Windows Script Components

Script Components

Microsoft® Windows® Script Components provide you with an easy way to create COM components using scripting languages such as Microsoft® Visual Basic® Scripting Edition (VBScript) and Microsoft® JScript®. Use script components as COM components in applications such as Microsoft® Internet Information Services (IIS), Microsoft® Windows® Script Host, and any other application that can support COM components. The areas discussed in this tutorial are shown in the following list.

- Script Components Overview   Learn what script components are and how to use them.
- Creating Script Components   Create a script component.
- Using Script Components   Use a script component in your applications.
- Implementing ASP Script Components   Create a script component that incorporates the functionality of Active Server Pages (ASP), allowing you to isolate and reuse ASP logic.
- Implementing DHTML Behavior Script Components   Create a script component that can be used in Microsoft® Internet Explorer 5.0 to define behaviors.

Build: Topic Version 5.6.9309.1546

Windows Script Components

Script Components Overview

Windows® Script Components are an exciting new technology that allows you to create powerful, reusable COM components with easy-to-use scripting languages such as Microsoft® Visual Basic® Scripting Edition (VBScript) and Microsoft® JScript®. The following topics provide an overview of script component technology, and explain how to create and use script components.

- Introducing Script Components   Learn what script components are, and the advantages to using them.
- How Script Components Work   Understand what script component technology consists of, and how to create script components.

Build: Topic Version 5.6.9309.1546

Windows Script Components

Introducing Windows Script Components

Windows® Script Components provide you with an easy way to create powerful, reusable COM components in script. You create script components using any scripting language that supports the Microsoft® ActiveX® Scripting interfaces. Script languages that support these interfaces include JScript, Microsoft® Visual Basic® Scripting Edition (VBScript), PERLScript, PScript, and Python.

> **Note** For more information about ActiveX Scripting interfaces, see the Microsoft Scripting Technologies Web site at www.microsoft.com.

This new script component technology supports common types of COM components, such as Automation, and is extensible with add-ons such as DHTML behaviors.

Script Components:

- Are small and efficient.
- Are easy to create, maintain, and deploy.
- Provide the ability to create COM components.
- Provide access to a broad range of system services.

Using script components, you can create COM components for a variety of tasks, such as performing middle-tier business logic, accessing and manipulating database data, adding transaction processing to applications, and adding interactive effects to a Web page using DHTML Behaviors.

See Also

How Script Components Work | Creating Script Components

Windows Script Components

How Script Components Work

Windows® Script Component technology consists of three pieces:

- The script component run-time (Scrobj.dll), which helps dispatch COM requests to your scripts. In COM terms, Scrobj.dll functions as the inproc server for script components.
- Interface handlers, which are compiled components that implement specific COM interfaces. Different interface handlers come ready to work as specific types of COM components.

  The most commonly used interface handlers, including the COM Automation interface handler, an ASP interface handler, and a handler for DHTML Behaviors, are already built into the script component run-time. Others are available as add-on components, or embedded into specific applications.

- Your script component file (an .wsc file). Script component files are XML (Extensible Markup Language) files that contain information about what type of COM component you want to create (that is, what interface handlers you want to use). Then, depending on what functionality the handler makes available, you write script in your script component to implement those interfaces.

The script component run-time serves as an entry point for the host application. The complexities of COM, including the implementation of such COM-standard interfaces as **IUnknown**, are embedded in the various interface handlers. Your script component contains only the script required to implement the functionality of the COM component.

For example, one of the most common types of COM components is an Automation component, which is a component with properties and methods that can be called from other applications. The low-level COM interfaces required to implement this functionality — such as dispatching to the correct function when a method is called — are built into an Automation interface handler. In your script component file, you define the properties, methods, and events you want to expose, and the Automation handler makes sure they are called correctly when the host application needs them.

See Also

Introducing Script Components | Creating Script Components

Build: Topic Version 5.6.9309.1546

Windows Script Components

Creating Script Components

Creating a Windows® Script Component (.wsc) file is much like creating any file containing scripts, including HTML files. The following topics provide information about the basic layout of a script component file, as well as information about creating registration information and type libraries.

- Script Component File Contents Learn about the XML (Extensible Markup Language) elements required for a script component.
- Using The Script Component Wizard Use the Script Component Wizard to create a script component file.
- Creating Registration Information Create information in the script component file that is used to register it as a COM component.
- Exposing Methods Add methods to your script component.
- Exposing Properties Add properties to your script component.
- Exposing Events Define and fire custom events in your script component.
- Creating a Script Component Type Library Create a type library for your script component so it can be used more easily in the host application.
- Referencing Other Components Include global references to objects you need in your scripts, to external type libraries that provide constants for use in your scripts, or to resources that point to strings or values not hard-coded into your scripts.
- Referencing Another Script Component in the Same Package When including multiple script components in the same package, embed the functionality of one script component inside another.
- Checking For Errors in Script Component Files Add compile-time error checking to your script component.
- Script Component Files and XML Conformance Learn about XML conformance.

Build: Topic Version 5.6.9309.1546

Windows Script Components

Script Component File Contents

Windows® Script Component files are XML (Extensible Markup Language) that are much like HTML files, but contain special elements that define the script component and its behavior. The elements used for defining script components are not HTML tags, but are XML elements specifically used for script component definitions.

A basic script component file contains these elements:

- **<component> and <package> elements** The <component> element encloses one entire script component definition. Multiple <component> elements can appear in the same .wsc file, and are contained within a master <package> element.
- **<registration> element** Includes information used to register your script component as a COM component. This element might not be required if the host application (such as Microsoft® Internet Explorer 5.0) does not directly use the Windows registry when creating an instance of the script component.
- **<public> element** Encloses definitions for properties, methods, and events that your script component exposes. The definitions point to variables or functions defined in a separate <script> block.
- **<implements> element** Specifies the COM interface handler for the script component, which determines what type of COM component the script component will be. For example, by specifying <implements type=ASP>, you implement the ASP interface handler and therefore get access to the ASP object model in your script component.

  The <public> element is used to specify that a script component implements the COM Automation interface handler. Therefore, you don't need to create an <implements> element for the Automation handler.

  > **Note** The script component run-time includes interface handlers for Automation (exposed using the <public> element), for ASP, and for Internet Explorer 5.0 DHTML Behaviors. Other interface handlers are available as external DLLs. For more information about additional interface handlers and script components, see the Microsoft Scripting Technologies Web site.

- **<script> element** Contains the script used to implement the logic of your script component, depending on what type of COM component you are creating. For example, if you are creating a COM Automation component, you declare properties, methods, and events in a <public> element, and then write the script to define them in one or more <script> elements.

- **<object> element** Contains information about an object that you use in your script, such as another COM component.
- **<resource> elements** Contain values that should not be hard-coded into script component code. Resource elements can include information that might change between versions, strings that might be translated, and other values.
- **<reference> element** References a type library you want to use in script.
- **<comment> elements** Contain text that is ignored when the script component is parsed and executed.

> **Note** If you are concerned that the .wsc files you create contain XML that conforms to XML standards, you can specify that the script component's XML parser check the XML syntax. For example, this is useful if you think you might someday use an XML editor to work with your files. Otherwise, however, it is not usually a concern. For more details, see Script Component Files and XML Conformance.

Skeleton Script Component File

The following example illustrates how to construct a script component file.

```
<?XML version="1.0"?>
<package>
<?component error="true" debug="true"?>

   <comment>
      This skeleton shows how script component elements are
      assembled into a .wsc file.
   </comment>

<component id="MyScriptlet">
   <registration
      progid="progID"
      description="description"
      version="version"
      clsid="{00000000-0000-0000-000000000000}"/>

   <reference object="progID">

   <public>
      <property name="propertyname"/>
      <method name="methodname"/>
      <event name="eventname"/>
   </public>

   <implements type=COMhandlerName id=internalName>
```

```
      (interface-specific definitions here)
   </implements>

   <script language="VBScript">
      <![CDATA[
      dim propertyname
      Function methodname()
      ' Script here.
      End Function
      ]]>
   </script>

   <script language="JScript">
      <![CDATA[
      function get_propertyname()
      { // Script here.
      }
      function put_propertyname(newValue)
      { // Script here.
         fireEvent(eventname)
      }
      ]]>
   </script>

   <object id="objID" classid="clsid:00000000-0000-0000-000000000000">
   <resource ID="resourceID1">string or number here</resource>
   <resource ID="resourceID2">string or number here</resource>
</component>
</package>
```

Note In XML, you can specify elements without content (attributes only), such as the <property> and <method> elements in the previous example, by closing the element with />.

Note that:

- The <?XML ?> declaration at the top indicates that this is an XML file and that it conforms to XML protocol. This declaration is optional; if you leave it out, you can use slightly looser syntax when creating the script component's elements. For details, see Script Component Files and XML Conformance.
- The <package> element is optional in this example, because the file contains only one <component> element.
- The <?component?> processing instruction includes attributes for specifying the error checking options. For details, see Checking For Errors in Script Component Files. If you do not include this element, the default options are all false.

- A <comment> element can appear anywhere in the script component.
- A <registration> element is not required in all cases. For example, a script component that implements the DHTML Behaviors interface handler in Internet Explorer 5.0 does not need to be registered, because Internet Explorer directly instantiates the behaviors as they are detected on the page. For details about registration requirements, see the documentation for the interface handler you are implementing, and note also which host the script component will be used in.
- A <reference> element allows you to include a type library in the script component so you can use the library's constants in your script.
- The <public> element encloses <property>, <method>, and <event> elements. The script that defines these appears later in the script component file.
- The <implements> element is used to make available non-default COM interfaces. For example, you can expose the ASP interface with an <implements> type such as the following:

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

  The exact elements that appear inside an <implements> element depend on what interface you are implementing.

  > **Note** The <implements> element is shown here with an id attribute. However, this attribute is optional, except in cases where you must disambiguate objects or variables. For details, see the <implements> element.

- In this skeleton, there are two script elements, one for VBScript and one for JScript. If you are using only one scripting language, you do not need more than one <script> element. Because the <?XML ?> declaration appears at the top of the file, a CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.
- The <object> element creates a reference to an object you want to use in script within the script component.

After creating the skeleton, fill in the elements to define the script component's functionality, depending on which interface handler you are implementing.

See Also

Checking For Errors in Script Component Files | Creating a Script Component Type Library | Creating Registration Information | Exposing Events | Exposing Methods | Exposing Properties | Implementing ASP Script Components | Implementing DHTML Behavior Script Components | Script Component Files and XML Conformance | Using the Script Component Wizard

Build: Topic Version 5.6.9309.1546

Windows Script Components

Using the Script Component Wizard

The Windows® Script Component package includes a Script Component Wizard that automates the process of creating a script component (.wsc) file. The wizard performs the following functions:

- Creates the .wsc file and adds basic XML elements.
- Prompts for and creates registration information.
- Prompts for the type of interface handler you want the script component to use.
- Prompts for properties you want to expose and creates the necessary elements in the scripting language you specify.
- Prompts for methods you want to expose and creates the necessary elements in the scripting language you specify.
- Prompts for events that the script component can fire and creates the skeleton event elements.

> **Note** You can download and install the Script Component Wizard from the Microsoft Scripting Technologies Web site on www.microsoft.com.

The wizard creates a skeleton script component with information you provide, and writes it out as an .wsc file. Edit the .wsc file and add your script to create the script component's functionality.

> **Note** The wizard creates a script component file containing the <?XML ?> declaration at the top, meaning that the contents of the file must be XML-conformant. For more details, see Script Component Files and XML Conformance.

See Also

Script Component File Contents | Creating Registration Information | Creating Script Components | Script Component Files and XML Conformance

Build: Topic Version 5.6.9309.1546

Windows Script Components

Creating Registration Information

If the host application requires it, you can register a Windows® Script Component as a COM component using a program such as Regsvr32.exe. Registration places information about the COM component in a public location (in Windows, registration information is stored in the Windows registry). By reading the registration information, applications can find and load the COM component.

> **Note** Registration is not required in every case. For example, a script component that implements the DHTML Behaviors interface handler in Internet Explorer 5.0 does not need to be registered, because Internet Explorer registers the behaviors as they are detected on the page. For more details about using Behaviors, see the "Using DHTML Behaviors" topic on the Microsoft Site Builder Network (SBN) Web site. If the host application supports monikers, you can also create an instance of a script component without registering it.

When you specify registration, the most important information is:

- A program ID (prog ID), which is the name used in the host application when creating an instance of the script component. For example, if your script component's program ID is Component.MyComponent, you can create an instance of it in Microsoft® Visual Basic® using a statement such as the following:

  ```
  Set Component = CreateObject("Component.MyComponent")
  ```

- A class ID, which is a GUID (globally unique ID) that uniquely identifies your script component. You can generate a GUID with a program such as Uuidgen.exe.

  > **Note** If you create a script component using the Script Component Wizard, the wizard automatically creates a program ID and class ID for you. For details, see Using The Script Component Wizard.

Registration information also includes a description and a version number.

The registration program can create a class ID for your script component when it is registered. However, it is highly recommended that you provide a class ID for the script component yourself, to ensure that your script component has the same class ID on all computers on which it is registered. Allowing the registration program to create a class ID can also cause problems if you use the script component with development tools that store class IDs. If the registration creates a new class ID each time, it will not match the ID stored by the application.

To create registration information for the script component

- Create an <registration> element that includes at least a program ID, and optionally, a class ID, description, and version, as shown in the following example:

  ```
  <registration
  description="My Test Component"
  progid="Component.TestScript"
  version="1"
  classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}"/>
  ```

  > **Note** The registration attributes can appear in any order in the <registration> element.

Running Script During Registration

The script component <registration> element also allows you to include script that will be executed when the script component is registered and unregistered. For example, you can post a message that the script component has been registered.

To run script during registration and unregistration

- In the <registration> element, include an <script> element. To run script during registration, write a **register( )** function. To run script when the script component has been unregistered, include an **unregister( )** function.

  The following example shows how to post messages when the script component is registered or unregistered.

  > **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

  ```
  <registration
     description="My Test Component"
     progid="Component.TestScript"
     version="1"
     classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}">

     <script language="VBScript">
     <![CDATA[
        Function register()
           MsgBox "Component 'My Test Component' registered."
        End Function
        Function unregister()
           MsgBox "Component 'My Test Component' unregistered."
  ```

```
      End Function
   ]]>
   </script>
</registration>
```

Registering for Remote Access

If you are deploying your script component in a distributed environment, you can specify that the script component can be instantiated remotely. For example, you might be creating a script component designed to run on a server, but which will be called from code on a client. This scenario is possible if the client and server machines are properly configured with DCOM, which provides the mechanism for passing object pointers to the client from the server.

To register a script component for remote access

- Include the remotable attribute in the <registration> element, as shown in the following example:

```
<registration
   description="My Test Component"
   progid="Component.TestScript"
   version="1"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}"
   remotable=true/>
```

For more details about creating instances of a script component remotely, see Using a Script Component in an Application.

See Also

Script Component File Contents | Using The Script Component Wizard | Creating a Script Component Type Library | Checking For Errors in Script Component Files | Script Component Files and XML Conformance

Windows Script Components

Exposing Methods

Methods are implemented as functions or subroutines in your Windows® Script Component file.

To expose a method

1. Create a <public> element as a child of the <component> element.
2. In the <public> element, include a <method> element. The method element can optionally include one or more <parameter> elements to define the method's parameters.
3. Write a procedure in any scripting language to implement the function. Place the procedure in a <script> element outside the <implements> element but within the <component> element. Be sure the function name matches the *functionName*, or if you did not specify a *functionName*, the *methodName* name you specified in the <method> element.

   For example, the following example shows a fragment from a script component file with two methods, factorial and getRandomNumber.

   > **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <method name="factorial"/>
   <method name="random" internalName="getRandomNumber">
      <parameter name="upperBound"/>
      <parameter name="seed"/>
   </method>
</public>

<script language="VBScript">
Function factorial(n)
   <![CDATA[
   If isNumeric(n) Then
      If n <= 1 Then
         factorial = 1
      Else
         factorial = n*factorial(n-1)
      End If
   Else
      factorial = -2   ' Error code.
   End If
```

```
      End Function

      Function getRandomNumber(upperBound, seed)
         getRandomNumber = Cint(upperBound * Rnd(seed) + 1)
      End Function
      ]]>
      </script>
```

You can specify a default method for a script component so the host application can invoke the method without explicitly calling it. For example, if you have exposed a method called factorial and marked it as the default, you can call it in the followoing ways in Visual Basic:

```
Set component = CreateObject("component.MyComponent")
n = component.factorial(4)   ' Calls factorial method explicitly.
n = component(4)   ' Calls factorial method as default.
```

To specify a default method, include an attribute assigning a special dispatch identifier (a *dispid*) to the method. For more information about dispids, see Exposing Events.

To specify a default method

- In the <method> element, include the attribute dispid="0", as shown in the following example:

```
<public>
   <method name="factorial" dispid="0"/>
</public>
```

> **Note** This technique can be used to assign either a default method or a default property, but not both. There can be only one element in the script component with the dispid of zero.

See Also

Exposing Events | Exposing Properties | Script Component File Contents

---

Windows Script Components

Exposing Properties

You can expose properties in two ways:

- **As simple values** The property is simply a global variable that can be read and written by the script component's user. Because the value is stored in a global value, you can manipulate it in your Windows® Script Component file's scripts.
- **As functions** The property is defined using a function. This allows you to calculate a property value, and to control whether a property is read-only, read-write, or write-only.

You can also mark a property as the default value for a script component.

To expose a property as a simple value

1. Create a <public> element as a child of the <component> element.
2. In the <public> element, include a <property> element that specifies the variable used to store property value.
3. To initialize the value of a simple property, create a global variable in the <script> element with a name matching *propertyName* (or *propertyVariable*, if you specified that) and assign it a value.
4. The following script component fragment illustrates two properties (name and tag) exposed as simple values. The properties are initialized using global variables in the <script> element.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <property name="name"/>
   <property name="tag" internalName="tagVar"/>
</public>

<script language="VBScript">
   <![CDATA[
   Dim name
   name = "script component"   ' Initializes the value of name property.
   Dim tagVar
   tagVar = "10"   ' Initializes the value of tag property.
   ]]>
</script>
```

Exposing a property using functions is similar to exposing a method.

To expose a property using functions

1. In the script component file's <public> element, include a <property> element that contains a <get> element to define the read function and a <put> element to define the write function. If you do not include the <put> element, the property will be read-only. If you do not include the <get> element, the property will be write-only.
2. Write procedures in a <script> element outside the <public> element to implement the functions. The function for setting the property's value — the put function — must accept a single parameter, the value to set the property to.

   The names of the procedures must match the internal names you specified in the <property> element. If you did not specify an internalName attribute, the names of the functions must be the name of the function with the get_ prefix for a read function, and with a put_ prefix for the write function.

3. For example, the following script component fragment is an example of a script component file that exposes three properties: sname, dateOfBirth, and age. The dateOfBirth property is defined by functions so it can include error checking. The age property is calculated, and is therefore defined as read-only.

   > **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <property name="sname"/>
   <property name="age">
      <get internalName="readAge"/>
   </property>
   <property name="dateOfBirth">
      <get internalName="readDOB"/>
      <put internalName="writeDOB"/>
   </property>
</public>

<script language="VBScript">
<![CDATA[
Dim sname   ' Read-write sname property (no functions).
Dim gDOB   ' Global variable used to store date of birth.

Function readDOB()
   ' Gets value of dateOfBirth property.
   readDOB = gDOB
```

```
     End Function

     Function writeDOB(newDOB)
        ' Sets value of dateOfBirth property.
        If isDate(gDOB) Then
           'Error checking
           gDOB = newDOB
        End If
     End Function

     Function readAge()
        ' Calculates read-only age property.
        If isDate(gDOB) Then
           dobM = DatePart("m", gDOB)
           dobD = DatePart("d", gDOB)
           dobY = DatePart("yyyy", gDOB)
           todayY = DatePart("yyyy", Date)
           age = todayY - dobY

           ' Adjust if birthday has not yet occurred this year.
           bday = DateValue(dobM & "/" & dobD & "/" & todayY)
           If DateDiff("d", bday, DateValue(Date)) < 0 Then
              age = age - 1
           End If
           readAge = age
        End If
     End Function
     ]]>
     </script>
```

You can specify a default property for a script component so the host application can get or set the property's value without explicitly naming the property. For example, if you have exposed a property called sname and marked it as the default, you can work with it in either of these ways in Visual Basic:

```
Set component = CreateObject("Component.MyComponent")
n = component.sname   ' Gets property explicitly.
n = component   ' Gets value of sname as default.
```

To specify a default property, include an attribute assigning a special dispatch identifier (a *dispid*) to the method. For more information about dispids, see Exposing Events.

To specify a default property

- In the <property> element, include the attribute dispid="0", as in the following example:

```
<property name="sname" dispid="0"/>
```

> **Note** This technique can be used to assign either a default property or a default method, but not both. There can be only one element in the script component with the dispid of zero.

See Also

Exposing Events | Exposing Methods | Script Component File Contents | Script Component Files and XML Conformance

---

Windows Script Components

Exposing Events

To add event capability to a Windows® Script Component:

- Declare each event that you want to be able to fire.
- Call a function to fire the event as required in the script component.

Some host environments also require that you generate a type library which they use to bind to events. For details, see Creating a Script Component Type Library.

> **Note** The Behavior handler exposes events in a slightly different way. For details, see Exposing Custom Events in Behavior Script Components.

Declaring Events

Each event you want to be able to fire must be individually declared.

To declare an event

1. Create a <public> element as a child of the <component> element.
2. In the <public> element, include an <event> element for each event you want to declare.
3. For example, the following script component fragment shows how to expose two events:

```
<public>
   <property name="sname"/>
   <method name="factorial"/>
   <event name="namechanged"/>
   <event name="querydone"/>
</public>
```

Specifying Dispatch Identifiers

COM programming provides event notification via dispatch identifiers (each referred to as a *dispid*), which are integer values that identify a component's events. The dispid is compiled into the component's type library and then used by the host application to bind to events.

The process of creating a type library for script components automatically generates dispids for your script component's events. However, if you prefer, you can specify your own dispids. Doing so allows you to:

- Guarantee that events in your script component will always have the same dispid. If the type library generator assigns dispids, they can change each time the library is generated.
- Map events in your script component to dispids with specific numbers. For example, if you want to fire a standard COM event such as an error notification, you can map your event to the values used by convention in COM.

To specify a dispid for an event, include the dispid attribute in the <event> element, as in the following example:

```
<public>
   <event name="namechanged" dispid="22">
</public>
```

Dispids must be unique within the script component. You can specify a negative value for a dispid to map to conventional events, but must use only specified ranges, such as -999 to -500 for controls. For details about reserved dispid ranges, refer to documentation for DISPID in the MSDN library.

Note The dispid number zero is used to identify a default method or property. For more details, see Exposing Methods and Exposing Properties.

Firing an Event

You can fire an event by calling the fireEvent method, specifying the name of the event to fire. You cannot fire events that you did not expose in the <implements> element. You can fire an event in any script in your script component file. For example, the following illustrates how you can fire an event when a property value changes.

```
<script language="VBScript">
<![CDATA[
Sub put_lowercaseName(newLCName)
   name = newLCName
   fireEvent("namechanged")
End Sub
]]>
</script>
```

See Also

Exposing Custom Events in Behavior Script Components | Exposing Methods | Exposing Properties | Handling Script Component Events in the Host Application | Script Component File Contents

Windows Script Components

Creating a Script Component Type Library

You can generate a type library for your Windows® Script Component containing information about its interfaces and members. In some host applications (such as Visual Basic), type libraries are necessary to enable events for the script component, while in other host applications, type libraries are optional. However, even if a type library is not required, generating one can make working with a script component easier and less error-prone in the host application.

For example, if you are using Visual Basic as your host application, use the **References** dialog box to select a script component's type library. This allows you to bind events to the script component and make them visible in Visual Basic. In addition, when you are writing Visual Basic code that references the script component, Visual Basic uses the type library information in statement completion and in the Object Browser so you can easily see and use properties, methods, and events that the script component exposes.

> **Note** For information about using a type library in the host application, refer to the application's documentation.

To create a script component type library

- In Windows Explorer, right-click the script component .wsc file, and then choose **Generate Type Library**.

  A .tlb file is generated for the script component with the same name as the script component file, written to the folder where the .wsc file is, and registered in the Windows registry.

For more precise control over generating type libraries, you can generate the type library dynamically from script inside the script component file, or you can use a command-line interface.

Generating Type Libraries Dynamically

The script component run-time includes an Automation interface implemented by the Component.GenerateTypeLib object. You can use this object in script to generate a type library from within the script component file. This is particularly useful for creating a type library automatically when the script component is registered.

To create a script component type library dynamically

1. In script inside the script component file, create an instance of the Component.GenerateTypeLib object by using syntax such as the following:

   ```
   set oTL = CreateObject("Scriptlet.GenerateTypeLib")
   ```

2. Set the following properties of the Component.GenerateTypeLib object:

Property/Method	Description
AddURL	(Method) Adds the URL of the script component file from which to generate the type

	library. You can call this method property multiple times to include multiple script components in the type library.
Path	(Property) The path and file where the type library will be written. The default path is the current directory and the default name is the name of the script component file with a .tlb extension. If the object is unable to create the specified library, it defaults to creating a type library called component.tlb.
Doc	(Property) A string containing any information that is stored in the registry with the type library information.
GUID	(Property) A GUID for the type library. (This is not the GUID for the script component.) If you do not provide one, the GenerateTypeLib object will create one, but then the type library will have a different GUID on each machine. The GUID must be exactly in this format, where *x* represents a hex value: `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}`
Name	(Property) The internal name for the type library. This name is displayed in some applications, such as the Visual Basic Object Browser.
MajorVersion	(Property) An integer value you assign.
MinorVersion	(Property) An integer value you assign.

3. Call the type library object's **Write** method to create the .tlb file, then register it.
4. If you want to create an additional type library, call the **GenerateTypeLib** object's **Reset** method to clear the list of script component files in the **AddURL** property, reset the URLs and any other properties you want, and then call the **Write** method again.

For example, the following script in a <registration> element creates a type library dynamically.

Note A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<registration
   description="My Test Component"
   progid="Component.TestScript"
   version="1"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}">
   <script language="VBScript">
   <![CDATA[
   Function Register()
      Set oTL = CreateObject("Scriptlet.GenerateTypeLib")
         oTL.AddURL "d:\components\MyComponent.wsc"   ' Script component URL.
         oTL.AddURL "d:\components\YourComponent.wsc"
         oTL.Path = "d:\components\MyComponent.tlb"   ' .tlb path.
         oTL.Doc = "Sample component typelib"   ' Documentation string.
         oTL.GUID = "{a1e1e3e0-a252-11d1-9fa1-00a0c90fffc0}"
         oTL.Name = "MyComponentTLib" ' Internal name for tlb.
         oTL.MajorVersion = 1
         oTL.MinorVersion = 0
         oTL.Write ' Write tlib to disk.
         oTL.Reset ' Clear list of URLs in AddURL/.
      End Function
   ]]>
   </script>
</registration>
```

Command-line Interface

If you are comfortable using the **Command Prompt** window, you can call the Rundll32.exe program to create type libraries.

To create a type library from the command prompt

- Call Rundll32.exe using this syntax:

  ```
  rundll32.exe path\scrobj.dll,GenerateTypeLib options
  ```

  Where:
  - *path* The path where Scrobj.dll is located on your computer.
  - *options* A set of flags and values you can use to specify type library information in the form *-flag*:*value*. You can specify the options in any order. The following flags are supported, with values as described under the previous section, To create a script component type library dynamically.
    -name:*Name*
    -file:*Path*
    -doc:\"*Doc*\"
    -guid:*GUID*
    -major:MajorVersion
    -minor:MinorVersion
    -URL:AddURL

For example, the following command creates a type library called MyComponent.tlb from the script component MyComponent.wsc (the command is wrapped for clarity):

```
rundll32.exe c:\winnt\system32\scrobj.dll,GenerateTypeLib
   -name:MyComponentTLib -file:d:\components\MyComponent.tlb
   -doc:\"Sample component typelib\"
   -guid:{a1e1e3e0-a252-11d1-9fa1-00a0c90fffc0} -major:1 -minor:0
   -URL:d:\components\MyComponent.wsc
```

Troubleshooting Type Libraries

The process of generating a type library can fail for various reasons. The error you see might not be clear enough in all cases to make it obvious where the problem lies. If you are unable to generate a type library, review the following list of likely causes for the failure.

- If a property is defined by functions, the **get** and **put** functions must have the same number of arguments with the same names. For details, see Exposing Properties.

  Note It is possible to define a script component in which the **get** and **put** property functions have different numbers of arguments, but you cannot create a type library for that script component.

- If you are exposing events, you cannot use the same dispatch identifiers (*dispid*) more than once in a script component. Additionally, you cannot use a negative value for the dispid unless it is within a specified range. For details, see Exposing Methods.
- The ID attributes of elements in the script component must be unique. If you are generating a type library from more than one script component, then the IDs must be unique in the type library as a whole.

See Also

Script Component File Contents | Creating Registration Information | Checking For Errors in Script Component Files | Script Component Files and XML Conformance

Referencing Other Components

Your Windows® Script Component can include references to external components that you need to create the script component, such as:

- Additional COM objects
- Type libraries
- Resources, such as numbers and text, that you do not want to hard-code into your script component's scripts.

Referencing Additional COM Components

In your script component, it might be necessary to create instances of other COM components as needed. You can do so in two ways:

- **In script** Create instances of other objects directly in your script. For example, you can use the **CreateObject** function in Microsoft® Visual Basic® Scripting Edition (VBScript) or the new **ActiveXObject** Object in JScript.
- **Using an OBJECT element** Use an <object> element similar to the <OBJECT> tag you use in HTML pages. Using an <object> element makes objects globally available to any script and allows scripting tools to provide statement completion. It also provides a convenient way to summarize and document the objects you use in your script component.

    Note Although <object> elements in script components are similar to <OBJECT> tags in HTML pages, the list of attributes for an <object> element in a script component is shorter, because script components do not present a user interface.

To create an OBJECT element

- Create an <object> element. This element should be inside the <component> element, but outside of any other element such as a <script> element.

The following example shows an object definition in a script component.

```
<object id="cnn" progid="ADODB.Connection"/>
```

Referencing an External Type Library

Many components that you might work with support type libraries, which provide a complete listing of the component's classes and their members. By referencing the component's type libraries, you can use constants defined in the type library.

To include a reference to a type library

- Include a <reference> element in your script component that specifies the location and name of the type library to include. For example:

  ```
  <reference object="ADODB.Connection.2.0"/>
  ```

Referencing Resources

Resource elements can include information that might change between versions, strings that might be translated, and other values.

To reference resources

1. In the script component, but outside of the <public> and <script> elements (and <implements> element, if any), create one <resource> element for each resource you want to define, giving each element a unique ID. The following example shows two <resource> elements:

   Note A CDATA section is required to make the contents of the <resource> element opaque to the parser. For details, see Script Component Files and XML Conformance.

   ```
   <component id="MyScriptlet">
   <public>
      <method name="random" internalName="getRandomNumber"/>
   </public>
   <resource id="errNonNumeric"><![CDATA[Non-numeric value passed]]>
   </resource>
   <resource id="errOutOfRange"><![CDATA[Passed value is out of range ]]>
   </resource>
   ```

2. In your script, include the resource text or number by calling the getResource function, as shown in the following example.

   Note A CDATA section is required to make the script in the <script> element opaque to the parser. For details, see Script Component Files and XML Conformance.

   ```
   <script language="VBScript">
   <![CDATA[
   Function getRandomNumber(upperBound)
      If IsNumeric(upperBound) Then
   ```

```
            getRandomNumber = Cint(upperBound * Rnd + 1)
         Else
            getRandomNumber=getResource("errNonNumeric")
         End If
      End Function
      ]]>
      </script>
```

See Also

Script Component File Contents | Referencing Another Script Component in the Same Package

Windows Script Components

Referencing Another Script Component in the Same Package

You can create a package that contains multiple Windows® Script Components, so you can also instantiate and use one of the script components from the other without having to register the second script component.

For example, you might create one script component that implements the **Automation** interface and exposes a series of properties and methods. A second script component in the same package might implement the ASP interface, which provides access to the server object model for Microsoft® Internet Information Services (IIS). You can then create a method or property in the Automation script component that exposes the ASP script component and makes its members available.

To reference one script component from another, create a skeleton member — a property or method — in one method that exposes the second script component.

To reference another script component in the same script component file

1. Declare a property or method in the first script component.

2. As part of the definition for the new property or method, call the createComponent function.

   For example, the following shows two script components in the same package. In the first script component, the math method simply references the second script component, which exposes the **add** method and the **multiply** method.

   > **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<?XML version="1.0"?>
<package>
<component id="component1">
<registration progid="Component.FrontEnd"/>
<public>
   <property name="math"/>
</public>
<script language="JScript">
<![CDATA[
var math = createComponent("component2")
   ]]>
</script>
</component>

<component id="component2">
<registration progid="Component.Math"/>
<public>
   <method name="add"/>
   <method name="multiply"/>
</public>
<script language="JScript">
<![CDATA[
function add(n1, n2){
   return n1+n2;
}
function multiply(n1, n2){
   return n1*n2;
}
]]>
</script>
</component>
</package>
```

To invoke the referenced script component, call the full member hierarchy to get to its methods or properties. The following example illustrates a few ways to do this:

```
' Creates instance of first script component.
set o1 = CreateObject("Component.FrontEnd")

' Invokes the second script component's functions directly.
msgbox(o1.math.add(3,5))
msgbox(o1.math.multiply(3,5))

' Creates a second object that references the math method directly.
Set o2 = o1.math()
msgbox(o2.add(4,5))
msgbox(o2.multiply(4,5))
```

Each time you call the **createComponent()** function, it creates a new instance of the referenced script component. If you need to preserve instance information between calls, store the pointer to the second script component in a global variable, as in the following example.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<component id="component1">
<registration progid="Component.FrontEnd"/>
<public>
   <property name="math">
      <get/>
   </property>
</public>
<script language="JScript">
<![CDATA[
var m = createComponent("component2")
function get_math(){
   return m
}
   ]]>
</script>
</component>

(Component2 as in previous example)
```

See Also

Script Component File Contents | <package> | <component>

Build: Topic Version 5.6.9309.1546

Windows Script Components

Checking For Errors in Script Component Files

Because Windows® Script Components are used as COM components, they normally run silently. However, while you are developing your script component file, you might find it useful to know about errors in the file. You can specify three types of error checking:

- Check for XML validity. For details, see Script Component Files and XML Conformance.
- Allow notification for syntax and run-time errors. By default, if an error occurs in a script component file, a generic error message is displayed. By specifying error notification, you can have the script component parser display details about the error.
- Allow debugging. By default, you cannot use the script debugger for script components. If you enable debugging, you can launch the script debugger in response to an error, by setting a breakpoint, or with a Stop (Microsoft® Visual Basic® Scripting Edition (VBScript)) or debugger (JScript) statement.

Setting Error Options

Specify error options as attributes of the XML <?component?> processing instruction.

To specify error checking

- Include the <?component?> processing instruction at the top of the script component file (but after the optional <?XML ?> declaration) with one or more of the following attributes:
  - ***error*** Set this to true to display detailed error messages for syntax or run-time errors in the script component.
  - ***debug*** Set this to true to enable debugging. If this debugging is not enabled, you cannot launch the script debugger for a script component in any way.

For example, the following enables all three error reporting options:

```
<?component error="true" debug="true"?>
```

Tip Turn error reporting off before deploying your script component to a production environment.

See Also

Script Component File Contents | Using The Script Component Wizard | Creating Registration Information | Creating a Script Component Type Library | Script Component Files and XML Conformance

Windows Script Components

Script Component Files and XML Conformance

Windows® Script Components are XML files and should follow the XML version 1.0 conventions for how elements are defined in the file. Although XML elements superficially resemble HTML tags, XML is a more strictly defined protocol. For example, element names are case-sensitive.

To make it easier to create script components, the script component run-time (Scrobj.dll) allows you to specify how strictly you want the XML in your file to be interpreted. Script component XML can be interpreted fairly loosely, allowing the same sorts of variations in tags that HTML does.

However, you can also specify XML validation in your script component file, which causes the script component run-time to check that your script component file's XML conforms closely to the XML standard. If you will ever be using an XML editing tool to work with your script component, you should set XML validation.

When you specify XML validation, you must make sure that your script component file conforms to the following rules.

- **Element types and attribute names are case-sensitive.** Element types and attributes names are usually all lowercase, except multipart names such as internalName. In XML files that conform closely to XML standards, all names must follow naming conventions exactly.
- **Attribute values require quotation marks.** Values for attributes must be in single or double quotation marks. For example, without XML validation, you can specify language=JScript in a <script> element. But to follow XML rules, the same attribute would be language="JScript".
- **Reserved characters in script elements must be made opaque.** Script elements frequently include greater than (<) and less than (>) symbols, the ampersand symbol (&), and other characters that are reserved in XML. If you are creating a closely conformant XML file, you must make sure that these reserved characters do not confuse the XML parser. You can individually mark characters with escape sequences (for example, specify "<" as "&lt;"), or you can make an entire script element opaque by enclosing it in a <![CDATA[...]]> section. Without XML validation, the script component XML parser treats <script> elements as opaque by default.

  > **Note** Do not include a CDATA section in script if you have not specified closely conformant XML parsing. If you do, the script component will report errors when you try to register, instantiate, or call it.

For more details about XML standards, see the XML specification Web site and the Microsoft® XML Web site.

To specify XML conformance

- Include the <?XML ?> declaration as the first element in your file:

  ```
  <?XML version="1.0" ?>
  ```

  > **Note** If you create a script component using the Script Component Wizard, the <?XML ?> declaration is added to the file, and the script component's XML is parsed strictly.

If this element is missing, the script component run-time assumes that you do not want XML validation. However, you will probably not be able to work with the file using an XML editor.

See Also

Script Component File Contents | Using The Script Component Wizard | Checking For Errors in Script Component Files

Build: Topic Version 5.6.9309.1546

Windows Script Components

Using Script Components

After you have created a Windows® Script Component, you can use it in your applications. Using script components is similar to using other COM components, however, each host application uses a slightly different way to instantiate COM components.

- Registering a Script Component Register and unregister a script component.
- Using a Script Component in an Application Create an instance of a script component in a host application.
- Handling Script Component Events in the Host Application Detect and respond to custom script component events.

Build: Topic Version 5.6.9309.1546

Windows Script Components

Registering a Script Component

To register a Windows® Script Component, you must have the script component run-time Scrobj.dll available, and it must be registered on your computer. When you install the script component package from the Microsoft® Scripting Technologies Web site, Scrobj.dll is automatically loaded and registered on your computer.

> **Note** Registration is not required if you are creating a script component to be called via DHTML Behaviors, because Microsoft®Internet Explorer 5.0 can use this type of script component without prior registration.

This topic provides information about:

- Registering a Script Component

- Registering a Script Component for Remote Instantiation
- Unregistering a Script Component

Registering a Script Component

You can register a script component on a local computer using a variety of methods.

To register a script component

- In Windows Explorer, right-click the script component (.wsc) file, and then choose **Register**.

  –or–

- Use the new version of Regsvr32.exe that ships with the script component package, and use this command:

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

  For example, to register a script component called MyComponent.wsc, use syntax such as this:

  ```
  regsvr32 file:\\myserver\MyComponent.wsc
  ```

  -or–

- If you do not have the version of Regsvr32.exe that ships with the script component package, use the existing version and register the script component run-time DLL using syntax such as the following:

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

  For example, to register a script component called MyComponent.wsc, use syntax such as this:

  ```
  regsvr32 scrobj.dll /n /i:http://myserver/MyComponent.wsc
  ```

Registering a Script Component for Remote Instantiation

If you intend to create a remote instance of a script component, the script component must be registered on the remote computer where it resides. You must also register it on each local computer from which you intend to instantiate the script component, so that DCOM can have a starting point in the registry from which to find and instantiate the remote script component.

To register a script component for remote instantiation

1. Determine the program ID and class ID of the script component to be instantiated remotely.

   Note The script component must have the same class ID on both the local and remote computers, so you must provide a class ID in the script component's <registration> element.

2. On each local computer, create the following registry entry:

   ```
   HKEY_CLASSES_ROOT\componentProgID
   ```

   where *componentProgID* is the program ID of the script component to instantiate.

3. Under the new entry, create the CLSID key and set its value to the class ID of the script component, enclosing the class ID in brackets.

   Tip An easy way to create the proper registry information is to register the script component on the server where it will be instantiated. Then using Regedit.exe, locate the HKEY_CLASSES_ROOT\componentProgID entry. From the **Registry** menu, choose **Export Registry File**, which creates a .reg file. This file can be distributed to local computers, and users can simply run the file to create the appropriate registry entries.

Unregistering a Script Component

If you no longer need to have a script component registered on your computer, you can unregister it using one of these methods:

- Right-click the script component file name in Windows Explorer, and then choose **Unregister**.
- Run Regsvr32.exe with the -u flag.

See Also

Creating Registration Information | Script Component File Contents | Using a Script Component in an Application

Build: Topic Version 5.6.9309.1546

Windows Script Components

Using a Script Component in an Application

After creating a Windows® Script Component, you can use it as you would any COM component, by calling it from a host application such as Microsoft® Visual Basic®, C++, Microsoft® Internet Explorer, or other applications.

> **Note** Script components written for DHTML Behaviors are instantiated differently than traditional COM objects. For details, see Using DHTML Behaviors on the Microsoft Site Builder Network (SBN) Web site.

There are a variety of options for creating an instance of the script component, depending on the host application, the type of script component you are using, and where the script component is deployed. The primary difference, however, is whether you want to create an instance of the script component locally (on the same computer as the application) or remotely (on another computer).

In either case, there are a few points to bear in mind. If you create an instance of the script component and change the .wsc file while using that instance, your instance of the component is not updated. To update it, create a new instance of the script component.

The exact properties and methods you can use are defined by the <public> element and by scripts in the script component file. If you are working in an environment that supports statement completion, such as Visual Basic, you can see a script component's properties and methods if you generate and use a type library. For details, see Creating a Script Component Type Library.

If your attempt to create an instance of the script component fails, a likely cause is a syntax or run-time error in the script component file. A parsing error in any XML element (including the <registration> element) can cause the instantiation to fail. While you are developing your script component file, set error checking options in the <?component?> processing instruction as described in Checking For Errors in Script Component Files.

> **Tip** To make it easier for the host application to know about the COM interfaces exposed by a script component, the script component run-time can generate a type library, which contains information about the properties, methods, and events available in the script component. For details, see Creating a Script Component Type Library.

Creating Local Instances of Script Components

If the script component is installed on the same computer as the host application, you can register the script component as its own component as described in Registering a Script Component. You can then use the host application's normal means for creating an object instance, such as the **CreateObject** function. For example, to create an instance of the script component that was registered with the program ID Component.MyComponent in Visual Basic, use a statement such as this:

```
Set oComponent = CreateObject("Component.MyComponent")
```

> **Note** If your host application is Visual Basic and you want to handle events fired by the script component, you must bind the object early with a **Dim** statement that includes the **WithEvents** keyword, such as the following:
>
> ```
> Dim WithEvents scMyComponent As MyComponent
> Private Sub Command1_Click()
> Set scMyComponent=CreateObject("MyComponent")
> End Sub
> ```

> **Note** For details, see Handling Script Component Events in the Host Application. This is not necessary if you do not intend to write handlers for script component events.

On a Web page, you can use the <OBJECT> tag to create an instance of the script component. You must know the class ID of the script component and include it in the <OBJECT> tag, as in the following example:

```
<OBJECT
   ID="oComponent"
   CLASSID="clsid:855c8606-49ba-11d2-a428-00c04f8ec80b">
</OBJECT>
```

If the script component is not registered on the local computer, you can use the script component moniker to create an instance of it. The moniker is supported in functions such as **GetObject**. The script component run-time, Scrobj.dll, must be registered on the local computer.

> **Note** The **GetObject** function is not supported for script components in Microsoft® Internet Explorer for security reasons.

For example, the following illustrates how to call the Visual Basic **GetObject** function to create an instance of an unregistered script component:

```
Set oComponent = GetObject("script:c:\COM\MyComponent.wsc")
```

If the .wsc file referenced by the moniker contains more than one script component, you can specify the script component to instantiate by adding its name to the file name with a hash mark (#) as a delimiter. The following creates an instance of the script component whose ID is "math" contained in the file MyComponent.wsc:

```
Set oComponent = GetObject("script:c:\COM\MyComponent.wsc#math")
```

Using a URL moniker allows you to create an instance of a script component that resides on another computer, such as a Web server. Use a complete URL (with HTTP protocol) as the location for the script component, as in the following example:

```
Set oComponent = GetObject("script:http://myserver/MyComponent.wsc")
```

Internet Explorer 5.0 supports DHTML Behavior syntax for creating instances of script components, which works somewhat differently than the traditional syntax for instantiating objects and will ensure that the script component cannot access potential unsafe system objects. For an example, see Using DHTML Behaviors on the Microsoft Site Builder Network (SBN) Web site.

Creating Remote Instances of Script Components

If the remotable attribute of a script component's <registration> element has been set to "true," the script component can be instantiated remotely from another computer using Distributed COM (DCOM).

Both computers must have basic DCOM installed. A computer is correctly configured with DCOM if it is running any of the following:

- Windows NT 4.0
- Windows 95 running Internet Explorer 4.0
- Windows 95 with the OEM Service Release 2 (OSR2) or later. For details, see the Windows 95 OSR2 page on the Microsoft® Web site.
- Windows 95 with DCOM for Windows 95, version 1.2. For details, see the DCOM for Windows 95 page on the Microsoft® Web site.

The script component itself must be deployed as follows:

- On the local computer, you do not require either the script component itself (.wsc file) or the script component run-time (Scrobj.dll) at the time you instantiate the script component. However, you must have a reference to the remote script component in the local Windows registry for DCOM. For details, see Registering a Script Component.
- On the remote computer, you require the script component and the script component runtime. Both must be registered.

When you create an instance of a remote script component, it works within your application as if it were a local object; you call its methods and get and set its properties normally. However, the remote script component's script runs on the remote machine and has access to that machine's resources (within any limitations imposed by security, and so forth). Communication between the host application on the local machine and the script component on the remote machine is handled automatically and invisibly by DCOM.

To create a remote instance of a script component, call the **CreateObject** method, passing it the name of the remote computer as a parameter.

> **Note** The ability to use CreateObject for instantiating remote script components requires Visual Basic 6.0 or later or VBScript 5.0 or later.

The following Visual Basic example shows how to do this on a computer named "myserver":

```
Set newS = CreateObject("Component.MyComponent", "myserver")
```

> **Note** There can be a slight delay when you first instantiate a remote script component while DCOM establishes communication between the computers.

See Also

Creating Registration Information | Creating Script Components | How Script Components Work | Introducing Script Components | Registering a Script Component

---

Windows Script Components

Handling Script Component Events in the Host Application

Most host applications can receive Windows® Script Component events as they do any other events. However, some host applications require some setup before they can receive script component events.

> **Note** If you are creating a Behavior script component, events are exposed using the DHTML object model. For details, see Exposing Custom Events in Behavior Script Components.

In Visual Basic, for example, you must use early (compile-time) binding to the component in order to receive events. Early binding requires a type library, so you must have generated a type library for the script component. For details, see Creating a Script Component Type Library.

In addition, when you declare the object variable for the component, you must specify the WithEvents keyword. (The class name used in the **Dim** statement is the ID that you assigned in the script component's <component> element.)

An example in Visual Basic might look like this:

```
Dim WithEvents Scriptlet1 as MyScriptlet
Set Scriptlet1 = CreateObject("MyScriptlet")
Sub Scriptlet1_namechanged
   MsgBox("Value of name property changed")
End Sub
```

See Also

Exposing Events

Windows Script Components

Implementing ASP Script Components

Using a Windows® Script Component, you can include the functionality of Active Server Pages (ASP). Doing so allows you to isolate server script into a component that can be easily called from ASP pages and which makes it easy to reuse ASP code. For example, you might have several ASP pages that process forms. Rather than implement all the form-processing logic directly in each ASP page, the ASP pages could all call a script component with form-handling logic that you create.

To create an ASP script component, create a script component normally, as described in Script Component File Contents. Use the <implements> element to implement the ASP interface handler, setting the type attribute of the <implements> element to "ASP." Doing so provides the means for accessing ASP objects: Response, Request, Server, Session, and Application. In an ASP script component, you can use these objects as you would directly in an ASP page. The ASP interface handler is built into the script component run-time (Scrobj.dll), so no external interface handler is required.

When the script component runs, it uses the same namespace as the ASP page that called it. The script component can access the Request and Server objects as if it were in the ASP page that called it. The script component also has access to the same Session and Application variables that the calling ASP page does. Similarly, if the ASP script component calls a method on the Response object, the result of these calls is available in the calling page. For example, if you call Response.Write, the output is directed to the calling ASP page.

The following shows a simple ASP script component that exposes a property and a method. The **applicationVar1** property makes available the hypothetical Application variable called Var1. The **AddDate** method outputs the current date into the calling ASP page.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<component id="ASPScriptlet">
<registration progid="ASPScriptlet"/>

<public>
   <property name="applicationVar1"/>
   <method name="AddDate"/>
</public>

<implements type="ASP"/>
<script language="VBScript">
<![CDATA[
dim applicationVar1
applicationVar1 = Application("Var1")
Sub AddDate()
   Response.Write(Date)
End Sub
]]>
</script>
</component>
```

The calling ASP page might look like this:

```
<HTML>
<HEAD>
<TITLE>Testing the Script Components ASP Handler</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>Testing the ASP Handler</H1>
<%Set wscASP = CreateObject("ASPScriptlet")%>
<P>The current date is <%= wscASP.AddDate()%></P>
```

```
<P>The value of the Application(Var1) variable is <%= wscASP.applicationVar1%></P>

</BODY>
</HTML>
```

Build: Topic Version 5.6.9309.1546

Windows Script Components

Implementing DHTML Behavior Script Components

Windows® Script Components provide a lightweight, easily maintainable way to create components for implementing DHTML Behaviors available in Microsoft® Internet Explorer 5.0. Using a script component allows script developers to implement behaviors using Microsoft® Visual Basic® Scripting Edition (VBScript), Microsoft® JScript® (ECMAScript), or any third party scripting language that supports the Microsoft® ActiveX® Scripting interfaces.

For general information about DHTML Behaviors, see **http://msdn.microsoft.com/workshop**.

By creating Behavior script components, you can:

- Expose behaviors by binding functions in the script component to events raised in the containing document, either for specific elements or for the DHTML document or DHTML window objects.
- Define layouts, which contain HTML text that is inserted into the containing document when the behavior is called.
- Expose custom properties and methods that extend the set of members for an element in the containing page.
- Expose custom events which the script component can fire at any time. For example, a script component can fire a custom event after changing the contents of a DHTML element on the page.
- Bind a script component function to certain standard events such as when the document is loaded or when the content of the element changes.

In this section you will find information about the following:

- Creating a Behavior Script Component See what elements to put into a Behavior script component file, how to share information with the containing document, what DHTML features apply to Behavior script components, and what to watch for when scripting.
- Exposing Properties and Methods in Behavior Script Components Create properties and methods that extend those already available in DHTML elements.
- Exposing Custom Events in Behavior Script Components Expose and fire events from the script component to the containing document.
- Behavior Handler Reference A list of XML elements, properties, and methods used in Behavior script components.

See Also

The "DHTML Behaviors" and "Using DHTML Behaviors" topics on the Site Builder Network (SBN) Workshop on the Microsoft Web site provide general

---

Windows Script Components

Creating a Behavior Script Component

Creating a Behavior script component is similar to creating any other kind of script component, except that you are linking Microsoft® Internet Explorer events to script that is run in response to those events.

This topic is divided into the following sections:

- Creating a Behavior Script Component File
- Behavior-Related Enhancements to the DHTML Object Model
- Getting Event Parameters in the Script Component
- Scope Rules
- Timing Considerations

Creating a Behavior Script Component File

A Behavior script component includes an <implements> element that specifies the Behavior interface handler. Within the <implements> element, you use:

- <attach>elements to bind events from the containing document to functions created in a separate <script> element in the script component.
- <layout>elements to define HTML text to be inserted into the containing document.
- <event>elements to define custom events that will be fired by the script component.

Behavior script components can also include custom properties and methods that extend those already available for the element in the containing document. For details, see Exposing Properties and Methods in Behavior Script Components.

The following example shows a Behavior script component that changes the color of an element in the containing page whenever the mouse is passed over the element. To do this, the sample binds the DHTML onmouseover and onmouseout events to functions in the script component that set the element's DHTML style attribute. The sample also sets the document's link color when the document is initialized by binding to the DHTML window object's onload event.

In addition to binding events to script, the script component also inserts text into any <H1> elements in the containing document that are linked to this script component. Finally, it exposes and fires an event called onchange, which extends the DHTML window object's event object with a custom property called newvalue.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<?XML version="1.0"?>
<component>
<implements type="Behavior">
   <comment>The following will cause the do_nmousedown and
   do_mouseout functions to be called when the mouse is
   passed over the element in the containing document.</comment>

   <attach event="onmouseover" handler="do_onmouseover"/>
   <attach event="onmouseout" handler="do_onmouseout"/>

   <comment> This will call the init function when the onload
   event of window is fired.</comment>
   <attach for="window" event="onload" handler="docinit"/>

   <comment>The following defines HTML text that will appear in
   the containing document.</comment>

   <layout>
      <h1>This is the HTML to show in the element</h1>
   </layout>

   <comment>The following defines a custom event that is fired
   from within the script component by the fireEvent method.</comment>

   <public>
      <event name="onchange"/>
   </public>

</implements>
<script language="JScript">
<![CDATA[
var normalColor, normalSpacing;
function do_onmouseover(){
   // Save original values.
   normalColor = style.color;
   normalSpacing= style.letterSpacing;
   style.color = "red";
   style.letterSpacing = 2;
   oEvent = createEventObject();
   oEvent.newcolor = "red";
   fireEvent("onchange",oEvent);
}
function do_onmouseout(){
   // Reset to original values.
   style.color = normalColor;
   style.letterSpacing = normalSpacing;
}

function docinit(){
   document.linkColor = "red";
}
]]>
</script>
</component>
```

There are a few things to point out in the preceding code:

- From a script component procedure, the implied *this* pointer in an event handler refers to the containing function, not to the element on which the event is being fired.
- Just as in an HTML page, it is possible to place inline script within a <script> in a script component. In this instance, the global variables normalColor and normalSpacing are defined in inline script.

> **Note** Inline script is executed even before the behavior is applied to the element, which limits what statements can be executed in inline script. For example, if the same behavior in the example exposed a property called **hiliteColor**, the inline script could refer to **hiliteColor** directly (in other words, it resolves against the script component's namespace). It is illegal, however, to refer to **hiliteColor** as *Behavior*.element.hiliteColor from an inline script, because at that point, the behavior has not yet been applied to the element. For more information, see Scope Rules and Timing Considerations later in this topic.

Behavior-Related Enhancements to the DHTML Object Model

The following enhancements were made to the DHTML object model for Microsoft® Internet Explorer 5 in order to add support for behaviors.

- The cascading style sheet (CSS) behavior attribute specifies the location of the behavior.
- The DHTML **attachEvent** and **detachEvent** methods enable a Behavior script component to receive event notifications from the containing page, by specifying a function that gets called whenever the event fires on the object.
- The DHTML **uniqueID** property enables a behavior script component to assign an ID to the element. This can be useful in cases where a script component injects code into the containing page and needs to specify the ID of the element to which the behavior is being applied.

Getting Event Parameters in the Script Component

In DHTML, the DHTML event object provides information about the event. Although in DHTML the event object is accessible to event handlers through the DHTML window object, in a behavior script component the event object is passed in as a parameter to the event handler.

The following code from a hypothetical calculator script component demonstrates keyboard and mouse events bound to a script component function called **doCalc**. The **doCalc** function uses the event object to get information about the conditions under which the event was fired.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<implements type="Behavior">
```

```
   <attach event="onclick" handler="doCalc"/>
   <attach event="onkeydown" handler="doCalc"/>
</implements>

<script language="jscript">
<![CDATA[
function doCalc(oEventParam){
   oElement = oEventParam.srcElement;
   if(oEventParam.type == "keydown"){
      sVal = KeyCodeToChar(oEventParam.keyCode);
   }
   else{
      if (oEventParam.srcElement.type != "button"){
         return;}
      sVal = stripBlanks(oEventParam.srcElement.value);
   }
}
// other script here
]]>
</script>
```

Scope Rules

When working with script components, you actually work with three namespaces: the behavior, the element, and the containing document. Scope rules define the order in which name conflicts are resolved in a behavior script component. A name will be resolved in this order:

The name is resolved to one defined by the behavior anywhere in the script component, whether a variable, a behavior-defined property, a method, or an event.

If the name is not resolved successfully, it is resolved to a property, method, or event that applies to the element.

Finally, the name is resolved to the name of a property, method, or event that applies to the window object in the containing page.

In the following example, note how the names have been resolved, using the scope rules defined above:

- ***normalColor*** Resolves to the variable defined by the behavior at the top of the script.
- ***style*** Resolves to the style property of the element in the containing document.

```
<implements type="Behavior">
   <attach event="onmouseover" handler="do_onmouseover"/>
   <attach event="onmouseout "handler="do_onmouseout"/>
</implements>

<script language="JScript">
<![CDATA[
var normalColor, normalSpacing;
function event_onmouseover()
{
   // Save original values.
   normalColor = style.color;
   normalSpacing = style.letterSpacing;

   style.color = "red";
   style.letterSpacing = 2;
}
function event_onmouseout()
{
   // Reset to original values.
   style.color = normalColor;
   style.letterSpacing = normalSpacing;
}
]]>
</script>
```

Timing Considerations

When creating behaviors, it is important to know when the behavior is applied to the element. Until the behavior has been applied, scripts cannot have access to the values of behavior-defined properties that might be set in the document.

Because the behavior is encapsulated in a separate file from the HTML document, it is downloaded separately from the rest of the document. As the document and behavior are parsed and loaded, the behavior receives notifications of progress through the function specified with the attachNotification method. Currently, the behavior is notified with a "contentChange" or a "documentReady" notification. The "contentChange" notification is sent when the content of the element to which the behavior has been attached has been parsed, and any time thereafter that the content of the element is changed. The "documentReady" notification is sent when the document has been downloaded and parsed.

Because inline script in the script component file is executed as soon as the behavior is instantiated, the values of behavior-defined attributes and properties that are being set in the document may not be accessible from an inline script. However, these properties will be available as soon as the first "contentChange" notification is sent.

See Also

Exposing Properties and Methods in Behavior Script Components | Exposing Custom Events in Behavior Script Components | Behavior Handler Reference

---

Windows Script Components

Exposing Properties and Methods in Behavior Script Components

Behavior script components can expose custom properties and methods in a manner similar to Automation script components. Behavior script component properties and methods extend the properties and methods available to elements in the containing page. For example, you might create a behavior script component that changes the color of an element when the mouse is passed over it. By defining a property in the script component, you can make available a custom property in the document, perhaps called **hiliteColor**, that allows the Web page author to set the color with which text gets highlighted.

A behavior can override an element's default behavior by exposing a property or method of the same name as that which is already defined for the element.

Properties and methods are defined in a <public> element separate from the <implements> element used to specify the Behavior handler. For details, see Exposing Properties and Exposing Methods.

Exposing a Property

Properties are exposed in a <public> element, as in any script component. The following script component fragment shows a Behavior script component that exposes the custom property **hiliteColor**. If the containing page does not specifically set the property's value, the default is set to "red."

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files

and XML Conformance.

```
<public>
   <property name="hiliteColor"/>
</public>

<implements type="Behavior">
   <attach for="window" event="onload" handler="event_onload">
</implements>

<script language="JScript">
<![CDATA[
   var hiliteColor;
   function event_onload(){
      // Initialize the properties.
      if (hiliteColor == null){
         hiliteColor = "red";}
   }
   // Further script here.
]]>
</script>
```

Exposing a Method

Exposing a method in a Behavior script component is identical to doing so in an Automation script component. For details, see Exposing Methods. In a Behavior script component, methods exposed in the script component extend those already available for the element in the containing document.

See Also

Exposing Custom Events in Behavior Script Components

Build: Topic Version 5.6.9309.1546

Windows Script Components

Exposing Custom Events in Behavior Script Components

Behavior script components, like Automation script components, can expose custom events that are fired inside the script component and can be handled in the containing document. An event is declared in the <public> element as in the following:

```
<public>
   <event name="onResultChange" />
</public>
```

The event can then be fired by calling the fireEvent method in script:

```
<script language="JScript">
   // Other code here.
   fireEvent("onResultChange");
   // Other code here.
</script>
```

A behavior can override an element's default behavior by exposing an event of the same name as one that is already defined for the element. For example, a behavior that exposes an onclick event can override the element's default onclick event.

Getting and Setting Custom Event Information

Your custom script component event can get access to the DHTML event object, which maintains event-specific information. You can use this object to:

- Change existing event object properties, such as **keyCode**, that have been set by in the calling event.
- Create new (expando) properties of the event object in order to pass information about your event back to the containing event.

In your script component code, call the createEventObject method to create a new instance of an event object, and then set one or more properties on the new event object. When you call the fireEvent method, you can pass the new event object with the event name.

To create a new expando property, simply name it when assigning its value in your script. The following shows how you can create a new property called myprop for the event object:

```
oEvent = createEventObject();
oEvent.myprop = "a new value"
```

> **Note** You can create expando properties only if you are using Microsoft® JScript® (or JavaScript). This feature is not supported in Microsoft® Visual Basic® Scripting Edition (VBScript).

Example

The following script component fragment is derived from a calculator script component sample. The sample defines a custom onResultChange event that is fired to the containing document whenever the result changes. Event-specific information (the actual calculation result) is passed via the expando property called **result**.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <event name="onResultChange" />
</public>

<implements type="Behavior">
   <attach event="onclick" handler="doCalc"/>
</implements>

<script language="JScript">
<![CDATA[
function doCalc()
{
   // Code here to perform calculation. Results are placed into
   // the variable sResult.
   oEvent = createEventObject();
   oEvent.result = sResult;
   fireEvent("onResultChange",oEvent);
}
]]>
</script>
```

The following shows what the containing page looks like. When the onResultChange event fires, the results of the calculation are extracted from expando property result of the DHTML window.event object and displayed in the resultWindow <DIV> element.

```
<HTML>
```

```
<HEAD>
<xml:namespace prefix="LK" />
<style>
   LK\:CALC {behavior:url(calc.wsc)}
</style>
<script language="JScript">
function showResults(){
   resultWindow.innerText=window.event.result;
}
</script>
</HEAD>

<LK:CALC id="myCalc" onResultChange="showResults()">
<TABLE>
<TR>
   <DIV ID=resultWindow
      STYLE="border: '.025cm solid gray'"
      ALIGN=RIGHT>0.</DIV>
</TR>
<TR>
   <TD><INPUT TYPE=BUTTON VALUE=" 0 "></TD>
   <TD><INPUT TYPE=BUTTON VALUE="+/-"></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" . "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" + "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" = "></TD>
<TR>
</TABLE>
</LK:CALC>
</HTML>
```

See Also

Exposing Properties and Methods in Behavior Script Components

Windows Script Components

Script Component Reference

When creating Windows® Script Component files, use XML elements to define different parts of your script component, such as the registration information, scripts, and objects that you intend to reference in your script component's code. You can also call methods to perform tasks in the script component. The following table lists the available XML elements and methods.

> **Note** Specific handlers, such as the Behavior handler, expose additional XML elements, methods, and other members. For more information, refer to the documentation for specific handlers.

XML Elements

- <?component?>
- <?XML ?>
- <comment>
- <component>
- <event>
- <implements>
- <method>
- <object>
- <package>
- <property>
- <public>
- <reference>
- <registration>
- <resource>
- <script>

Methods and Functions

- createComponent
- fireEvent
- getResource

Build: Topic Version 5.6.9309.1546

Windows Script Components

<?component?>

Specifies attributes for error handling.

```
<?component error="flag" debug="flag" ?>
```

Values

flag
A Boolean value: "true" or "false," 1 or 0, or "yes" or "no." The default value for all attributes is false.

Remarks

Because script components are used as COM components, they normally run silently. However, while you are developing your script component file, you might find it useful to be notified about errors in the file. You can specify these types of error notification:

- **error** Set this to true to allow error messages for syntax or run-time errors in the script component file.
- **debug** Set this to true to enable debugging. If this debugging is not enabled, you cannot launch the script debugger for a script component in any way.

Example

```
<?component error="true" debug="true" ?>
```

See Also

Checking For Errors in Script Component Files | Script Component File Contents

Build: Topic Version 5.6.9309.1546

Windows Script Components

<?XML ?>

Indicates that the file should be parsed as XML.

`<?XML` `version="`*`version`*`" standalone="`*`DTDflag`*`" encoding="`*`encname`*`"` **`?>`**

Values

version
: A string in the form *n.n* specifying the XML level of the file. Use the value 1.0.

DTDflag
: (Optional) A Boolean value indicating whether the XML file includes a reference to an external Document Type Definition (DTD). Script component XML files do not include such a reference, so the value for this attribute is always "yes."

encname
: (Optional) A string that describes the character set encoding used by the XML document. The string may include any of the character sets supported by Microsoft® Internet Explorer.

Remarks

This declaration must be the first element (including white space) in the file.

The existence of this declaration puts the script component compiler into strict XML mode where element types and attribute names are case-sensitive, attribute values must be in single or double quotes, and all elements are parsed. If the declaration is not included, the compiler allows looser syntax.

In general, you should include this declaration and follow XML conventions if your script component file will ever be edited in an XML-conformant editor.

Example

```
<?XML version="1.0" standalone="yes" encoding="UTF-16" ?>
```

See Also

Script Component Files and XML Conformance

Windows Script Components

<comment> Element

Marks text in a script component file that should be ignored.

```
<comment>
   text here
</comment>
```

Remarks

Use the <comment> element inside other XML elements to enclose text that is not markup and not data, but that you want to include for other purposes, such as documentation.

Example

The following script component fragment shows a <comment> element inside a <public> element.

```
<public>
   <comment>
```

```
      This implements an Automation component with the
      method "random" and the property "uppercaseName"
   </comment>
   <method name="random" internalName="getRandomNumber"/>
   <property name="uppercaseName">
      <get internalName="get_ucname"/>
      <put internalName="put_ucname"/>
   </property>
</public>
```

See Also

Script Component File Contents

Windows Script Components

createComponent Function

Returns a reference to another script component in the same package (.wsc file).

`value = **createComponent**(*componentID*)`

Values

componentID
    The unique identifier for the script component to create an instance of.

Remarks

By calling the **createComponent** method, you can include the functionality of another script component in the same file. For more details, see Referencing Another Script Component in the Same Package.

Example

The following example shows two script components in the same package. The first script component calls the second script component when its math method is called.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<package>
<component id="component1">
<registration progid="component.FrontEnd"/>
<public>
   <method name="math"/>
</public>
<script language="JScript">
<![CDATA[
function math(){
   return createComponent("component2")
}
]]>
</script>
</component>

<component id="component2">
<registration progid="component.Math"/>
<public>
   <method name="add"/>
</public>
<script language="JScript">
<![CDATA[
function add(n1, n2){
   return n1+n2;
}
]]>
</script>
</component>
</package>
```

After registering this package, you can use it as illustrated in the following commands:

```
set o1 = CreateObject("component.FrontEnd")
Set o2 = o1.math()
msgbox(o2.add(4,5))
```

See Also

Referencing Another Script Component in the Same Package

Windows Script Components

<event> Element

Declares the name of a custom event that can be fired from within the script component.

`<event` `name="`***`name`***`" dispid="`***`dispid`***`"`**`/>`**

Values

name
: The name of an event being exposed.

dispid
: (Optional) A numeric value specifying the dispatch ID for the event, which is compiled into the component's type library and then used by the host application to bind to events. If you do not specify a dispatch ID, it is assigned automatically to the event when the type library is created. Specifying your own dispatch ID allows you to make sure the dispatch ID for an event is always the same, even if you recreate the type library. It also allows you to map your event to a specific dispatch ID, such as those used by convention in COM. For more details, see Exposing Events.

Remarks

To fire an event, use the fireEventmethod.

Example

The following script component fragment defines two events (namechanged and querydone) and shows how to fire one of them (namechanged).

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <property name="name">
      <get/>
      <put/>
   </property>
   <event name="namechanged"/>
   <event name="querydone" dispid="22"/>
<public>

<script language="VBScript">
<![CDATA[
var gName
Sub get_lowercaseName()
   get_lowercaseName = name
End Sub

Sub put_lowercaseName(newLCName)
   gName = newLCName
   fireEvent("namechanged")
End Sub
   ]]>
</script>
```

See Also

<method> Element | <property> Element | Exposing Events | Exposing Methods | Exposing Properties | fireEvent Method

Build: Topic Version 5.6.9309.1546

Windows Script Components

fireEvent Method

Sends notification of a custom event from the script component to the host application.

`fireEvent(`*`eventName`*`[,...]`**`)`**

Values

eventName
The name of the event as defined in the <event> element.

Remarks

You can only fire events if they have been declared in the <public> element.

> **Note** The Behavior handler exposes a **fireEvent** method that is similar to the Automation handler version, but which includes support for event objects. For details, see Exposing Custom Events in Behavior Script Components.

Example

The following shows a fragment of a script component that defines the event namechanged and shows how to fire it.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <property name="name">
      <get/>
      <put/>
   </property>
   <event name="namechanged">
</public>

<script language="VBScript">
<![CDATA[
var name
Sub get_lowercaseName()
   get_lowercaseName = name
End Sub

Sub put_lowercaseName(newLCName)
   name = newLCName
   fireEvent("namechanged")
End Sub
]]>
</script>
```

See Also

<event> Element | Exposing Events | fireEvent (Behavior)

Windows Script Components

getResource Function

Gets the value of a resource defined with the <resource> element.

`value = getResource(resourceID)`

Values

resourceID
A unique identifier for the resource within the script component.

Remarks

The <resource> element allows you to isolate strings or numbers in your script component that you want to intermingle with script in your script component. For example, resource elements are typically used to maintain strings that might be localized into another language. You can use the **getResource** function in the script component's script to extract the contents of a <resource> element.

Example

The following script component fragment defines a resource (called errNonNumeric) and how to use it in script.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <method name="random" internalName="getRandomNumber"/>
</public>
<resource id="errNonNumeric">Non-numeric value passed</resource>

<script language="VBScript">
<![CDATA[
Function getRandomNumber(upperBound)
   If IsNumeric(upperBound) Then
      getRandomNumber = Cint(upperBound * Rnd + 1)
   Else
      getRandomNumber=getResource("errNonNumeric")
   End If
End Function
]]>
</script>
```

See Also

Referencing Other Components

Build: Topic Version 5.6.9309.1546

Windows Script Components

<implements> Element

Specifies additional COM interface handlers for a script component.

```
<implements type="COMHandlerName" [id="internalName"] [default=fAssumed]>
   handler-specific information here
</implements>
```

Values

COMHandlerName

The name of the interface handler to reference. Interface handlers are usually implemented as DLLs, which you must be sure are available and registered in the production environment for your script component. Some handlers, such as the Automation and ASP handlers, are built into the script component run-time (Scrobj.dll). Examples of available interface handlers include:

Interface handler	Description	How implemented
ASP	Allows a script component to get access to the Microsoft Internet Information Services (IIS) Active Server Pages (ASP) object model.	Built into Scrobj.dll
DHTML Behaviors	Allows a behavior script component to communicate with the containing page so it can fire events and access the DHTML object model.	Built into Scrobj.dll

internalName

(Optional) A name you can use to reference the handler in your script. By default, properties, methods, events, and other members of a script component are available in the global namespace. However, if there is a naming conflict between <implements> elements, the names can be disambiguated by prefixing them with the ID of the <implements> element to which they belong, as in the following:

```
<implements type="Behavior" id="sctBehavior">
   [...]
```

```
     </implements>

     [...]

     <script language="JScript">
        // [...]
        sctBehavior.fireEvent("onResultChange",oEvent);
     </script>
```

fAssumed

(Optional) A Boolean flag indicating that the *internalName* is assumed in scripts. The default value for this attribute is true, and members of the object model exposed by the handler are added to the global script namespace and can be accessed unqualified. You only need to include this attribute if you want to set it to false and therefore hide members of a specific <implements> element.

Remarks

Interface handlers extend the script component run-time. An interface handler is a compiled component (generally written in C++) that implements specific COM interfaces.

Script components by default implement the COM Automation interface (specifically, the **IDispatchEx** COM interface). The Automation object's properties, methods, and events are defined in the script component's <public> element. Because the Automation handler is implemented by default, you do not need to implement it with the <implements> element.

Script components can also implement additional COM interfaces by including an <implements> element. Inside the <implements> element, you specify information specific to the interfaces you are implementing. Each interface handler requires different information. For example, a Behavior script component can include <attach> and <layout> elements that are specific to the DHTML Behavior interface.

Example

```
<implements type="Behavior">
   <event name="onResultChange" />
</implements>
```

See Also

How Script Components Work | Script Component File Contents

Build: Topic Version 5.6.9309.1546

Windows Script Components

<method> Element

Declares a method.

```
<method name="methodName" internalName="functionName" dispid=dispID>
   [<parameter name="parameterID"/>]
</method>
```

Values

methodName
: The name of the method to expose.

functionName
: (Optional) The name of the procedure (function or subroutine) in the script component file that implements the method. If you do not specify an internal name, *methodName* is used.

> **Tip** In XML, you can implement elements that have no content (such as the <method> element) by closing the element with />.

dispID
: (Optional) COM dispatch ID for the method. If no dispid is specified, the dispid is automatically generated. If the dispid is set to 0 (zero), the method becomes the script component's default method. For more information about dispids, see Exposing Events.

parameterID
: If a parameter is explicitly declared for the method, identifies the parameter's name.

Remarks

A method is implemented as a procedure (function or subroutine) in a separate <script> element. The <method> element maps the name of the method to the procedure used to implement it.

You can optionally declare the parameters for your method. Doing so is not required, but exposes parameter information if you generate a type library for your script component (see Creating a Script Component Type Library).

Example

The following script component fragment defines two methods (factorial and random). The random method includes definitions for its parameters, and binds to a function called **getRandomNumber**.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <method name="factorial"/>
   <method name="random" internalName="getRandomNumber">
      <parameter name="upperBound">
      <parameter name="seed">
   </method>
</public>

<script language="VBScript">
<![CDATA[
   Function factorial(n)
      If isNumeric(n) Then
         If n <= 1 Then
            factorial = 1
         Else
            factorial = n*factorial(n-1)
         End If
      Else
         factorial = -2    ' Error code.
      End If
   End Function

   Function getRandomNumber(upperBound, seed)
      upperBound = CInt(upperBound)
      Randomize
      getRandomNumber = Cint(upperBound * Rnd(seed) + 1)
   End Function
]]>
```

```
</script>
```

See Also

<event> Element | <property> Element | Exposing Events | Exposing Methods | Exposing Properties

---

Build: Topic Version 5.6.9309.1546

Windows Script Components

<object> Element

Defines objects that can be referenced by script.

```
<object id="objID" [classid="clsid:GUID" | progid="progID"] events="true|false"/>
```

Values

objID

A name by which the object can be referenced in your script. Object ID values must begin with a letter and can include letters, digits, and underscores (_). The object ID must be unique throughout the scope of the script component. For example, if you specify the name CObj, you can reference the object in your script this way:

```
x = CObj.Prop1
```

GUID

(Optional) A reference to the class ID (GUID) under which the object has been registered. Use "clsid:" followed by the class ID (without curly brackets). Either a classid or a progid attribute must be specified. For example:

```
classid="clsid:2154c700-9253-11d1-a3ac-0aa0044eb5f"
```

progID
: (Optional) The program ID of the object, which can be specified as an alternative to the class ID. Either a classid or a progid attribute must be specified.

events
: (Optional) An attribute that allows you to hook events from the object. By default, *events* is false. If the attribute is true, you can connect to any events the object may fire. You must add an event handler for each event that is to be handled.

Remarks

The <object> element provides a way to expose objects globally for use in scripting within the script component without having to use a function such as **CreateObject()**. Using an <object> element makes the object available with global scope, and allows scripting tools to provide statement completion for the object's members.

Example

The following script component fragment includes an <object> element to create an object reference to the ADODB.Connection object.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<registration progid="ADOScriptlet"/>
<object id="cnn" progid="ADODB.Connection"/>
<public>
   <property name="cnnState"/>
   <method name="openconnection"/>
</public>

<script language="VBScript">
<![CDATA[
Dim cnnState
Function openconnection()
   cnn.ConnectionString =
      "driver={SQL Server};server=myserver;uid=sa;database=pubs"
   cnn.Open
   If cnn.State = 1 Then
      cnnState = "open"
      cnn.Close
   Else
      cnnState = "closed"
   End If
End Function
]]>
</script>
```

See Also

Script Component File Contents

Build: Topic Version 5.6.9309.1546

Windows Script Components

<package> Element

Encloses multiple script component definitions.

```
<package>
   one or more script components here
</package>
```

Remarks

The <package> element is optional when a .wsc file contains only one script component definition.

Example

```
<package>
<component id="MyScriptlet">
   (script component information here)
</component>

<component id="MyOtherScriptlet">
```

```
   (script component information here)
</component>
</package>
```

See Also

Script Component File Contents

Windows Script Components

<property> Element

Declares a property.

```
<property name="propertyName" [internalName="propertyVariable"] />
```

–or–

```
<property name="propertyName">
   <get [internalName="getFunctionName"] />
   <put [internalName="putFunctionName"] />
</property>
```

Values

propertyName
: The name of the property to expose. Unless you specify an internalName attribute, this name must match the name of a global variable that will be used to hold the property's value.

propertyVariable
: (Optional) The name of the global variable in the scriptlet file's <script> element that will hold the value for *propertyName*. If you do

not include the internalName attribute, the property value is maintained in a variable named the same as *propertyName*. Specifying the internalName attribute allows you to use different names for the property and its corresponding global variable.

getFunctionName
: (Optional) The name of a procedure that is used to read the value of the property. If you include <get> element but no <put> element, the property will be read-only. If you include a <get> element but do not specify an internalName attribute, the method used to read the property value must have the name of the property plus the get_ prefix (for example, get_lastname).

putFunctionName
: (Optional) The name of a procedure that is used to write the value of the property. If you include a <put> element but no <get> element, the property will be write-only. If you include a <put> element but do not specify an internalName attribute, the method used to read the property value must have the name of the property plus the put_ prefix (for example, put_lastname).

> **Tip** In XML, you can implement elements that have no content (such as the <property> element) by closing the element with />.

Remarks

Properties can be exposed as simple values. In that case, the property is treated as a global variable inside the script component file.

You can also implement properties as procedures (functions or subroutines), which allows you to calculate a property value and to control whether a property is read-only, read-write, or write-only. In this technique, properties are implemented as a procedure (function or subroutine) in a separate <script> element. The <property> element maps the name of the property to the procedures used to implement it. The names of the procedures must match the internal names you specified in the <property> element.

When *putFunctionName* is called, it is passed one argument that contains the value to which to set the property.

In addition to the standard syntax shown above, you can use a shorthand notation to specify information what would otherwise be added using child tags. For example, if you want to declare a property with a "get" and "put" accessor of the same name as the property, you can use the following syntax:

```
<property name="myproperty" get put/>
```

which is functionally equivalent to:

```
<property name="myproperty">
   <get/>
   <put/>
</property>
```

If you wanted to explicitly name those accessors differently from the default, you can use the following syntax:

```
<property name="myproperty" get="testget" put="testput"/>
```

To specify a default property, include the attribute dispid="0" in the <property> element. For details, see Exposing Properties.

Example

The following script component fragment shows the definition for four properties (sname, age, dateOfBirth, and mstatus). The sname property is a simple value. The age property is read-only, and is implemented with the function readAge. The dateOfBirth property is read-write, and is implemented wih the functions readDOB and writeDOB. Finally, the mstatus property is implemented with the default functions get_mstatus and put_mstatus.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <property name="sname"/>
   <property name="age">
      <get internalName="readAge"/>
   </property>
   <property name="dateOfBirth">
      <get internalName="readDOB"/>
      <put internalName="writeDOB"/>
   </property>
   <property name="mstatus">
      <get/>
      <put/>
   </property>
</public>

<script language="VBScript">
<![CDATA[
Dim sname   ' Read-write sname property (no functions).
Dim gDOB   ' Global variable used to store date of birth.
Dim gMStatus   ' global variable used to store marital status.

Function readDOB()
   ' Gets value of dateOfBirth property.
   readDOB = gDOB
End Function
Function writeDOB(newDOB)
   ' Sets value of dateOfBirth property.
   If isDate(gDOB) Then
      ' Error checking.
      gDOB = newDOB
   End If
End Function

Function readAge()
   ' Calculates read-only age property.
   If isDate(gDOB) Then
      dobM = DatePart("m", gDOB)
      dobD = DatePart("d", gDOB)
      dobY = DatePart("yyyy", gDOB)
      todayY = DatePart("yyyy", Date)
      age = todayY - dobY

      ' Adjust if birthday has not yet occurred this year.
      bday = DateValue(dobM & "/" & dobD & "/" & todayY)
      If DateDiff("d", bday, DateValue(Date)) < 0 Then
         age = age - 1
      End If
      readAge = age
   End If
End Function

Function get_mstatus()
   ' Reads value of mstatus property.
   get_mstatus = gMStatus
End Function

Function put_mstatus(s)
   ' Writes value of mstatus property.
   If s = "M" Or s = "S" Then
      gMStatus = s
   Else
      gMStatus = "?"
   End If
End Function
]]>
</script>
```

See Also

<event> Element | <method> Element | Exposing Events | Exposing Methods | Exposing Properties

---

Windows Script Components

<public> Element

Encloses the script component's property, method, and event declarations.

```
<public>
   <property name="pname"/>
   <method name="mname"/>
   <event name="ename"/>
</public>
```

See Also

<event> Element | <method> Element | <property> Element | Exposing Events | Exposing Methods | Exposing Properties

---

Windows Script Components

<reference> Element

Includes a reference to an external type library.

```
<reference [object="progID" |guid="typelibGUID"] [version="version"]>
```

Values

progID

The program ID from which the type library can be derived, which can include a version number (for example, ADO.Recordset.2.0). This can include the explicit program ID of a type library, or the program ID of the executable (such as a .DLL) that incorporates the type library. If you use the object attribute, you do not need to specify a version attribute, because the version can be inferred from the program ID.

If the object attribute is specified, you cannot also specify a guid attribute.

typelibGUID

The GUID of the type library to reference. If the guid attribute is specified, you cannot also specify an object attribute.

version

(Optional) The version number of the type library to use. It must be in the form <major version>[.<minor version>]. If a version is not specified, the default version is 1.0. If the object attribute is used to specify the type library and the version is not specified, the version is derived from the Registry key for the specified program ID. If none can be found, the default is 1.0.

Remarks

Referencing a type library in your script component allows you to use constants defined in the type library in scripts. The <reference> element looks up and makes available the type library associated with a specific program ID or type library name. Type library information can be available in .tlb, .olb, or .dll files.

The <reference> element should appear inside the <component> element. If there is more than one script component in the package, the type library applies to only the script component in whose <component> element it is declared.

Example

In the following script component fragment, referencing the type library for ADO (contained in the file MSAD015.DLL) allows you to use ADO constants such as adStateOpen in your scripts.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<reference object="ADODB.Connection.2.0"/>
<registration progid="ADOScriptlet"/>
<public>
   <property name="cnnstate"/>
   <method name="openConnection"/>
   <method name="closeConnection"/>
</public>

<script language="VBScript">
<![CDATA[
Dim cnn
Dim cnnState
Function openConnection()
   Set cnn = CreateObject("ADODB.Connection")
   cnn.ConnectionString =
      "driver={SQL Server};server=myserver;uid=sa;database=pubs"
   cnn.Open
   If cnn.State = adStateOpen Then
      cnnState = "open"
   Else
      cnnState = "closed"
   End If
End Function

Function closeConnection()
   cnn.Close
   cnnState = "closed"
End Function
]]>
</script>
```

See Also

Referencing Other Components

Windows Script Components

<registration> Element

Defines information used to register the script component as a COM component.

```
<registration progid="progID" classid="GUID" description="description"
   version="version" [remotable=remoteFlag]/>
```

–or–

```
<registration progid="progID" classid="GUID" description="description"
      version="version" [remotable=remoteFlag]>
   <script>
      (registration and unregistration script)
   </script>
</registration>
```

Values

progID
: (Optional) A text name that programmers use to reference your script component when creating an instance of it. For example, if your script component's program ID is Component.MyComponent, you can create an instance of it in Microsoft® Visual Basic using a statement such as the following:

```
Set Component = CreateObject("Component.MyComponent")
```

> **Note** Although a progid attribute is optional, you must include either a progid or a classid attribute (you can include both). If only the progid attribute is specified, the class ID is generated automatically. If only the class ID is created, then no progid is registered and the object can be created only by referencing the class ID directly.

GUID
: (Optional) A GUID that you have generated using a class ID generation program (such as Uuidgen.exe). If you do not include a class ID, the registration program assigns a class ID to your script component.

description
: (Optional) A text description of the script component that is stored in the registry and that is used in certain tools such as the Visual Basic object browser.

version

(Optional) A numeric version number that you assign. The version is appended to the program ID with a period (for example, MyComponent.1) when applications request a version-specific name. Use only numbers (decimal points are not allowed).

> **Note** The registration attributes can appear in any order in the <registration> element.

remoteFlag

(Optional) A Boolean value indicating whether the script component can be instantiated remotely using DCOM. For details, see Creating Remote Instances of Script Components in the topic "Using a Script Component in an Application."

Remarks

After a script component is created, it can be registered using a program such as Regsvr32.exe, which reads the information in the <registration> element and writes it into the computer's Windows Registry. For example, a script component can be registered this way:

```
regsvr32 file:\\myserver\MyComponent.wsc
```

> **Note** A <registration> element is not required in all cases. For example, a script component that implements the DHTML Behaviors interface handler in Microsoft® Internet Explorer 5.0 does not need to be registered, because Internet Explorer registers the behaviors as they are detected on the page. For details about registration requirements, see the documentation for the interface handler you are implementing and note also which host the script component will be used in.

If you do not include class ID information, the registration program assigns a class ID to your script component at the time it is registered. However, the script component will have a different class ID everywhere it is registered. It is highly recommended that you provide a class ID for the script component yourself, to ensure that your script component has the same class ID on all computers on which it is registered.

Allowing the registration program to create a class ID can cause problems if you use the script component with development tools that store class IDs. If the registration creates a new class ID each time, it will not match the the ID stored by the application.

You can optionally run scripts when a script component is registered and unregistered. To do so, include a <script> element within the <registration> element. To run script during registration, write a register( ) function. To run script when the script component has been unregistered, include an unregister( ) function.

Example

The following shows a typical <registration> element that includes both a prog ID and a class ID.

```
<registration
   progid="Component.TestScript"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}"
   description="My Test Component"
   version="1"/>
```

The following registration element allows the script component to be instantiated via DCOM:

```
<registration>
   progid="Component.TestScript"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}"
   version="1"
   description="My Test Component"
   remotable=true/>
```

The following example shows a <registration> element that includes script to be run when the script component is registered and unregistered.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<registration
   progid="Component.TestScript"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}">
   version="1"
   description="My Test Component">

   <script language="VBScript">
      Function register()
         MsgBox "Component 'My Test Component' registered."
      End Function
      Function unregister()
         MsgBox "Component 'My Test Component' unregistered."
      End Function
   </script>
   ]]>
</registration>
```

See Also

Creating Registration Information | Registering a Script Component

Build: Topic Version 5.6.9309.1546

Windows Script Components

<resource> Element

Isolates textual or numeric data that should not be hard-coded into the script component's scripts.

```
<resource id="resourceID">
   text or number here
</resource>
```

Values

ResourceID
A unique identifier for the resource within the script component.

Remarks

The <resource> element allows you to isolate strings or numbers in your script component that you want to intermingle with script in your script component. For example, resource elements are typically used to maintain strings that might be localized into another language.

To get the value of a resource, call the getResource function, passing it the ID of the resource you want to use.

Example

The following script component fragment defines a resource (called errNonNumeric) and demonstrates how to use it in script.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<public>
   <method name="random" internalName="getRandomNumber"/>
</public>
<resource id="errNonNumeric">
   Non-numeric value passed
</resource>

<script language="VBScript">
<![CDATA[
Function getRandomNumber(upperBound)
   If IsNumeric(upperBound) Then
      getRandomNumber = Cint(upperBound * Rnd + 1)
   Else
      getRandomNumber=getResource("errNonNumeric")
   End If
End Function
]]>
</script>
```

See Also

Referencing Other Components

Windows Script Components

<script> Element

Defines the script component's behavior.

```
<script language="language">
   script here
</script>
```

Values

language

The name of the scripting language used in the script component file, such as Microsoft® Visual Basic® Scripting Edition (VBScript) or JScript.

Remarks

If XML validation is not enabled, the XML parser ignores all lines inside the <script> element. However, if XML validation is enabled by including the <?XML ?> declaration at the top of the script component file, the XML parser can misinterpret greater than (<), less than (>), and other symbols used in script as XML delimiters.

If you are creating a file that conforms closely to XML syntax, you must make sure that characters in your script element are not treated as XML reserved characters. To do so, enclose the actual script in a <![CDATA[...]]> section. For details about XML validation, see Script Component Files and XML Conformance.

Note Do not include a CDATA section unless you also include the <?XML ?> declaration.

Example

```
<?XML version="1.0"?>
<component id="ScriptletFactorial">
<registration progid="Component.Factorial"/>
<public>
   <method name="factorial"/>
</public>

<script language="VBScript">
<![CDATA[
Function factorial(n)
   If isNumeric(n) Then
      If n <= 1 Then
         factorial = 1
      Else
         factorial = n * factorial(n-1)
      End If
   Else
      factorial = -2   ' Error code.
   End If
End Function
]]>
</script>
</component>
```

See Also

Script Component File Contents

Windows Script Components

<component> Element

Encloses an entire Windows Script Component definition.

```
<component id=componentid>
   script component information here
</component>
```

Values

componentid

A string that uniquely identifies the script component. This value is useful if a document contains multiple script components or when you are generating one type library for several script components. Identifiers must begin with a letter and can contain no spaces. The identifier must be unique within a script component package.

If specified, this value functions as the class name for the script component in the host application. For example, if you specify the component ID "MyComponent" in the <component> element, the script component is identified as the class MyComponent in the Visual Basic object browser. If no component ID is specified, the default value is ComponentCoClass.

Remarks

In script component files, an entire script component definition — including<registration>, <public> and <implements> elements — must appear inside a <component> element. If the file contains multiple script components, they must in turn be enclosed within a <package> element.

Example

The following shows a simple but complete script component that includes a factorial method and a name property.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<?XML version="1.0"?>
<component>
<registration>
   description="My Test Component"
   progid="Component.TestScript"
   version="1"
   classid="{2154c700-9253-11d1-a3ac-0aa0044eb5f}"
</registration>

<public>
   <property name="name"/>
   <method name="factorial"/>
</public>

<script language="VBScript">
   <![CDATA[
   Function factorial(n)
      If isNumeric(n) Then
         If n <= 1 Then
            factorial = 1
         Else
            factorial = n*factorial(n-1)
         End If
      Else
         factorial = -2   ' Error code.
      End If
   End Function
   ]]>
```

```
</script>
</component>
```

See Also

Script Component File Contents

Windows Script Components

Behavior Handler Reference

With the Behavior handler, Microsoft® Internet Explorer provides a means for a Behavior script component to communicate back to its containing page through custom events and to access the containing document's object model.

The following elements, properties, and methods are specific to the Behavior Handler.

- <attach> Element
- <layout> Element
- element Property
- attachNotification Method
- createEventObject Method
- fireEvent Method

Windows Script Components

<attach> Element

Binds an event from the containing document to a function in the script component.

```
<attach event="eventName" handler="functionName" [for="elementName"]/>
```

Values

eventName
: The event being bound, such as onmouseclick.

functionName
: The name of the procedure (function or subroutine) in the script component file that is executed in response to the event. If this attribute is omitted, it is generated.

If the `for` attribute is not specified, the default value of the handler attribute is the value of the event attribute. If the `for` attribute is specified, the default handler attribute value is the string generated by concatenating the `for` attribute value, "_", and the event attribute value.

elementName
: The element associated with the event, used for containing elements to which DHTML Behaviors are not explicitly attached. The only possible values for the `for` attribute are "document," "window," and "element." If the `for` attribute is not included, "element" is the default and the event is assumed to be fired on the element to which the behavior is attached.

Example

In the following example, the <attach> element binds three events to functions. For example, the DHTML onmouseover element is bound to the script component's do_onmouseover function. The functions bound to the DHTML onmouseover and onmouseout events are executed only if they are fired on the element in the containing document to which the behavior is attached. The docinit function is explicitly bound to the DHTML document object's onload event.

Note A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<?XML version="1.0"?>
```

```
<component id="bvrscript component1">
<registration progID="Behaviorscript component"/>
<implements type="Behavior">
   <attach event="onmouseover" handler="do_onmouseover"/>
   <attach event="onmouseout "handler="do_onmouseout"/>
   <attach for="window" event="onload" handler="docinit"/>
</implements>
<script language="JScript">
<![CDATA[
var normalColor, normalSpacing;
function do_onmouseover(){
   // Save original values.
   normalColor = style.color;
   normalSpacing= style.letterSpacing;
   style.color = "red";
   style.letterSpacing = 2;
}
function do_onmouseout(){
   // Reset to original values.
   style.color = normalColor;
   style.letterSpacing = normalSpacing;
}
function docinit(){
   document.linkColor = "red";
}
]]>
</script>
</component>
```

See Also

Creating a Behavior Script Component

Windows Script Components

attachNotification Method

Binds a function in the script component to a notification message sent from the host.

`behavior.attachNotification (fpNotify)`

Values

behavior
The ID of the <implements> element used to implement the Behavior interface.

> **Note** By default, the properties and methods exposed by the Behavior handler are automatically added to the global script namespace and can be accessed without referencing the Behavior handler ID. In this case, instead of using *Behavior*.attachNotification as shown in the syntax, the method can be used in script simply as attachNotification. For more details, see the <implements> element.

fpNotify
The name of the function to bind to the notification.

Remarks

Currently, the host can call the specified function with the following notifications:

- **"contentReady"** There has been a change to the content of the element to which the behavior is attached. This is fired as soon as the closing tag of the element is parsed, as well as when a script sets the DHTML innerHTML property of the element. A behavior can use this notification to retrieve the content of the element it is applied to.
- **"documentReady"** The browser has finished parsing the document. Any modifications or initializations that need to be made to the content can take place at this point.

> **Note** The attachNotification method does not notify the script component of standard events that occur in the document, window, or any element on the page. The recommended way to receive notifications on this type of event is by using the DHTML achEvent method.

When dealing with changes to an element's DHTML style property, such as setting its visibility, changing colors, or changing fonts, it is recommended that the changes be made inline in the script component's <script> element, as shown in the following script component fragment. Making the change in the documentReady notification clause can cause slight flashing.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<implements type="Behavior"/>
<script language="JScript">
<![CDATA[
   style.color = "green";
   style.letterSpacing = 5;
]]>
</script>
```

Example

The example below demonstrates how a function could be set up to trap notifications and process them as appropriate.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<?XML version="1.0"?>
<component>
<implements type="Behavior">
   <event name="onResultChange"/>
</implements>

<script language="JScript">
<![CDATA[
   attachNotification (onNotification);
   function onNotification (sNotification){
      if (sNotification == "contentReady"){
         // Contents of element have changed.
      }
      else if (sNotification == "documentReady"){
         // Host has finished parsing element.
      }
      window.status = sNotification;
   }
]]>
</script>
</component>
```

See Also

<event> Element

Build: Topic Version 5.6.9309.1546

Windows Script Components

createEventObject Method

Creates a DHTML event object to be used in passing event context information to the containing document when the fireEventmethod is called.

```
oEvent = behavior.createEventObject()
```

Values

behavior
The ID of the <implements> element used to implement the Behavior interface.

> **Note** By default, the createEventObject method has global scope and can be accessed without referencing the Behavior handler ID. For example, instead of using *Behavior*.createEventObject as shown in the syntax, you can simply call the function as createEventObject. For more details, see the <implements> element.

oEvent
An event object created by the method.

Remarks

The same event object cannot be reused in multiple fireEvent calls.

Example

The following partial script component is derived from a hypothetical calculator script component. Whenever the result changes, the script component fires a custom onResultChange event back to the page, passing the result as an expando property of the event object.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<component>
<implements type="Behavior">
   <event name="onResultChange"/>
</implements>

<script language="JScript">
<![CDATA[
attachEvent("onclick", doCalc);

function doCalc()
{
   // Code here to perform calculation. Results are placed into
   // the variable sResult.
   oEvent = createEventObject();
   oEvent.result = sResult;
   fireEvent("onResultChange",oEvent);
}
]]>
</script>
</component>
```

The following shows what the containing DHTML page might look like. When the onResultChange event fires, the results of the calculation are extracted from expando property `result` of the DHTML window.event object and displayed in the resultWindow <DIV> element.

```
<HTML>
<HEAD>
<xml:namespace prefix="LK" />
<style>
   LK\:CALC {behavior:url(engine.wsc)}
</style>
<script language="JScript">
function showResults(){
   resultWindow.innerText=window.event.result;
}
</script>
</HEAD>
```

```
<LK:CALC id="myCalc" onResultChange="showResults()">
<TABLE>
<TR>
   <DIV ID=resultWindow
      STYLE="border: '.025cm solid gray'"
         ALIGN=RIGHT>0.</DIV>
</TR>
<TR>
   <TD><INPUT TYPE=BUTTON VALUE=" 0 "></TD>
   <TD><INPUT TYPE=BUTTON VALUE="+/-"></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" . "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" + "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" = "></TD>
<TR>
</TABLE>
</LK:CALC>
</HTML>
```

See Also

Exposing Custom Events in Behavior Script Components

Windows Script Components

element Property

Returns the element to which the behavior is being applied.

```
[oElement = ] Behavior.element
```

Values

oElement
    Element to which the behavior is being applied.
Behavior
    The ID of the <implements> element used to implement the Behavior interface.

> **Note** By default, the properties and methods exposed by the Behavior handler are automatically added to the global script namespace and can be accessed without referencing the Behavior handler ID. In this case, instead of using *Behavior*.element as shown in the syntax, the property can be used in script simply as `element`. For more details, see the <implements> element.

Remarks

The property is read-only.

With this property, a behavior is able to communicate with the containing document. All properties, methods, and events exposed by the DHTML object model are accessible through this property.

Example

The following script component fragment implements an expanding and collapsing table of contents using a script component. The script component attaches to the element's DHTML onmouseover event. It sets the DHTML **cursor** property of the element to "hand" to signal the user that the element can be clicked in order to toggle visibility of its children.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, seeScript Component Files and XML Conformance.

```
<public>
   <attach event="onmouseover" handler="event_onmouseover");
</public>

<implements type="Behavior"/>
<script language="JScript">
<![CDATA[
   function event_onmouseover()
   {
      oElement = window.event.srcElement;
      if (oElement == element)
         oElement.style.cursor = "hand";
   }
]]>
</script>
```

See Also

For general information about applying behaviors to DHTML elements, see the "Using DHTML Behaviors" topic on the Site Builder Network (SBN)

Windows Script Components

<layout> Element

Defines HTML text inserted into the containing document.

```
<layout>
   <HTMLtag>Inserted text</HTMLtag>
</layout>
```

Values

HTMLTag
The name of HTML tag for which the element's text is a replacement.

Remarks

When the behavior is called, the text in the <layout> element is read into the corresponding element in the containing document. This behavior is equivalent to setting the element's DHTML innerHTML property.

If the script component is not defined to be XML-conformant, the contents of the <layout> element are opaque. However, if the script component is XML-conformant, the contents of the <layout> element must explicitly be made opaque by including them in a CDATA. For more details, see Script Component Files and XML Conformance.

Example

The following shows an example of a script component with a <layout> element. Because the script component contains the <?XML ?> declaration at the top, the <layout> element contains a CDATA section to make the contents of the <layout> element opaque to the XML parser.

```
<?XML version="1.0" ?>
<component id="bvrScriptlet1">
<registration progID="BehaviorScriptlet"/>
<implements type="Behavior">
   <layout>
   <![CDATA[
      <h1>This is the HTML to show in the element</h1>
   ]]>
   </layout>
</implements>
</component>
```

See Also

For general information about applying behaviors to DHTML elements, see the "Using DHTML Behaviors" topic on the Site Builder Network (SBN).

Windows Script Components

fireEvent method

Fires a custom event.

```
Behavior.fireEvent(sEvent[, oEvent])
```

Values

Behavior
The ID of the <implements> element used to implement the Behavior interface.

> **Note** By default, the properties and methods exposed by the Behavior handler are automatically added to the global script namespace and can be accessed without referencing the Behavior handler ID. In this case, instead of using *Behavior*.fireEvent as shown in the syntax, the property can be used in script simply as fireEvent. For more details, see the <implements> element.

sEvent
The name of the custom event as declared in the <implements> element.

oEvent
(Optional) Specifies an event object containing context information. The event object is created using the createEventObject method.

Remarks

The same event object cannot be reused in multiple calls to the fireEvent method.

Example

The following partial script component is derived from a calculator script component sample. Whenever the result changes, the script component fires a custom onResultChange event back to the page, passing the result as an expando property of the event object.

> **Note** A CDATA section is required to make the script in the <script> element opaque. For details, see Script Component Files and XML Conformance.

```
<component>
<public>
   <event name="onResultChange" />
</public>

<implements type="Behavior">
   <attach event="onclick" handler="doCalc");
```

```
</implements>

<script language="JScript">
<![CDATA[
function doCalc(){
   // Code here to perform calculation. Results are placed into
   // the variable sResult.

   oEvent = createObjectEvent();
   oEvent.result = sResult;
   fireEvent("onResultChange",oEvent);
}
]]>
</script>
</component>
```

The following shows what the containing DHTML page might look like. When the onResultChange event fires, the results of the calculation are extracted from expando property `result` of the window.event object and displayed in the resultWindow <DIV> element.

```
<HTML>
<HEAD>
<xml:namespace prefix="LK" />
<style>
   LK\:CALC {behavior:url(engine.wsc)}
</style>
<script language="JScript">
function showResults(){
   resultWindow.innerText=window.event.result;
}
</script>
</HEAD>

<LK:CALC id="myCalc" onResultChange="showResults()">
<TABLE>
<TR>
   <DIV ID=resultWindow
      STYLE="border: '.025cm solid gray'"
      ALIGN=RIGHT>0.</DIV>
</TR>
<TR>
   <TD><INPUT TYPE=BUTTON VALUE=" 0 "></TD>
   <TD><INPUT TYPE=BUTTON VALUE="+/-"></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" . "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" + "></TD>
   <TD><INPUT TYPE=BUTTON VALUE=" = "></TD>
<TR>
</TABLE>
</LK:CALC>
</HTML>
```

See Also

Exposing Custom Events in Behavior Script Components | fireEvent