

Windows Script Components

Brian Matsik, Object Oriented Consulting Services

Introduction

A Windows Script Component (WSC) can be a very handy tool to add to the Active Server Pages (ASP) developer's toolbox, since these components can contain all common library code, usually added through an `#INCLUDE` directive within an ASP page. However, unlike an `include` file, a WSC is not loaded into memory until the `server.createObject` method is called, making the pages much more efficient. WSCs are a great link between ASP code and COM objects. This is the perfect gateway for those developers that want to move into COM programming, leveraging their current skills in HTML and ASP development with JavaScript and VBScript.



Brian Matsik, MCSD/MCDBA/MCT/CTT

Brian is the President and Senior Consultant for OOCS in Charlotte, NC. He has worked on titles such as "Professional ADO 2.5 Programming", "The VBScript Programmer's Reference", and "Professional Visual InterDev 6 Programming" (Wrox Press). He specializes in web development using Microsoft tools and technologies as well as training in Microsoft development tools.

Brian has been working with Windows applications for over 8 years and has been working with Visual Basic since version 2.0. Some of his past clients include Microsoft, First Union National Bank, and New Riders Press.

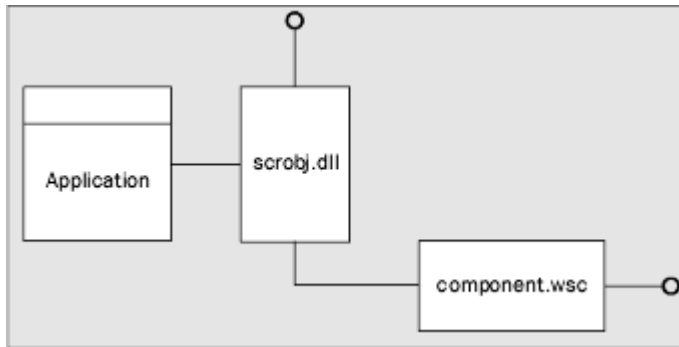
Windows Script Components

WSCs are an underutilized COM technology that really elicits the "wow" factor in any project. With just a little XML and a text editor, you can have a fully functioning COM component with only VBScript – that's correct, there is no compilation going on here. Even better, you can directly integrate with ASP and finally move all of that library code into a COM component without having to learn Visual Basic (VB) in the process.

A WSC is easy to create and maintain. Almost anyone of any level, from administrators to developers, can create and use a WSC. Here you will find out what is involved and how to work this little bit of XML and scripting magic.

How a WSC Works

A WSC is not compiled and uses only VBScript and/or JavaScript for the coding of a component. If this is a true COM component then how does the object "work"? The answer is that there is an interpreter that sits between your application and the WSC that handles all of the details for you. As you will see in a short while, this will become both a performance hit as well as an elegant workaround.



The application makes a call to the component (`myWSCComponent.wsc` in this example), and the scripting library (`scrobj.dll`) captures the call and then opens the WSC. The scripting library interprets the script contained in `myWSCComponent.wsc` to process the application request. In all actuality the script library does all of the work, and when we look at the WSC registration process you will see that the WSC does not really get registered. Rather, it is the `scrobj.dll` file – the COM component – that is registered as `myWSCComponent.wsc`.

The Structure of a WSC

A WSC is an XML document that details the essential parameters of a COM object. Below is an example of a WSC that has one property (`ConnectionString`) and one method (`ReturnRS`).

Note that this document is well-formed and valid XML:

```
<?xml version="1.0"?>
<component>

<registration
  description="ADOSample"
  progid="ADOSample.WSC"
  version="1.00"
  classid="{d4860bf1-dee3-4dd4-aff4-7446da874800}"
>
</registration>
```

```

<public>
  <property name="ConnectionString">
    <get/>
    <put/>
  </property>
  <method name="ReturnRS">
    <PARAMETER name="sSQL" />
  </method>
</public>

<implements type="Behavior" id="Behavior"/>

<script language="VBScript">

<![CDATA[

dim ConnectionString

function get_ConnectionString()
  get_ConnectionString = ConnectionString
end function

function put_ConnectionString(newValue)
  ConnectionString = newValue
end function

function ReturnRS(sSQL)
  ReturnRS = "Temporary Value"
end function

]]>

</script>

</component>

```

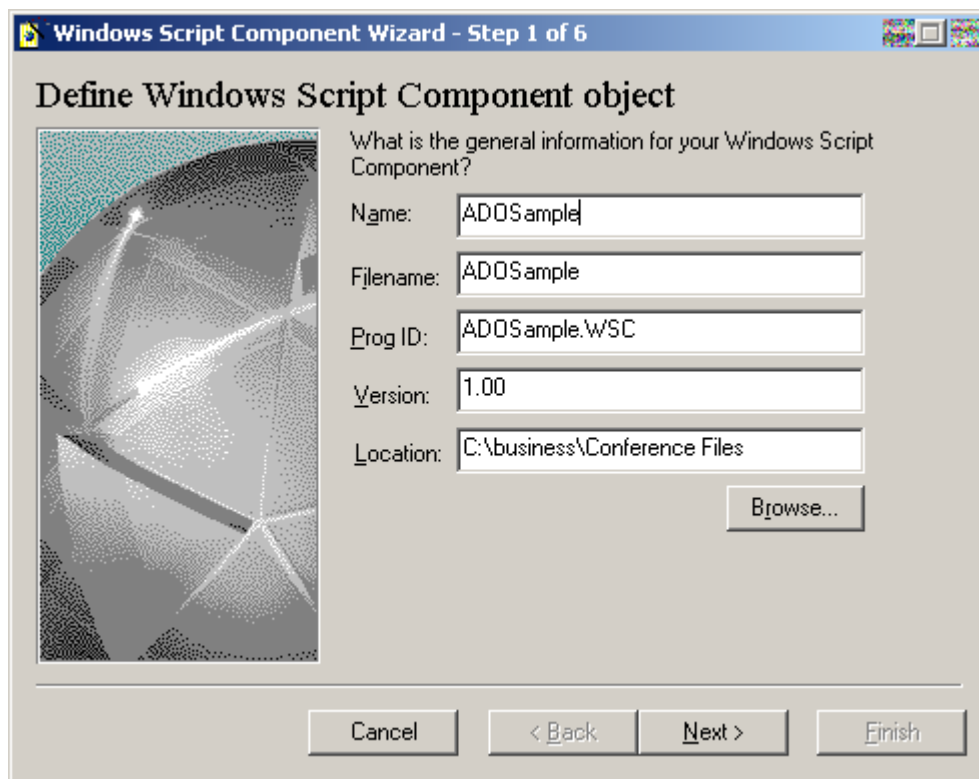
This may seem like a lot of data to remember, especially when you consider that XML is case sensitive. It is very easy to incorrectly type the skeleton of a WSC document, so there must be an easier way to create a WSC. You'll be pleased to know that there is.

Creating a WSC

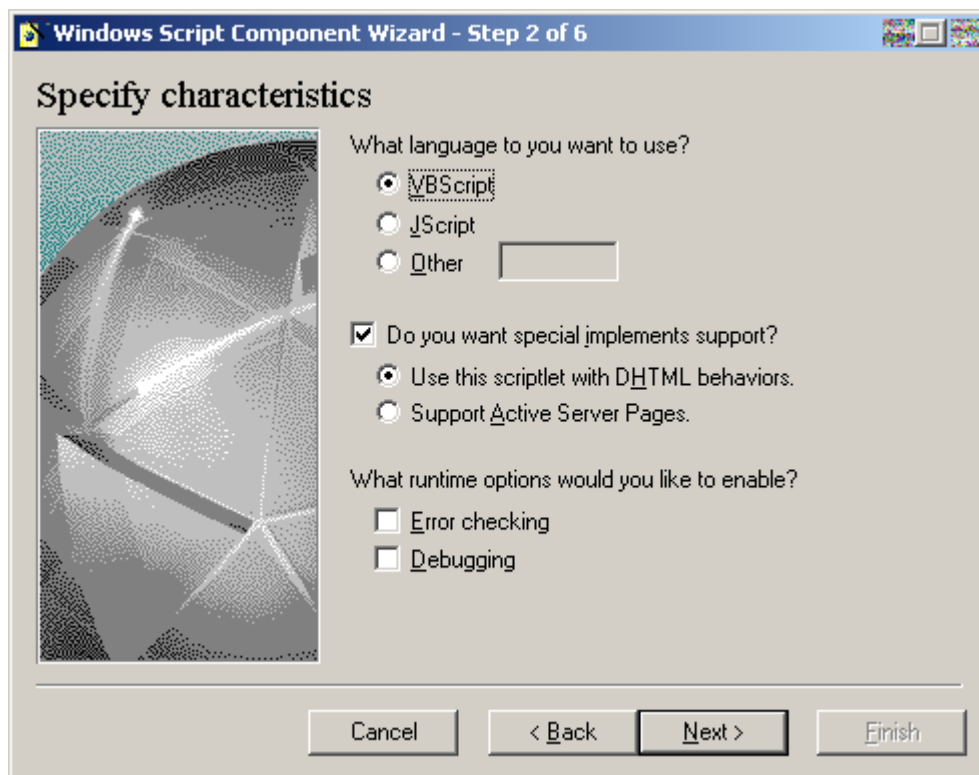
The easiest way to create a WSC is to utilize the Windows Script Component Wizard provided by Microsoft. You can download this wizard at: <http://msdn.microsoft.com/scripting>

It assists you in building the skeleton structure of a WSC.

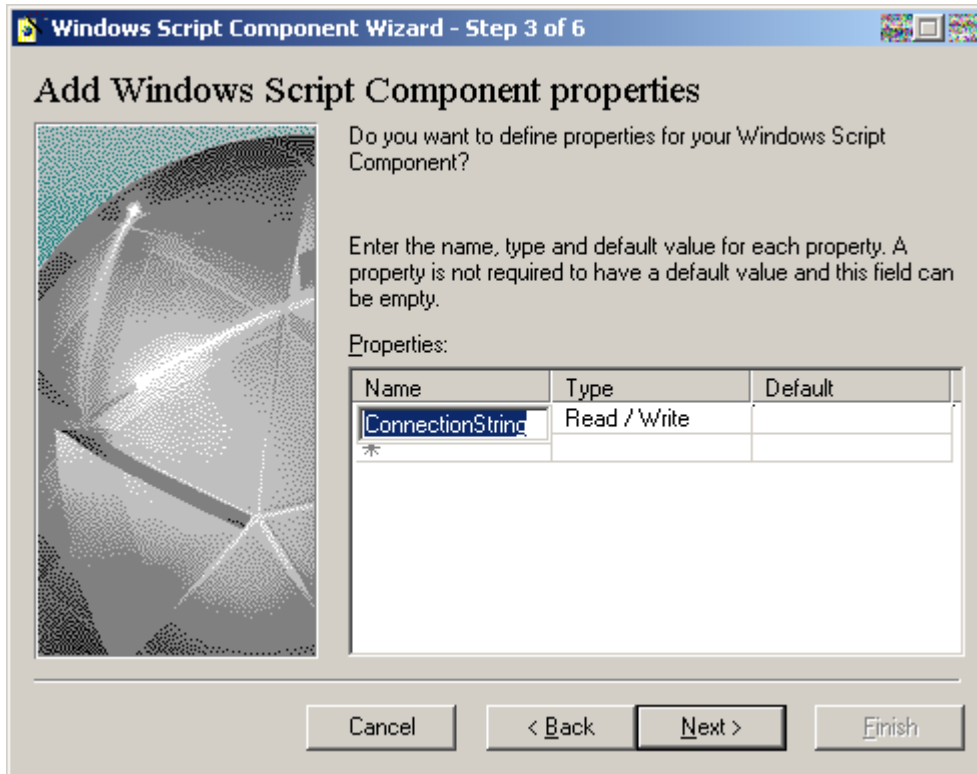
The first step of the process is to name the component and add a ProgID and filename. You can also version your WSC in the same manner that you version other COM components.



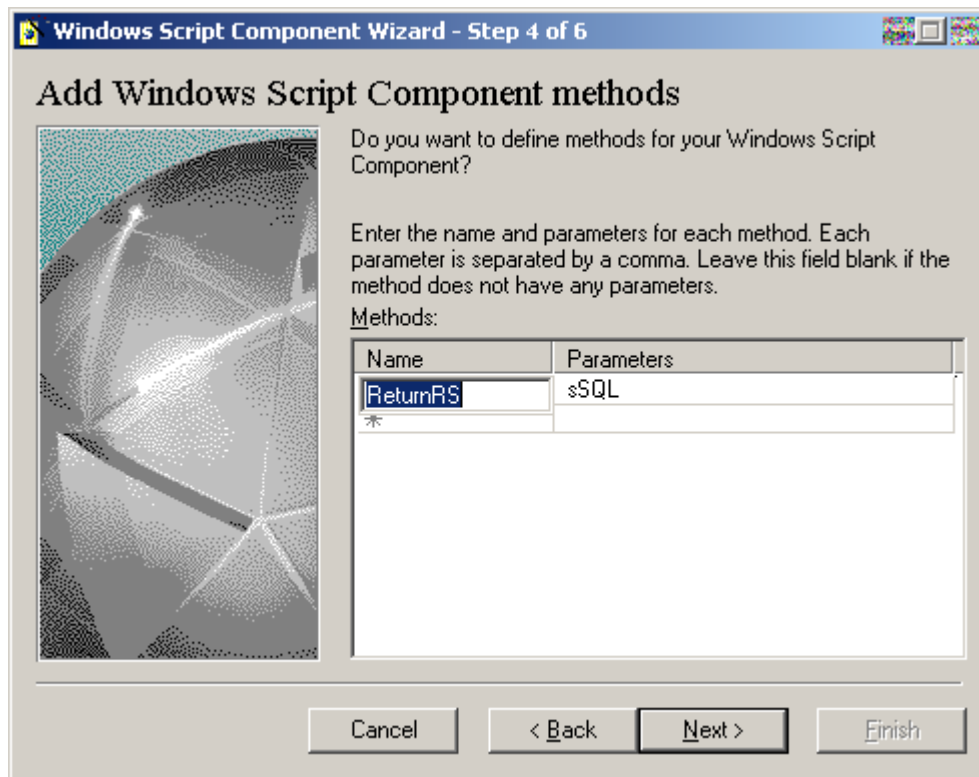
Pressing the Next button will take you to the next page of the wizard, where you define some of the characteristics of the component. First, you select the scripting language - VBScript is the default. If you plan on supporting DHTML behaviors or ASP then you select one of the built-in libraries. You can also enable error checking and debugging within the components as well. WSC errors are cryptic at best (remember this is XML!), so enabling both of these during development will help your troubleshooting process greatly.



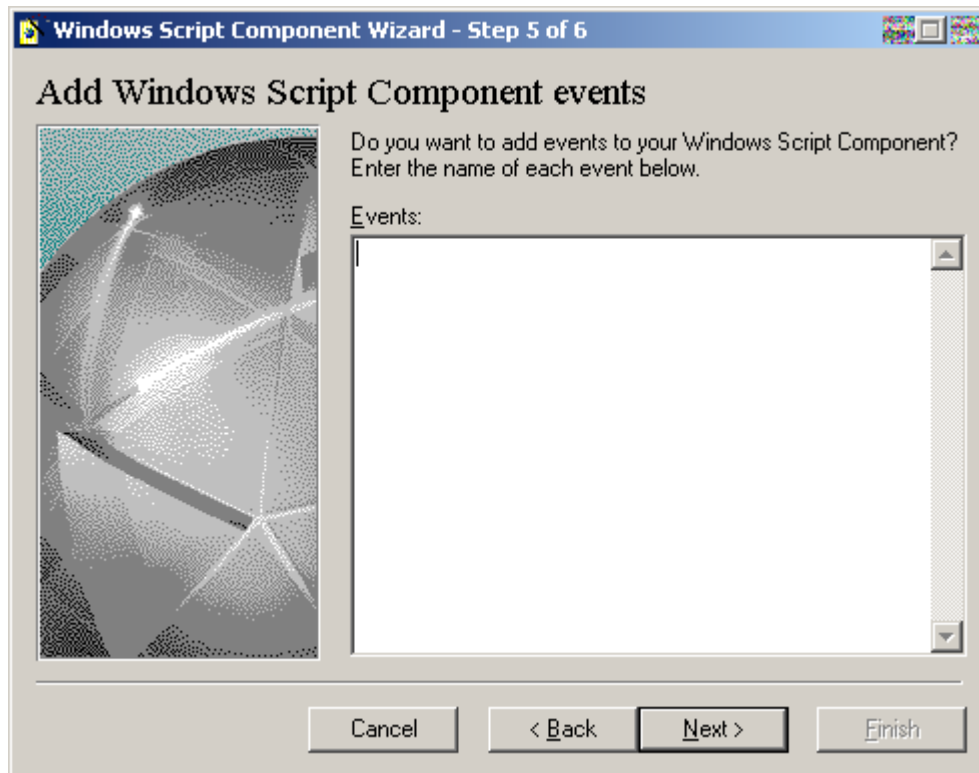
Selecting the Next button takes you to the property definition page of the WSC Wizard. In the first box you enter the name of the property, and in the dropdown you select the type of property you want to create: read, write, or read/write. The wizard will add the appropriate get (read) and put (write) methods into the component. If there is a default value for a property then you enter that into the third column. Continue adding your properties in the list.



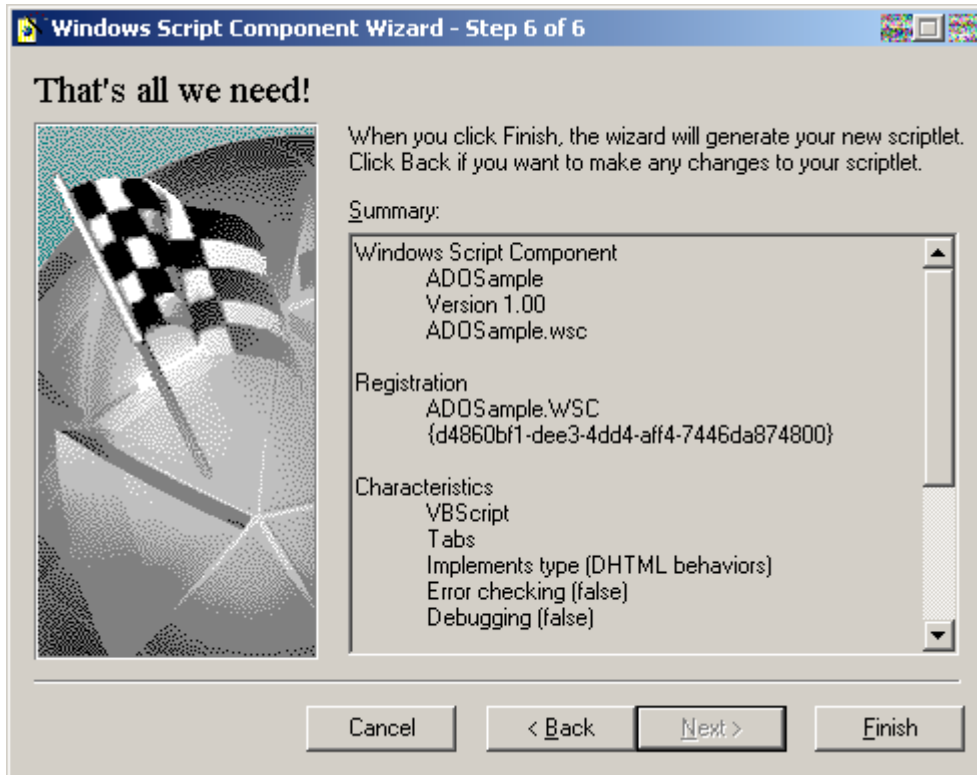
Once you have defined all of your properties, the Next button takes you to the method definition step of the wizard. This is very similar to the property definition step. You enter the name of the method in the first column and the parameters in the second column. Remember that this is VBScript, so variants only! Do not add a data type to a parameter. If you have multiple parameters then you should separate them with a comma:



Pressing the Next button takes you to the event definition page. Enter the name of the event on a separate line. The wizard will add the event declarations. When adding the code to the component then you will need to raise the event accordingly.



Here is the last step of the wizard. You can review all of the information about the component and go back to make any final changes. If you are satisfied with all of the options, press the Finish button to complete the wizard and create the component file.



At this point you will now have the same code that we looked at in the previous section.

Registration of the WSC

You can register a script component in several ways. First, you can right click on the component file and select Register from the context menu (right mouse click). This is by far the easiest way to register a component. You can also generate a type library from this menu as well.

The other method is to use the `regsvr32` file. If you use the most recent `regsvr32` file that ships with the newest version of VBScript, then you can use the following syntax:

```
regsvr32 URL/component_name.wsc
```

For example:

```
Regsvr32 c:\components\adotest.wsc
```

If you have an older version of `regsvr32` then you need to register the component through the `scrobj.dll`:

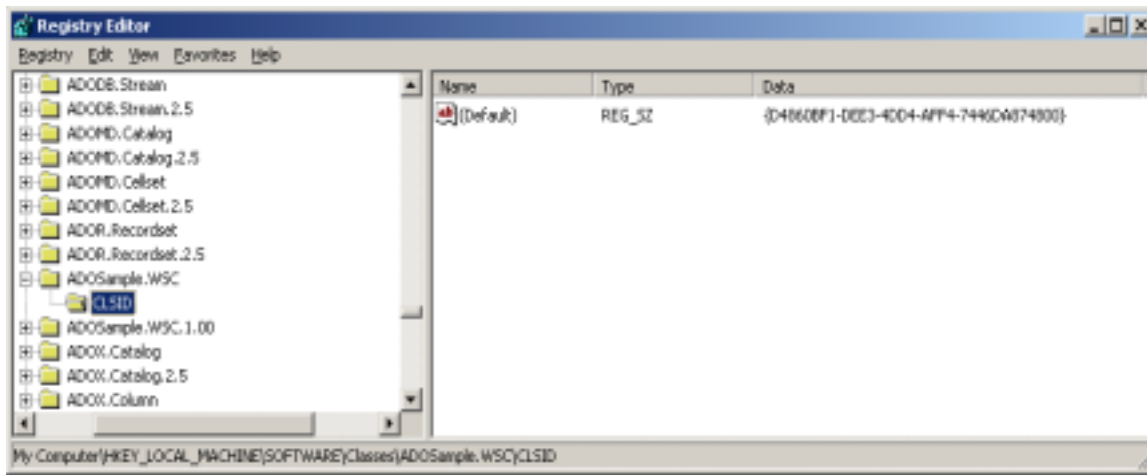
```
regsvr32 scrobj.dll /n /i:URL/component_name.ext
```

For example:

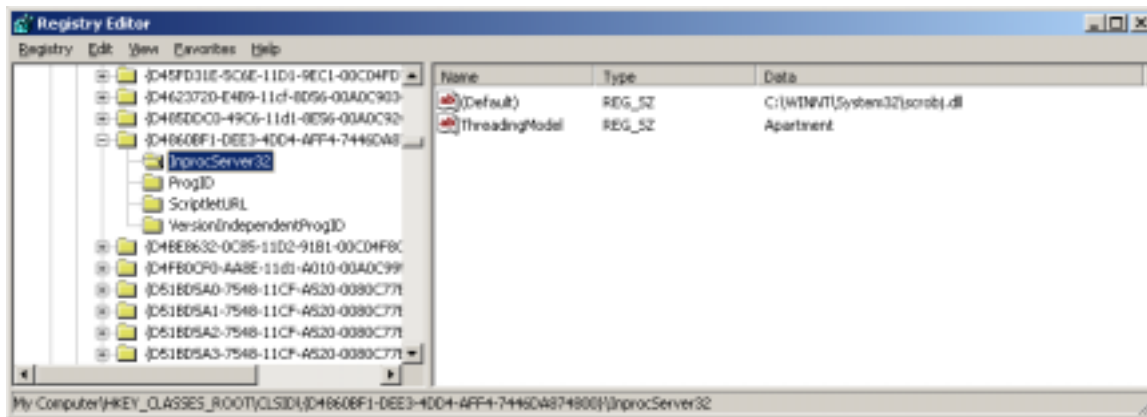
```
regsvr32 scrobj.dll /n /i:c:\components\adotest.wsc
```

At this point, your component is registered and is ready to run. Before we look at calling the WSC, let's take a look at how this is setup in the registry.

The `adosample.wsc` file is in the registry with the following CLSID:



If we look at the CLSID entry, we see the information pertaining to the `scrobj.dll` file. Note that the CLSID of the component will be different depending on what you named the component in the wizard or in the body of the component itself.



Notice that there is an entry for `ScriptletURL`. This is where the configuration information is stored that the `scrobj.dll` file needs to locate and interpret the `wsc` file. The `scrobj.dll` is the entry point for `adosample.wsc`, so there is a little trickery going on here.

Calling the WSC from Another Application

Since a WSC is a regular COM object it receives no special treatment in being created and referenced. If we create a file called `test.vbs` and enter the following code, we will be able to see how transparent the WSC implementation is:

```
dim oTest  
  
set oTest = createobject("ADOSample.wsc")  
  
msgbox oTest.ReturnRS
```

By default, this returns a message box with "Temporary Value" as the text. The reason why we get this text is because the wizard enters this as the default text of a method when it creates the prototypes.

Calling a WSC is no different from calling any other COM component. Remember that a WSC will be slower than a compiled component, but try the following with a VB COM component: change the code to return a recordset, and change the prototype of the `ReturnRS` method to accept the SQL string and the connection string. In a traditional COM component, this would break binary compatibility and would cause some problems with clients. The new component would need to be reregistered on the client or the server, but the process would not be transparent. With a WSC, this entire process is completely transparent.

If we try this, our new code is as follows:

```
function ReturnRS(sConnectionString, sSQL)

    dim cn
    dim rs

    set cn = createobject("adodb.connection")
    set rs = createobject("adodb.recordset")

    cn.open sConnectionString

    set rs = cn.execute(sSQL)

    set ReturnRS = rs

end function
```

We then call the new function with a VBScript file:

```
dim oTest
dim rs

set oTest = createobject("ADOSample.wsc")

set rs = oTest.ReturnRS("DSN=Northwind", "select * from orders")

msgbox rs(0) & ", " & rs(1)
```

We did not have to reregister the component throughout this exercise. Not only did we change our code, we changed the entire prototype of a function. Once a WSC is registered on a server then you should not have to register the component again unless you are generating a type library or versioning. It's probably best practice in COM to reregister, however.

Linking with ASP

A WSC can be enabled to integrate directly with ASP. If you select ASP support in the WSC Wizard then a section of the following is added to the file:

```
<implements type="ASP" id="ASP"/>
```

The `implements` keyword is used to link to either the DHTML behaviors library or the ASP library. With a link to the ASP library, the WSC has total access to all of the ASP objects: session, server, application, request, and response. This functionality is similar to implementing

the `onStartPage` method in VB or using the `ObjectContext` to obtain the instance of the running ASP process.

Once you implement the ASP library then you have created a much more powerful object than you may have used in the past. For instance, we can create a WSC that writes text back to the browser with very little code:

```
<?xml version="1.0"?>
<component>

<registration
  description="ASPObject"
  progid="ASPObject.WSC"
  version="1.00"
  classid="{4f24752f-65b3-4b6e-bcba-202e4e4186fa}"
>
</registration>

<public>
  <method name="WriteText">
    <PARAMETER name="sText"/>
  </method>
</public>

<implements type="ASP" id="ASP"/>

<script language="VBScript">
<![CDATA[

function WriteText(sText)
  Response.write sText
end function

]]>
</script>

</component>
```

This is a very simple example, but it is very easy to take common ASP code that would reside in `include` files and place it into a COM library. When you use the `#include` directive in ASP, the entire file is loaded with the page, even if none of the functions of the `include` file are used. With a WSC, we are using a true COM component and we have much less overhead than with the `include` files. We can set the object to nothing and recover memory, or place code into several objects and call only the necessary items. This is one of the areas where the power and flexibility of a WSC is evident.

So, why make the move to COM anyway? Well, there are several reasons why this approach may be better for you:

- Moving to components increases the scalability of your application.
- Using components in ASP pages will provide better performance than multiple `include` files.
- Script components provide a flexible and easy test bed for component development.
- COM allows better code reuse, code sharing, and application organization.
- Valuable business logic can be used across multiple applications and maintained outside of those applications. Even though the model of an application may grow in

complexity, the overall maintainability of the application will increase and lower development costs over the life of an application.

Remember that you can edit the WSC with a text editor (Interdev works just fine), so these objects are as easy (if not easier) to maintain as `include` files. One of the features of script components is the ability to modify the source of the component without the need to reregister the component. Since the `scrobj.dll` file is the real server that is being created, there is no instance of the script component in use. On web servers it is very difficult to update DLL files since many applications may have the component open and it cannot be replaced until all instances of the object are out of scope. This can be difficult to accomplish without stopping a web server or taking some related but drastic measure to keep people out of your site. The script component is interpreted through the `scrobj.dll` file, so the actual component source is not in memory, the `scrobj.dll` file is. That is what allows developers to dynamically update the COM object on a web server.

Remember that a script component is also an XML file, so you can use all of the XML parser functions and operations to dynamically build these files on a web server, but that is a discussion for another day.

Where to Go from Here

There is a great deal of information about WSCs on the MSDN site at:
<http://msdn.microsoft.com/scripting>

Also, check out the Web workshop at:
<http://msdn.microsoft.com/workshop>.

Summary

A WSC can be a very powerful tool in the ASP and the COM developer's toolbox. Anything from code libraries to prototypes can be handled with a WSC. If you are a web developer, give a WSC a try next time you are developing your site. I bet that once you have tried a WSC you will not go back.