** method. It turned out to be your regular "gotcha"; I quote from the **focus()** method documentation: "As of Microsoft Internet Explorer 5, elements that expose the focus method must have the TABINDEX attribute set."

Another UI thing that bugs me is that IE displays a disabled scrollbar even when the entire content of the document is visible. Since the content (input + div) in WinPrompt can never exceed the window limits, I didn't want a redundant scrollbar cramping my style (sheet). That's easily done by adding "scroll='no'" to the body element.

Genesis (the onLoad event)

When a user runs WinPrompt, the **onLoad** event is fired and the **onLoad** function or event handler is executed. This function initializes the user interface and then it creates several objects, initializes the script control, and adds the objects to the script control.

Creating objects in JScript is a piece of cake. All you have to do is:

```
oObj = new ActiveXObject('ProgId');
```

Every automation object (almost) must have a **ProgId** which identifies it. **ProgId**s can be found in the registry, using COM/OLE Viewer or by looking at some MSDN samples.

The objects I created are :

WScript.Shell — The windows scripting host shell object that I use for displaying message boxes.

Scripting.FileSystemObject — Always handy for file manipulation (opening, reading, writing).

MSScriptControl.ScriptControl — The one and only script control, which is just another automation object.

InternetExplorer.Application — This is the IE browser that I want to explore.

ScriptControl Initialization

About the only thing you must do to initialize the ScriptControl is to set the language. I use the JScript language. You can initialize it to any scripting engine language that is installed on your computer, such as Perl or Python. Note that the scripting language must support the active scripting interfaces. You can't expect the script control to hack its way into any script language just like that. JScript and VBScript are prevalent on many computers since they come with newer versions of Windows and Internet Explorer. If you don't have any script engine Installed, you may download JScript and VBScript from **www.microsoft.com/scripting**.

Adding Objects to the ScriptControl

To add an object to the script control, you call a method with the surprising name of **AddObject(name, oObj)**. It takes an object name and an automation object reference as input, and once added, you may call that object's methods and get/set its properties by using its name (more on that later). A nice self-reference trick is adding the ScriptControl to itself. Suppose **oSC** is your script control. Then by calling

```
oSC.AddObject("oSC", oSC);
```

the script control may be called by script code. Taking it to the extreme, you can create a full-fledged scripting IDE with very little effort.

Adding Code to the ScriptControl

The ScriptControl also allows adding code (functions), which may be called later like normal functions. The ScriptControl supports the concept of modules and functions that are local to specific modules, but the most convenient thing is just adding functions to the global module, which is the default. To add code to the ScriptControl, I call the **AddCode()** method. This method accepts a string that must be a syntactically correct function in the selected script language. Once added, that function may be called from any other code. Adding code to the script control makes it easy to extend its capabilities. Consider, for example, a function that accepts a script filename, opens it, and adds its functions to the ScriptControl.

initUI()

initUI() bears the tremendous burden of making the window of WinPrompt look the way I want it. Much of the settings in this function could have been done in other places like attributes of HTML elements and style tags, but it took me a while to get the settings right, so it was convenient putting it in a function where I can check after each line whether the desired effect actually happened. Some properties, such as the width of **txtCode** and width/height of divOut, are dynamic properties calculated automatically whenever the window is resized. This is a new feature of IE 5, and it saves the hard-working developer from adding an **onResize()** event handler for this purpose.

loadCode()

loadCode() is a kind of internal command. You can call this function and pass it a filename. The contents of the file will be read and added to the ScriptControl. This is very convenient when you want to perform some complex initialization on an object or just perform a sequence of operations on an object. The loaded file may contain any JScript functions and refer to other objects that were added to the ScriptControl.

go()

Despite its simplicity, **go()** is where all the action is. The command line (**txtCode** input element in HTML) has an event handler for the **onkeydown** event that calls **go()** whenever the user presses the Enter key. **go()** reads the contents of the command line and calls the ScriptControl's **ExecuteStatement()** method, passing the command line as input. The ScriptControl executes the code, which may be a JScript statement, a method call of some known object or a known function. Once the execution is complete, the command line is copied to the output pane (with the WinPrompt's prompt), and the code in the command line itself is selected, so it can be modified or overwritten easily by the user. That's all there is to it.

Error Handling

I use JScript's built-in (since version 5.0) exception handling for error handling. Basically, potentially unsafe operations like calling a method with arbitrary parameters are wrapped in a **try...catch** block. The semantics of JScript's exception handling is that illegal operations throw an exception. This bumps the execution point to the **catch** clause of the **try...catch** block, passing an exception object that contains error information, and the program continues running from there. All I do in the **catch** clause is pop up a message box with an error number and description. The ScriptControl itself has some pretty good mechanisms to handle and report errors, but I left it for future development.

WinPrompt in Action

To test drive WinPrompt, just use Internet Explorer's File|Open to enter the path to **winprmpt.hta**. You should see the window shown in Figure 1. **onLoad()** creates an instance of the browser and stores it in a variable called **oBrowser**. You can cause the corresponding browser window to appear by typing:

```
oBrowser.Visible = true;
```

into the WinPrompt command prompt. The browser window should appear. You can similarly invoke any other methods that the browser object supports. For example, typing:

```
oBrowser.Navigate('www.wdj.com');
```

will cause the browser window to request and display the indicated URL. Typing:

```
oBrowser.Document.bgColor='Yellow';
```

will change the browser window's background color.

For fun, I prepared an external file called **effects.js** (Listing 2) that contains some functions you can invoke. **animate()** causes the browser to glide from right to left. **jitter()** makes the browser jitter on the screen. **stopJitter()** stops the jittering.

Future Enhancements

There are quite a few enhancements I plan to make for WinPrompt to become a truly killer tool:

- Save sessions as script files
- Use the error information and events of the ScriptControl
- View automation object properties, methods, enumerations, etc.
- Port to .NET to explore any object derived from the CLR's Object

References

HTML Applications — Power Up Your Web Applications with IE5; **msdn.microsoft.com/library/periodic/period99/htmlapplications.htm**

HOWTO: Call a Script Method from an ActiveX Script Host; **support.microsoft.com/support/kb/articles/Q222/9/66.ASP**

Say Goodbye to Macro Envy With Active Scripting; **msdn.microsoft.com/library/periodic/period97/activescripting.htm**

What is Active Scripting?; **msdn.microsoft.com/library/periodic/period99/visualprog1099.htm**

Visual Programmer: Add Scripting to Your Apps with Microsoft ScriptControl; **msdn.microsoft.com/library/periodic/perod00 /visualprog0600.htm**.

Figure 2: WinPrompt after entering some commands

Gigi Sayfan *is a sotware developer which specializes in object-oriented and component-oriented programming using C++ for the Win32 platform. He also utilizes scripting languages and various web technologies when the situation calls for it.*

Figure 1: The WinPrompt window

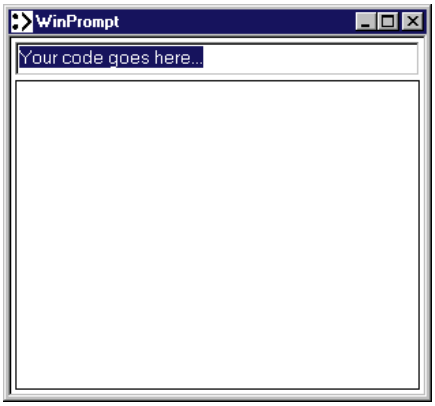
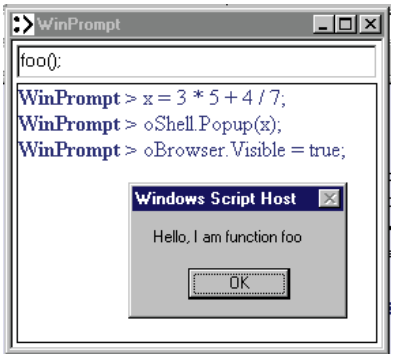


Figure 2: WinPrompt after entering some commands



Listing 1: winprmpt.hta — HTML source for WinPrompt

```
<!DOCTYPE html public "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<hta:application id="objHTA"
                 applicationname="Windows Script Prompt"
                 border="1"
                 borderstyle="NORMAL"
                 icon="WinPrmpt.ico"
                 showintaskbar="1"
                 singleinstance="1"
                 windowstate="NORMAL">

<title>WinPrompt</title>
<script language="JScript">
//-----------------------------------------------
// G L O B A L   V A R I A B L E S
//-----------------------------------------------
var oShell;
var oFS;
var oSC;
var oBrowser;

window.onload = onLoad; // attaching handler to the onLoad event

function onLoad()
{
    try
    {
        oShell   = new ActiveXObject
                   ('WScript.Shell');
        oFS      = new ActiveXObject
                   ('Scripting.FileSystemObject');
        oSC      = new ActiveXObject
                   ('MSScriptControl.ScriptControl');
        oBrowser = new ActiveXObject
                   ('InternetExplorer.Application');

        oBrowser.Left    = 600;
        oBrowser.Width   = 300;
        oBrowser.Height  = 300;

        // Initialize Script Control
        oSC.Language = 'JScript';
        // Add objects to the script control
        oSC.AddObject("oShell", oShell);
        oSC.AddObject("oFS", oFS);
```

```
        oSC.AddObject("oBrowser", oBrowser);

        // Adding the ScriptControl to itself
        oSC.AddObject("oSC", oSC);

        // Adding code to the script control
        code =  'function foo()' +
                '{ oShell.Popup("Hello, I am function foo"); }';

        oSC.AddCode(code);

        // loading external code
        loadCode('effects.js');

        initUI();
    }
    catch(e)
    {
        displayErrorMessage(e);
    }
}

function initUI()
{
    var nFrame = 3;

    resizeTo(500,500);

    with (txtCode.style)
    {
        left  = nFrame;
        top   = nFrame;

        widthExpr = 'document.body.clientWidth - ' +
                    '2*txtCode.style.pixelLeft';

        setExpression('width', widthExpr);
        height= 25;
    }

    with (divOut.style)
    {
        left  = nFrame;
        with (txtCode)
        {
            top = style.pixelTop + style.pixelHeight + nFrame;
        }

        setExpression('width',
           'document.body.clientWidth-2*divOut.style.pixelLeft');

        s1 = 'document.body.clientHeight-divOut.style.pixelTop-';
        s2 = 'divOut.style.pixelLeft';
        heightExpr = s1+s2;
        setExpression('height', heightExpr);

    }

    txtCode.select();
    txtCode.focus();

    top.moveTo(100,100);
}

function loadCode(filename)
{
    try
    {
        oFile = oFS.OpenTextFile(filename);
        code  = oFile.ReadAll();
        oSC.AddCode(code);
    }
    catch(e)
    {
        displayErrorMessage(e);
    }
}

function displayErrorMessage(e)
{
    oShell.Popup('Error (' + e.number.toString(16) +
                 '): ' + e.description);

}

function go()
{
    try
    {
        oSC.ExecuteStatement(txtCode.value);
    }
    catch(e)
    {
```

```
        displayErrorMessage(e);
    }
    output = '<b>WinPrompt ></b> '+txtCode.value+'<br/>';
    divOut.innerHTML += output;
    txtCode.select();
}
</script>
<!------------------------------>
<!--   H T M L  S T A R T    -->
<!------------------------------>
</head>
<body scroll='no'>
<input id='txtCode' type='text' tabindex='1'
        value='Your code goes here...'
        style='position:absolute; '
        onkeydown='if (event.keyCode==13) go();'/>
<div id="divOut"
    style='position:absolute; border: 1 solid black;
           color:blue; overflow:auto;' >
</div>
</body>
</html>
```

Listing 2: effects.js — Some fun JScript functions

```
var timeoutId;

function animate()
{
    if (oBrowser.Left > 100)
    {
        oBrowser.Left -= 30;
        with (oBrowser.Document.parentWindow)
        {
                setTimeout(animate, 100);
        }
    }
}

function jitter()
{

    oBrowser.Visible = true;

    with (oBrowser)
    {
        x = Left + 2 * (Math.random() - 0.5) * 10;
        y = Top  + 2 * (Math.random() - 0.5) * 10;

        Left = x;
        Top  =  y;

        // A document must be loaded into the browser,
        // or setTimeout() will throw an exception
        with (oBrowser.Document.parentWindow)
        {
            timeoutId = setTimeout(jitter, 500);
        }
    }
}

function stopJitter()
{
    oBrowser.Document.parentWindow.clearTimeout(timeoutId);
}
```