

Advanced Behavior Programming with Internet Explorer 5.x

Dino Esposito, Artis.Net

Introduction

A behavior is a Cascading Style Sheet (CSS) style for IE 5.x that lets you add extra attributes to an HTML tag. Behaviors work in a similar way to subclassing in Win32 windows. They provide a way to customize the look and feel of a certain tag. The code defining a behavior is written following a special XML-based syntax, and which is separated from the HTML page that hosts the tag. Through a behavior, you can hook up and handle DHTML events, and you can make the tag expose new methods and properties.

Behaviors were introduced with IE 5.0 in March 1999. This browser comes with a number of predefined behaviors that look more like extended DHTML features than features capable of extending the functions of certain HTML elements. Among the standard IE 5.x behaviors, in fact, you'll find client-side persistence, Vector Markup Language (VML) support, and download capabilities.

The very first behaviors available in source code were written as Windows Script Components (WSCs). They are recognizable by the file extension ".sct". Today, the HTC (HTML Components) format is generally used to arrange behaviors. HTCs are XML files with a more compact and specialized syntax than WSCs. A WSC, in fact, is the script-based description of a generic COM object that can implement several groups of interfaces, one of which is the interface that provides behavior facilities.

Behaviors are COM objects that you can write with any development tool capable of producing COM objects. Scripting languages, on the top of the WSC architecture, are just one option. Active Template Library (ATL) and C++ provide another.



Dino Esposito is a trainer and consultant based in Rome, Italy. He specializes in Windows and scripting, and has authored *Visual C++ Windows Shell Programming* and *Windows Scripting Host Programmer's Reference*, both from Wrox Press. Dino is also a contributing editor to Microsoft Internet Developer, MSJ and MSDN News. He's the cofounder of www.vb2themax.com, a VB-oriented web site, and works for Artis srl, a Rome-based consulting company active in the system integration and web-based architecture areas.

Using Behaviors

We can assign a behavior to a certain HTML tag through the already familiar CSS syntax. This saves us from getting run-time errors with browsers other than IE 5.x. At the moment, the CSS style behavior is not a standard, and is supported only by IE 5.0 and newer. While defining a CSS style, you can customize the behavior of the target element like this:

```
<style>
  .MyStyle {
    behavior: url(http://expoware/library/MyBehavior.htc)
  }
</style>
```

In the rest of the page, you simply assign the `MyStyle` style to all the elements you like. All the additional methods and properties defined by the `MyBehavior.htc` component will be available through the tag that takes the style.

More tags can be assigned the same behavior. This is a clear step forward for code readability and maintenance, since you can hide lots of script code in a separate file, and don't need to duplicate it for each element with the same behavior.

Behaviors and Custom HTML Tags

Since behaviors tell the browser how to cope with a certain tag, it follows that we can also employ them with custom tags. There's just one caveat to pay attention to – custom tags must be part of a custom namespace.

For example, suppose we want a custom tag called `<box>` that draws a frame around the specified text:

```
<box border="2">Wrox Press</box>
```

This will give an effect like this:



With IE 5.x, we can obtain this effect by defining a behavior capable of drawing a frame around the text. Let's say this is a file called `box.htc`. The behavior's code (`box.htc`) looks like this:

```
<PUBLIC:HTC URN="WroxBox">
<PUBLIC:ATTACH EVENT="ondocumentready" HANDLER="DoInit" />

<PUBLIC:PROPERTY NAME="border" />
<PUBLIC:PROPERTY NAME="backcolor" />

<SCRIPT LANGUAGE="jscript">

function DoInit() {
  SetDefaults();
  BuildTable();
}

function SetDefaults() {

  if (border == null)
    border = 1;
```

```

    if (backcolor == null)
        backcolor = "white";
}

function BuildTable() {
    htmlText = "";
    htmlText += "<table cellpadding=2 cellspacing=0 bordercolor=#000000 ";
    htmlText += "border=" + border + " bgcolor=" + backcolor;
    htmlText += "<tr><td>";
    htmlText += element.innerHTML;
    htmlText += "</td></tr>";
    htmlText += "</table>";

    element.innerHTML = htmlText;
}
</SCRIPT>
</PUBLIC:HTC>

```

This is a very simple behavior that basically waits for the `onreadystatechange` event on the specified element, sets its internal state to default values, and draws its user interface (UI). In this case, the UI consists of a table that contains the element's inner HTML text.

The next step is defining a new namespace with a `<box>` tag. At this point we can associate the `<box>` tag of the given namespace with a behavior. We'll call the namespace "wrox". Our final HTML file will look like this:

```

<HTML xmlns:wrox>
<STYLE>
    @media all {
        wrox\:box {
            behavior:url(box.htc);
        }
    }
</STYLE>

<BODY>
<wrox:box border="2">Wrox Press</wrox:box>
</BODY>
</HTML>

```

There are two aspects of the syntax that we need to consider, because we'll be exploiting them further on: the `<PUBLIC:PROPERTY>` tag and the `element` object.

Any variable that you declare through the `<PUBLIC:PROPERTY>` tag can be set in the source HTML code, just like we set the `border` in our HTML code above:

```

<wrox:box border="2">Wrox Press</wrox:box>

```

The `element` object, instead, refers to the tag with the attached behavior. We used this in our behavior above:

```

htmlText += "<table cellpadding=2 cellspacing=0 bordercolor=#000000 ";
htmlText += "border=" + border + " bgcolor=" + backcolor;
htmlText += "<tr><td>";
htmlText += element.innerHTML;

```

In this case, `element.innerHTML` evaluates to "Wrox Press". The variable `htmlText` contains the HTML to display a table containing this text, with the properties we specify. Assigning

anything to the element's `innerHTML` property corresponds to changing the element's UI, so the following will display the table:

```
element.innerHTML = htmlText;
```

The `onreadystatechange` event indicates when the browser has finished building the DOM for the present tag. This is the right time to change anything before displaying the tag.

Actually, the behavior does the following: it waits for the `onreadystatechange` event, builds the table string, and replaces the current content of the HTML element with it. All this happens before the page gets displayed so the user won't know of the behavior's presence.

Custom HTML tags can be realized through this kind of behavior. However, there are a few drawbacks and potential problems with this implementation that will be fixed by IE 5.5. More on this later, when we discuss enhancements of IE 5.5 and element behaviors.

Scriptlets vs. Behaviors

Both DHTML scriptlets and behaviors are solutions to improve the responsiveness and the capabilities of client-side modules. They are expressions of two completely different approaches to the same problem.

With scriptlets, you import a brand new component into the page. Logically speaking, there's no difference at all between a DHTML scriptlet and an ActiveX control. There are, though, a number of other differences between them from a technical standpoint: scriptlets are interpreted, lightweight, easy to write, and expose their source code. They're fine if you need a custom component – that is, a component with a behavior not provided by any of the standard HTML tags – but not if you just need to slightly enhance a standard tag.

Another point in favor of scriptlets is that they automatically provide scrolling capability, context menu, and selectable text. All these features can be turned off and on at your leisure. In most cases, you would have to code these features yourself in a behavior.

Behaviors, however, are ideal for extending existing tags. To create a custom component with behaviors, say a color picker or a slider control, you need to define a custom tag and assign to it a custom behavior. Here's an example that would produce a vertically sliding slider:

```
<control:slider orientation="vertical">
</control:slider>
```

Defining a custom tag also means that you have to specify the namespace for it. This requires something like this:

```
<html xmlns:control>
```

Finally, you have to specify the behavior file that tells IE 5.0 how to deal with it.

```
<style>
@media all
{
    control\:slider {behavior:url(slider.htc);}
}
</style>
```

If you need a custom component and don't want to create custom tags, you can still use IE 5.0 behaviors, but you will have to work in a very inelegant way. Look at the following example

taken from the Microsoft Platform SDK. It comes from the Internet\Ie\Behaviors\Library\ColorPick directory:

```
<input type="text" class="colorpick">
```

Here, we have defined an <INPUT> tag with a class attribute of "colorpick". This looks like it should produce a textbox. Instead, you get something like this:



A real color picker – clearly not what the code was intended to do - what's going on? The colorpick class is defined in this manner:

```
.colorpick {behavior: url(ColorPick.htc)}
```

The behavior's file simply hides the <INPUT> element and replaces it with a colorful table. Here, the <INPUT> tag was used simply because its programming interface is easily able to return the selected color.

When to Use Which

When should we use scriptlets and when should we use behaviors?

The first issue to consider is the (IE) browser that you expect to find. Behaviors are an exclusive feature of IE 5.x. DHTML scriptlets are also supported by IE 4.0, which gives a much larger potential audience. In those cases where scriptlets offer highly competitive functionality in comparison with behaviors, you would probably be better off using scriptlets.

If your user community uses IE 5.x exclusively, behaviors are certainly better if tag customization is what you really need. If you like the idea of custom tags, then behaviors can be as good as scriptlets when it comes to defining new HTML components.

However, it's not easy to say whether a scriptlet or a behavior is always better than the other. It all depends upon the type and the nature of the component.

For example, if you need a special type of data grid, then go for a scriptlet that can encapsulate better the dynamic construction of the columns, the code that fills them out, and the page scrolling. Your client code just has an <OBJECT> tag with a high-level programming interface to add and manage columns and data.

Instead, if you want to create a sort of tabbed page, behaviors and custom tags are preferable, because it's easy and natural to model the tabs as tags in the page. The following code should be extremely clear and intuitive.

```
<DINO:tab>
  <DINO:tabitem title="one" src="one.htm" />
  <DINO:tabitem title="two" src="two.htm" />
  <DINO:tabitem title="three" src="three.htm" />
  <DINO:tabitem title="four" src="four.htm" />
  <DINO:tabitem title="five" src="five.htm" />
</DINO:tab>
```

So the behavior behind the `DINO:tab` tag can take care of a number of things, including the creation of the physical HTML tags that will realize the new component, and operational stuff such as tab selection, page switching, colors, and so on.

In conclusion to this section, there are no hard and fast rules – you need to try to understand the design pattern behind scriptlets and behaviors before deciding what's better for your specific problem. And never forget that behaviors always require IE 5.0 or higher.

What's New with IE 5.5?

Version 5.5 comes with the following major enhancements:

- Element behaviors
- Windowless frames
- CSS extensions

The concept behind element behaviors builds on what we've already seen of custom tags and behaviors. We will look at element behaviors in more detail now.

Element Behaviors

Element behaviors are a new type of behavior that provides a different method of binding to an HTML element. To distinguish between old-fashioned behaviors and the new type, Microsoft suggests referring the older type to as attached behaviors. So, what's the difference between attached and element behaviors? It's all in the name:

- An **attached behavior** is attached to an element, either through a CSS declaration (the `behavior` property), or programmatically through functions like `addBehavior` and `removeBehavior`.
- An **element behavior**, by contrast, binds to one element and becomes an intrinsic part of it. You can never detach an element behavior. Furthermore, only one element behavior is allowed per element, unlike attached behaviors.

Element behaviors serve the purpose of creating new HTML tags in a safer way than the method we showed earlier. In fact, only custom elements – namely those that have an explicit namespace – can have an element behavior. Using element behaviors to create custom tags has basically two advantages:

- No-one can ever detach the behavior
- Behaviors are synchronously attached to the elements

You can create custom tags in plain HTML, limit to pass parameters using tag attributes, and wait for the `onreadystatechange` event before doing anything in the behavior's body. This always works but forces you to check for the document's readiness. If you want to call a method on a behavior, you can't do that safely unless you wait for the `onreadystatechange` event or check the value of `readyState`.

Look at the following code, for example:

```
<HTML xmlns:wrox>
<STYLE>
  wrox/:hello {behavior: url(hello.htc)}
</STYLE>
```

```
<BODY>
<wrox:hello id="MyHello">

<SCRIPT LANGUAGE=JScript>
  MyHello.doSomething();
</SCRIPT>

</BODY>
</HTML>
```

Here, the `<wrox:hello>` tag gets created with an ID of `MyHello`. The next script line, which calls the `doSomething` method on it, might work intermittently. Whether it works depends on the time it takes to create an instance of the behavior. If the HTC file is cached locally, it should work regularly. The problem is that attached behaviors, by design, are managed asynchronously, because they are applied through CSS and are non-persistent.

The result is that you can use behaviors to create custom HTML tags, but you're somewhat limited in the use of the new tag. Element behaviors have been introduced just to work around this.

Loading Element Behaviors

The key instruction for loading an element behavior is the `<?IMPORT>` directive.

```
<?IMPORT namespace=Wrox implementation="hello.htc">
```

With this line at the top of the HTML page, the parser will load the HTML element referred to synchronously, like any other HTML or XML tag. The behavior code must declare its associated tag through the following line:

```
<PUBLIC:COMPONENT tagName="HELLO">
```

These two new lines transform the clandestine `<HELLO>` tag into a first-class citizen coming from the `wrox` namespace, with the behavior provided by `hello.htc`.

The ViewLink Technology

Behaviors hide their internal details behind properties and methods. However, they somewhat violate the "information hiding" principle, according to which no external component should be able to modify the internal structure of the host if not expressly allowed. What occurs instead is that behaviors actually inject code into the primary DOM of the hosting document. This also means that a behavior's elements inherit existing CSS styles and even that their IDs show up in the DOM. To prevent all this, IE 5.5 introduces the viewlink technology.

This allows you to write behaviors that remain distinct from the hosting page, preventing all possible side effects. Basically, the behavior is seen as a second document to be displayed. You don't have physical code injection in the DOM, but a "link" to another DOM, and the details of the behavior remain hidden. The line that enables this is:

```
<public:defaults ViewLinkContent />
```

Alternatively, you can set it programmatically:

```
defaults.viewLink = document
```

Summary

In this paper we initially looked at what behaviors are, and their evolution from WSC's to HTC's. Next we went onto see how they are used to give us an excellent method of customizing web pages. Scriptlets (a rival to behaviors) were then looked at, a comparison was made, and we also saw how to ascertain when it is appropriate to use each of these two styling tools. Finally, the viewlink technology was covered, along with how it helps to prevent the possible side effects of using behaviors in your web pages!

Further Resources

The April 1999 issues of MIND contained useful articles on behaviors

The March/April issue of MSDN News describes in detail the element behaviors in IE 5.5.

Some useful behavior-related URLs:

Behaviors tutorial by Alex homer:

<http://www.asptoday.com/articles/19990511.htm>

Microsoft behaviors overview:

<http://msdn.microsoft.com/workshop/author/behaviors/overview.asp>

Microsoft DHTML scriptlets:

<http://msdn.microsoft.com/library/officedev/odeopg/deovrdynamichtmldhtmlscriptlets.htm>

Microsoft viewlink overview:

http://msdn.microsoft.com/workshop/author/behaviors/overview/viewlink_oww.asp