10

Embedding Objects

While what we've looked at in HTML so far, such as links, tables, and forms, offers a lot of important functionality, there are still many things that can't be achieved with just HTML alone. What happens if we want a movie embedded in our web page, or an audio MP3 file? In the first days of the web, we could do little more than browse hyper-linked text pages with the very occasional image if we were lucky. With HTML 2.0, we could only insert images, videos, sounds, and text. This improved further with HTML 3.2, but to add anything more, such as simple animated messages, spreadsheets, word documents, or even full 3D renderings of landscapes, we needed to insert them as objects.

HTML 4.01 allows for this by making the `<object>` element the standard way of doing this. Previously, the placement of objects in web pages was supported in a mish-mash of standard and non-standard ways. Indeed, the `<object>` element supercedes the `<applet>`, `<embed>`, and `<img>` elements, as well as Microsoft's `<bgsound>` element and `dynasrc` attribute. The `<object>` element is mainly used for content types the browser can't handle itself, and therefore an `<object>` element needs to provide both the data to be displayed and the plug-in that handles the display of the data. `<object>`, in theory, provides a single way to embed a whole range of objects into web documents, and offer more universal, cross-platform support for HTML pages.

However, a major source of disappointment is that the <object> element, despite being the standard way of doing this, was until recently, only really available in Internet Explorer. This is because the plug-ins needed to support many objects are not often universally cross-platform compatible. Netscape 6 goes part of the way to rectifying this, by adding support. Despite this, though, in the early versions of Netscape 6 (6.0 and 6.01), support for the <object> element when adding anything other than images, was erratic verging on non-existent at times. However, in the more up-to-date Mozilla milestone builds (0.8 and beyond, which form the basis for future Netscape releases), many of these problems have been resolved, and <object> will therefore work properly in the next major upgrade to Netscape 6 expected in summer 2001. Also, other browsers such as Opera 5, and older browsers like Netscape 4 still have limited support for this element and rely entirely on the old <embed> element to enable components.

In this chapter we will be focusing on the <object> element, as the other elements (except for <img>) have either been deprecated, and are covered in Chapter 15, or in the case of the <embed> and <bgsound> elements, were never part of the standard. Both of these elements are covered in Chapter 16.

The <object> Element – History

Originally, the HTML 2.0 Standard provided only one method for incorporating media into HTML documents: the <img> element. This element has certainly proved worthwhile, but it is, of course, restricted to image media, which means that its usefulness is limited as richer media find their way onto the Web.

For other media though, there was the <embed> element, which was introduced in Netscape 2 and IE 3, and is still in many browsers the only way to insert many types of objects into web pages. With the introduction of Java, the <applet> element was introduced in HTML 3.2, to allow developers to insert Java applets.

The <object> element was originally introduced by Microsoft in Internet Explorer 3 for the inclusion of Microsoft **Component Object Model** or **COM** objects (such as ActiveX Controls and a wide variety of different media types and plug-ins). Internet Explorer introduced this tag with support for ActiveX controls, and Microsoft has continued to develop around it.

> *When Microsoft released its first set of Internet tools in March 1996, it announced **ActiveX** technology – which in all truth was just a new marketing name for its existing **OLE** technology. ActiveX (or 3rd generation OLE technology) was a framework that allowed software components to cooperate even if they had been written by different vendors, at different times using different tools and different languages, and if the objects were located in the same process – same machine or distributed over multiple machines. Some ActiveX controls came with the IE browser.*

The reason the <object> element has proved so popular with the HTML W3C standards body, and was ultimately adopted by it, was that it introduces a uniform way to embed all kinds of object – images, Java applets, movies, etc. We can't use the <embed> tag to place images or Java applets, and we can't use the <img> element for anything other than images, yet the logic behind introducing this content is very similar for each type of object. For each object, regardless of whether it's a graphic, a Java applet, or a RealTime movie, we still need to state the location of the object, the type of object we are including, and a place to download a "helper application" so that the browser can still invoke the object, even if it doesn't have native support for it.

A helper application is one that needs to be downloaded by the browser before it can run a particular object. For instance with some browsers need to download Macromedia's Flash before they can run a Flash animation.

Arguably, the `<embed>` element handles all of these features as well and could have been adapted to handle images and Java applets; however, the `<object>` element, together with the `param` attribute provides a cleaner and more general approach to passing parameters to the component, and that may have been what tipped the scales in its favor.

It's an attractive solution, if a somewhat flawed one in some ways. One problem was that Netscape 4 didn't support the `<object>` element – Netscape still preferred to use the `<embed>` element, which was never officially recognized, but as previously mentioned, `<object>` has been introduced in Netscape 6 and Opera 5, albeit to a limited extent. There is also the problem that several components created by Microsoft only ship with Internet Explorer, and are therefore only available with Internet Explorer, such as the graphic effects filters. In general, it is possible to add a lot of other third-party components with `<object>` in Netscape 6, such as images and applets, and it is possible to include images in Opera 5. However, at this precise moment in time, the `<object>` element is not an entirely cross-browser solution.

How the <object> Element Works

Most browsers have built-in native support for common features such as images (GIFs, JPEGs, and PNGs), text, fonts, and colors. When IE 4 was released in 1997, the most common components available on the web were ActiveX controls and Java applets, both of which could be run by the `<object>` element. However, there has been a considerable shift since then. COM objects have replaced ActiveX controls, and these in turn are in the process of being partially superceded by web services in the .NET framework.

The progress with compression algorithms and higher bandwidth being made available more cheaply means that streaming video, such as MPEGs, AVIs, and Quick time Movies, are now much more commonplace on the web. These require separate applications such as the Windows Media Player, or the Apple Quick Time Movie Player to be available to run them. Secondly, audio has moved beyond large, clunky WAV files. Streaming audio is common in the form of `.ra` (real audio files), but by far the most common form of audio file is the MP3 format, which has shrunk the size of audio files to a manageable few megabytes, while retaining a sound quality close to that of a CD player. Once again, we need to insert a third party player such as Windows Media Player into our page to be able to play MP3 files.

> *Technically speaking, most MP3 files are saved at a better sample rate than the average CD player, which saves at 44.1Khz. However, even with files saved at 128KHz, the lossy compression algorithm used in MP3 removes non-audible information to reduce file size. This is still flawed, and you will find CDs audibly superior to MP3 files.*

We can use the `<object>` element to insert these kinds of technologies. However, due to large file sizes, the preferred method for many web users is to download these files separately, and as a result it is becoming less common for web developers to actively embed either video or audio files within a web page. Instead, the most common form of embedded application in a web page is Macromedia's motion web graphics tool – Flash. This allows developers to create animations, interactive graphics, and menus and embed them into web pages. Flash can also handle sound and text, and has become a popular catchall solution for developers. Flash animations are embedded into HTML pages using the `<object>` element.

To be able to embed an object in a page, the W3C states that we need to specify three types of information:

- The implementation of the included object (or the location of the executable code)
- The data to be rendered
- Additional values required by the object at run time

The first two values are specified within the attributes of the `<object>` element, while additional values are specified within the `<param>` element. However, we don't need to specify all three at once. Some objects might not have to be initialized at run time, while others won't have any data to render. When a browser comes across an object it must attempt to render its contents on the web page, otherwise it must attempt to render the contents of the `<object>` element. For example:

```
<object type="oojit/wotsit">
  If you are reading this text then your browser can't render the object.
</object>
```

Here we have placed some alternative text inside a non-existent object. All browsers will display the alternative text, as they can't render the object.

We can also use the `<object>` element to embed objects one inside another. The browser will attempt to the render the first `<object>` – if it can't render that, then it should attempt to render the second `<object>` element, and so on, until it reaches the innermost `<object>` element, if it hasn't yet managed to render any objects. If it can't render any objects, then it will render the text contained within. If we wish, we can supply the object as an `<embed>`, `<applet>`, or `<img>` element within the innermost object instead, for those browsers that don't support the `<object>` element.

It is also possible to insert the `<object>` element within the document header (within the `<head>` element), as long as the `<object>` element isn't intended to render any physical content on the web page. This sounds a little strange, given that the `<object>` element is primarily intended for embedding multimedia content, but it is a useful technique to consider – for example, we could use an `<object>` element in the document header to invoke a database, values out of which could then be manipulated by script in the body.

However, looking at the way in which Netscape 6 and Opera 5.x handle the `<object>` element, it seems we need to ensure ourselves that the element doesn't render, for example by setting it within a style sheet `display: none;`. IE 5.5 on the other hand automatically won't render any `<object>` elements appearing in the `<head>` section.

Now, let's take a look at the `<object>` element's attributes, and see how we can use an object in our web page.

<object> Attributes

The `<object>` element supports the following attributes:

```
archive      border      classid      codebase
codetype     data        declare      height
width        hspace      vspace       name
standby      tabindex    usemap
```

plus the usual HTML 4.01 attributes `id`, `class`, `style`, `dir`, `lang`, and `title`, and the common events (see Chapter 2 for more details on these).

archive

This attribute specifies a list of space-separated URLs, which have information relevant to the resource specified in `classid` and `data` attributes. The syntax is:

```
archive="urllist"
```

border

This attribute specifies the width of the border to be drawn around the object. The syntax is:

```
border="n"
```

where `n` is an integer. For `border="0"`, no border is drawn. Like `align`, this attribute is deprecated in HTML 4.01. Style sheets are the preferred method for adding borders.

classid

This specifies a class identifier for an object (on Windows platforms, this information is stored in the registry on the user's machine, once the object has been installed). The syntax is:

```
classid="class_identifier"
```

The `class_identifier` is the information used to create an object on our web page. The class identifier (the HTML 4.01 standard specifies that this should be a URL) tells the browser to draw an object of a specified type. However, in Internet Explorer it's treated slightly differently; the `classid` attribute is the key to the whole element – this is a value that is unique for every instance of the object, and this is how the browser knows which object to load into the page. Here's an example of the `classid` for the RDS control.

```
<object id="RDS Control" classid="clsid:BD96C556-65A3-11D0-983A-
        00C04FC29E33">
...
```

It is used as an alternative to the `data` attribute, and is often preferred in Internet Explorer to `data`, where it doesn't seem to work as well for the `<object>` element.

codebase

This allows us to specify the URL location and version of the object to be used. The syntax is:

```
codebase="url"
```

However, it's only IE that really implements this attribute, and it implements it in the following way:

```
codebase="url#Version=a,b,c,d"
```

Most components need to be installed on our system before we can use them in our web pages. In IE, the `codebase` attribute points to a location from where the object can be downloaded and installed on our system for use. It also identifies the version of the file that should be downloaded. If an object isn't available already on our local machine then the system will have to go to the `url` specified. Of course, if the site specified in the `codebase` is busy or out of action, then unfortunately our page won't download and insert the requisite object correctly.

As we can see from the syntax, we can optionally append the URL with a version number string. The string has the following format:

- **a** – High-order word of the major version of the component available at the specified URL
- **b** – Low-order word of the major version of the component available at the specified URL
- **c** – High-order word of the minor version of the component available at the specified URL
- **d** – Low-order word of the minor version of the component available at the specified URL

If `a`, `b`, `c` and, `d` are all set to `1` then the component is only downloaded if the release date is later than the installation date on the user's machine.

An example `codebase` to enable us to run a Macromedia Flash 5 animation looks like this:

```
CODEBASE="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.ca
          b#version=5,0,0,0"
```

However, this will only work with IE.

codetype

This specifies the Internet Media Type expected by the browser when downloading an object of the type that has been referenced in the `classid` attribute. It is only relevant if a `classid` attribute has already been specified. The syntax is as follows:

```
codetype="media_type"
```

Browsers may use the value of the `codetype` attribute to skip over unsupported media types, without the need to download unnecessary objects. See "*type*" for more details on media types.

declare

This declares the object without instantiating it. The syntax is simply `declare`.

Use this when you are creating cross-references to objects that occur later in the document, or when you are using the object as a parameter within another object.

data

This defines the URL, or the data itself, that is the source for the object. The syntax is:

```
data="source"
```

where `source` can be a URL from which the data can be downloaded, or the data itself as a string of hexadecimal values. We'll look at an example of this later in the chapter.

height and width

These specify the height and width that an object is to be displayed at. The syntax is:

```
width="n"
height="n"
```

where `n` is the width or height in pixels. This can also be expressed as a percentage value, in which case, the value should be appended with a `%` symbol.

When using the `<object>` element to display images `width` and `height` are scaleable. That is, we can use them to specify the size of box that we want the image to fit into, and the browser will scale the image to suit, which is exactly the same as how the `<img>` element handles images. However, there is an exception to this rule, in that IE 5.5 puts the image in its original size into a frame, and this frame is scaled to the dimensions specified by the `width/height`. We'll see more on this in a minute.

Using `width` and `height` with the `<object>` element helps the page to load faster, because the browser can lay out the rest of the page, as it knows the dimensions of the object before loading it.

hspace and vspace

These attributes are used to control the white space around an object. The syntax is:

```
hspace="n"
vspace="n"
```

where `n` is a numerical value in pixels. Other elements next to, above, and below the image will be moved away by the specified number of pixels. These attributes are now deprecated with the HTML 4.01 standard.

name

This attribute provides a name to refer to the object by. The syntax is:

```
name="name"
```

where `name` is a unique name within the page. In HTML 4.01, this is equivalent to the `id` attribute, but is only required for form submission.

standby

This attribute specifies a text string that will be used while the object data is loading. The syntax is:

```
standby="text"
```

where `text` is a word, phrase or sentence that describes the object, or provides a meaningful description of the object, when displayed while it loads.

tabindex

This is the tab index for the object within the page. The syntax is:

```
tabindex="n"
```

where `n` is the position within the tabbing order of the page. By default, all elements in the page that can receive the input focus are part of the tabbing order, in the order they are defined in the HTML source. Each receives the focus in turn as the *Tab* key is pressed. By setting the value of `tabindex` to `-1`, the element is removed from the tabbing order. See Chapter 6, for more on `tabindex` and focus.

usemap

This attribute indicates that the object is an image map containing defined areas that are individual hyperlinks. The `usemap` attribute is used to specify the map file to use with this object. The standard allows for a complete URL to an external map file, but usually a reference `#mapName` to an inline `mapElement` is used. Also, note that this attribute only exists in Internet Explorer 6, and not earlier IE versions. There is more information on image maps in Chapter 5.

Internet Explorer <object> Extensions

Internet Explorer adds other attributes to the `<object>` element:

```
align        accesskey      alt      code
datafld      hidefocus      notab    type
unselectable
```

Note: only the `align` attribute is actually supported in IE 5.5 onwards, the rest are IE 4 only. Now let's look at these in turn:

align

The syntax for the `align` attribute in this context is slightly different from normal:

```
align="top | middle | bottom | left | right | absbottom* | texttop* | absmiddle* |
baseline*"
```

** For explanations of these values, please refer back to Chapter 5, and look at the section on the `<img>` element. This attribute is deprecated in HTML 4.01.*

accesskey

This attribute defines the "hot-key" that can be used to activate the element, or switch the input focus to it. This is used where a hyperlink takes the form of an image, rather than a text string. For details of using an image as a hyperlink see Chapter 4. For more about the uses of the `accesskey` attribute, see Chapter 6.

alt

This defines a text alternative to the graphic. See Chapter 5 for more details.

code

This attribute defines the URL to the Java class file implementing the object, if this is the object source instead of an image. Its syntax is:

```
code="url"
```

datafld and datasrc

These attributes are used to connect the `<object>` element to a client-side cached data source in Internet Explorer 4, in a technique called **data binding**. These have since been removed in Internet Explorer 6. For a look at using these attributes, see the "`<frame>`" section of Chapter 8.

hidefocus

This attribute holds a value indicating whether or not the object is indicated visibly when the element is in focus. See Chapter 8 for more information.

notab

This attribute was present very briefly in Internet Explorer 3, but was dropped by the time of version 4. It was used to exclude an element from the tabbing order, but now this can be achieved by setting the `tabindex` attribute to –1.

type

This attribute defines the MIME type for the object, as defined in the registry on a Windows Machine. The syntax is:

```
type="mime-type"
```

where `mime-type` is a unique text string of a standard format, which tells the browser what kind of information the file contains, and which application to use to read or execute it – as appropriate. The MIME-types for popular image formats are `"image/gif"`, `"image/jpeg"`, and `"image/png"`. This is overridden by the `classid` attribute.

unselectable

This attribute specifies that the element cannot be selected. The syntax is:

```
unselectable="on | off"
```

If it is set to `off` (the default), then the element can be selected. Setting it to `on` makes the object unselectable.

Tags <object> Can Enclose

The `<object>` element operates in conjunction with the `<param>` element, which is used to specify the different parameters that each object can take. These parameters are values that the object will require at run time. They must be placed at the beginning of the content of the `<object>` element.

The <param> Element

The `param` element can take the following attributes:

```
name      type      value      valuetype
```

plus the universal attributes and core events discussed in Chapter 2. The majority of these additional elements, which are specific to a particular object, will just use a `name` and a `value`.

name

This attribute specifies the `name` or property of the parameter. This attribute is mandatory for every `<param>` element. The syntax is:

```
name="string"
```

value

This attribute specifies the value to set for the parameter. The syntax is:

```
value="string"
```

type

This attribute has the value of the MIME type that is retrieved if `valuetype` (the data type of the `value` attribute) is set to `ref`. The syntax is:

```
type="string"
```

valuetype

This attribute specifies how the parameter value will be obtained. The syntax is:

```
valuetype="data | ref | object"
```

`ref` is via a URL, while `data` is the default, an implicit value, and `object` is an identifier that refers to the `id` of another `object` defined in the page. IE 6 is the first IE version to support `valuetype` for the `<param>` element.

Using the <object> Element

The `<object>` element is a general-purpose element, designed to insert many different types of content into an HTML document. In order to cope with this diversity, the `type` attribute is used to indicate the type of data the object displays, and the `codetype` attribute indicates the type of application that implements the display of the data. Of interest to us when working with images, is the `type` attribute, which is a string description of the content – such as `"image/gif"` for a GIF image, or perhaps `"video/avi"` for an AVI video clip. In general terms for types other than video, audio, or text, the application that is required to display the data is defined by `"application/<document_type>"`. In the case of the generic image types, like GIF and JPEG, the browser itself handles the display of the image.

In Windows, a list of all the MIME types supported by your machine can be found in registry under: `HKEY_CLASSES_ROOT\MIME\Database\ContentType\`, which can be used by all browsers.

Specifying the Data

The `data` attribute provides the data for the object, either as a URL from where it can be downloaded, or inline as a string of values. As an example, this code will display an AVI file named `MyVideo.avi`. If the browser doesn't support the `<object>` element, it will display the text My Video:

```
<object data="http://mysite.com/video/MyVideo.avi"
        type="video/avi">
  My Video
</object>
```

This is a general example of how we can take advantage of the `<object>` element as a containing element.

The `type` attribute, however, is optional; but if it isn't present, then the only way that the browser can be sure of knowing whether it can handle the object is by downloading it first – not all files can be uniquely identified from, say, a file extension. In addition, not all files necessarily have to have the correct extension anyway, and not all systems even use file extensions. By including the `type` attribute, we can tell the browser exactly what type of file it is. Then, if it can't handle it, it won't waste time and bandwidth downloading it.

Fall Back in Browsers That Don't Support an Object

If the browser recognizes the `<object>` element, then all well and good – the object will be invoked. However, if it doesn't recognize the `<object>` element, or can't handle the data that forms the source of the object, we can ensure that an alternative will be displayed.

We can include text and other elements that are only visible on a browser that either doesn't recognize the `<object>` element, or that can't handle the content type of the data it specifies. We do this by placing text or other elements between the opening and closing `<object>` tags, and outside any parameter tags. Here's an example:

```
<object data="http://mysite.com/video/MyVideo.avi"
        type="application/avi">
  <img src="http://mysite.com/stills/MyPicture.gif"
       alt="This is a picture of my dog">
</object>
```

In this case, browsers that support the `<object>` element should display a video clip. However, if the browser either doesn't support the `<object>` element, or can't display AVI files, the viewer will see an ordinary image defined by the `<img>` element. And if the browser doesn't recognize the `<img>` element, or can't display the content of it for any reason, the alternative text in the `<img>` element's `alt` attribute will be displayed.

Inline Data Definitions for Objects

If the data content of the object is reasonably small, it can be defined by including the data itself in the `<object>` tag, this is called an **inline definition:**

```
<object data="data:application/x-oleobject;3300,FF00,E3A0, ...etc... ">
</object>
```

This isn't very common as the amount of data needed for inline definition is usually quite impractical.

Inserting Images with an <object> Element

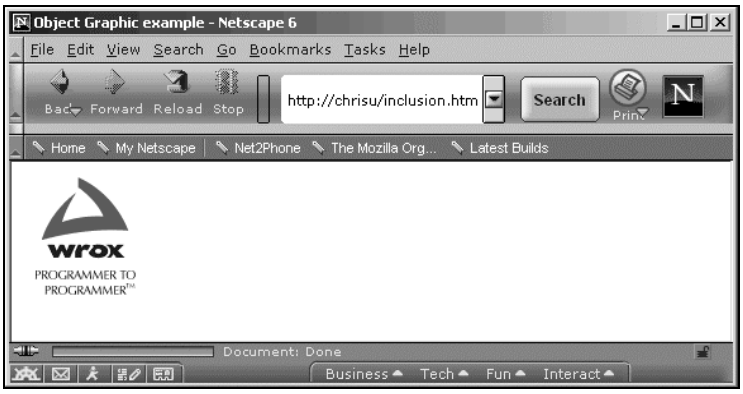
One interesting proposal in HTML 4.01, as we've seen, is the use of the `<object>` element to embed ordinary graphics files, such as GIF and JPEG files. In its most basic form the use is simple:

```
<object data="wroxLogo.gif">
</object>
```

The `data` attribute works just like the `src` attribute in an `<img>` tag, but can also accept inline data where the image is small. This can speed up the time to view of a page, by reducing the number of server connections required:

```
<object type="image/jpeg"
  data="data:image/jpeg;3300,FF00,2756,E5A0,E3A0,22F6, ...etc... ">
</object>
```

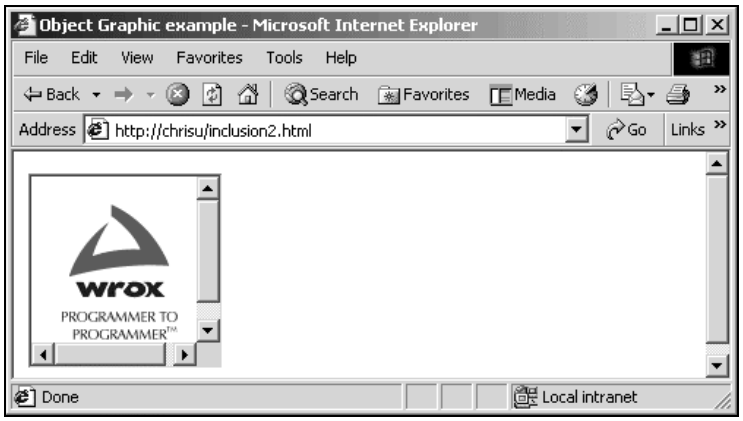
However, unlike the `<img>` element, which will display a graphic in the browser with this minimal information, this isn't enough for Internet Explorer (versions 4 to 6). It will assume that the `height` and `width` settings are set to zero, and we won't see anything, although Netscape 6 and Opera 5.02 will display the graphic quite happily:



By adding the usual `width` and `height` attributes common to the `<img>` element, and adding the MIME type in the `type` attribute, the image will be displayed in Internet Explorer with the following code:

```
<object type="image/gif" data="wroxLogo.gif"
        width="100" height="100" >
  This is our Wrox Logo
</object>
```

While adding the `height` and `width` now means that we can see the image in Internet Explorer, all is still not well:



Looking at the result, we can see an example of a **container** with the `<object>` element. Unlike the `<img>` element, in Internet Explorer (versions 4 to 6) the `<object>` element doesn't use the default size of the file to size the container. The `height` and `width` we've specified are used to size a frame, which contains the `<object>` element itself, but not the image. Because it's larger than the available space, the browser automatically adds scroll bars.

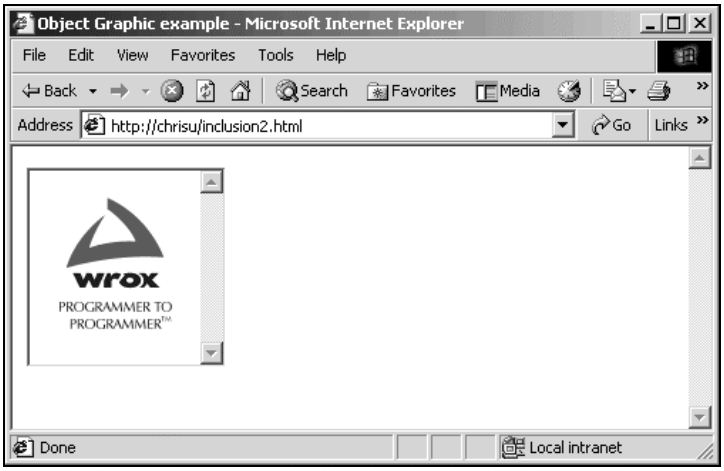
Of course, the "proper" way to size an element in HTML 4.01 is to use style sheets to specify the `height` and `width` styles. Below, we've used the `width` attribute in an inline style sheet for brevity. This code produces the same page as above:

```
<object type="image/gif" data="wroxLogo.gif"
        style="width:100px; height:100px">
  This is our Wrox Logo
</object>
```

However, this doesn't solve the problem of the scroll bars. Resizing the image container so that it fits the image also doesn't solve the problem:

```
<object type="image/gif" data="wroxLogo.gif"
        style="width:105px; height:105px">
  This is our Wrox Logo
</object>
```

Now the horizontal scrollbar is removed, but the vertical one remains, although it is now inactive:

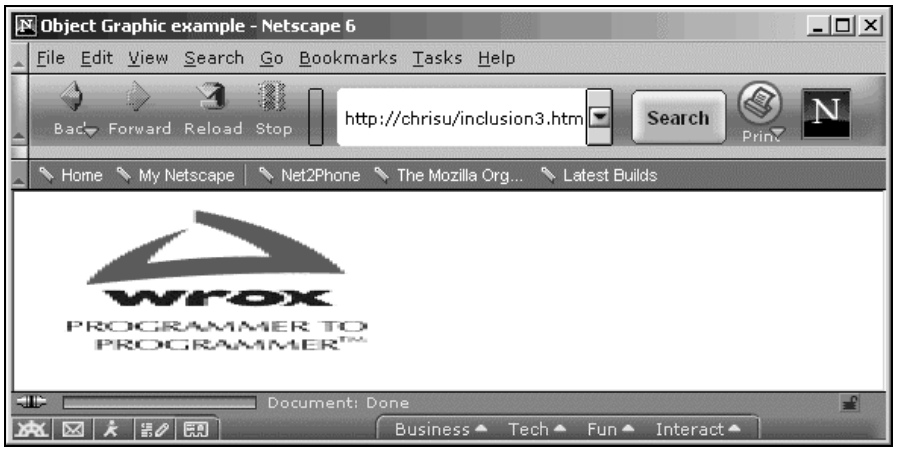


In Internet Explorer therefore, we can't use images inside the `<object>` element without incurring an unsightly scrollbar. This makes it unusable until Microsoft updates this.

Netscape 6 and Mozilla render the image correctly without scrollbars, but there is one anomaly in its representation of images in the `<object>` element. If we resize the image, using a style sheet, Netscape simply sets the image size as the default, ignoring any values set in the style sheet. If we want to stretch the image, as with the `<img>` element, we must use the `height` and `width` attributes:

```
<object type="image/gif" data="wroxLogo.gif"
        height="250" width="100">
  This is our Wrox Logo
</object>
```

This would then stretch the image to fit:



Hidden Image Elements

We can also specify other style properties, for example this code:

```
<object type="image/gif" data="wroxLogo.gif"
        style="visibility:hidden; height:105pt; width:105pt">
  This is our Wrox Logo
</object>
```

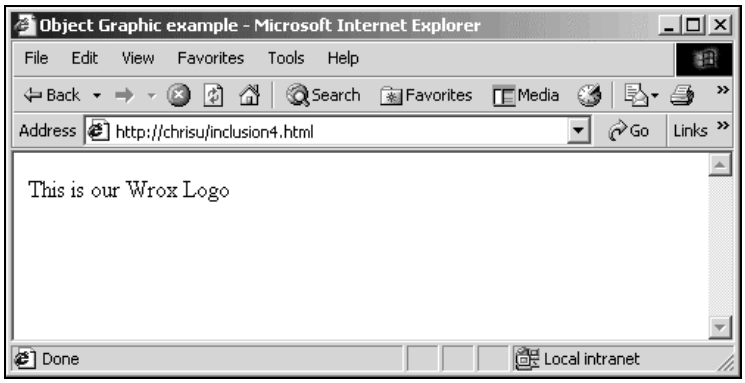
loads the image into the `<object>` element on the page, but makes it invisible. While it may seem a little pointless, it is a good way to get the browser to cache images that we want to make available quickly, such as rollovers – like a rotated ad, or a series of frames of animation in an animated-GIF. When the image needs to be displayed, it doesn't need to be downloaded – just lifted from the local system's cache, although it can slow down the loading of the first page, as all the images have to be loaded first.

The Alternative Content

Finally, what happens if the browser doesn't know how to handle the object specified by the `type` and `data` attributes? Here, we've used an unknown value for `type`:

```
<object type="haven't/aclue" data="wroxLogo.gif"
        style="width:100; height:100">
  This is our Wrox Logo
</object>
```

The result is that, after a few moments' indecision, the browser (IE and Netscape 6 only – Opera 5.x displays the image regardless) reports that it can't handle the file and displays the alternative text content of the `<object>` element:



Unfortunately, Internet Explorer's quirky rendering of images with a permanent vertical scrollbar means that adoption of the `<object>` element in preference to the `<img>` element is not likely to happen yet.

Inserting Components into a Web Page

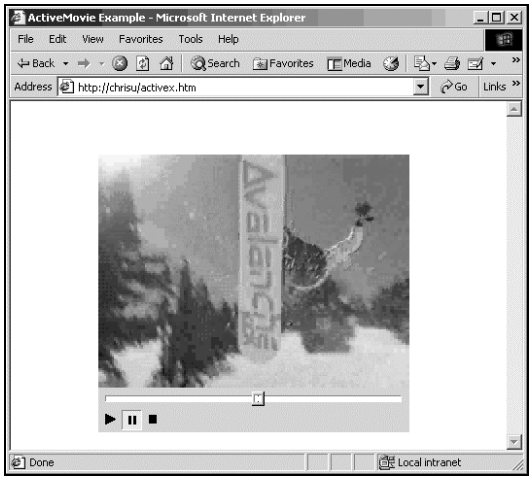
While it isn't possible to look at an example of how to embed every single object or component into web pages, we're going to look at a cross section of components that we can embed into web pages using the `<object>` element.

Showing a Movie

There are several components that are included with each different version of Internet Explorer, which give us the ability to make our web pages come alive by providing special formatting features, animation, video, and much more.

Perhaps the most useful of them is Microsoft's Windows Media Player control, which comes as part of the DirectX SDK and is available in IE 4 and upwards. Windows Media Player allows us to play back popular media formats on the web, including progressive playback of MPEG Audio and Video, AVI files, some types of QuickTime MOVies, MP3, WAV Files, and MIDI. As this component comes with IE, we don't need the `codebase` attribute to locate a version of Windows Media Player.

Here we are going to show how to put in-line video into a web page, by inserting Windows Media Player as an object, like so:



Take a look at the following code:

```
<html>
  <head>
    <title>ActiveMovie Example</title>
  </head>
  <body>
    <object id="WindowsMediaPlayer" classid="CLSID:05589FA1-C356-11CE-BF01-
            00AA0055595A" style="position:absolute;top:55;left:90">
      <param name="ShowDisplay" value="0">
      <param name="ShowControls" value="1">
      <param name="AutoStart" value="1">
      <param name="AutoRewind" value="-1">
      <param name="Volume" value="-5000">
      <param name="FileName" value="http://webdev.wrox.co.uk/books/0707/
            chapter12/RockClmb.avi">
    </object>
  </body>
</html>
```

The versatility of the Windows Media Player Control is that we can just as easily change the code to play an MP3 sound file, by changing the `FileName` parameter value as follows:

```
<param name="FileName" value="http://www.beemen.com/NewBeemenPages/
       TheBeemenGoodnightBirmingham.mp3">
```

This will embedded an MP3 file within the web page.

As can be seen, there are a number of `<param>` tags included within the `<object>` element itself: these are all parameters for the Windows Media Player control. If we wanted to embed a different object in our page, the properties for that object would be different, and we would need to enter those different parameters. We are not actually going to spend any time explaining these properties, but further information on Windows Media Player can be found at: http://msdn.microsoft.com/workshop/imedia/directx/docs/wmp/content_authoring_guide.asp.

Inserting a Java Applet with the Object Element

With the `<applet>` element's deprecation in the HTML 4.01 standard, the way to embed Java applets is with the `<object>` element. It isn't quite as straightforward as all that. Java has a somewhat checkered history with web browsers. With its creation by Sun in the mid-1990s, the main browser vendors of the time (Netscape and Microsoft) were quick to realize its potential. They both quickly included versions of the Java Run-Time Environment within their browsers, which could be made use of by the `<applet>` element. However, Sun updated the Java environment on a regular basis, and so the browsers were left supporting out-of-date versions of the software.

While the creation of Opera didn't make much impact on the two main browsers, it did make the browser agent a much smaller entity, and was the first to consider making the Java Virtual Machine an optional add-on to the main browser. This has some ramifications now for putting Java applets in our web pages, because we still can't assume that every browser will automatically have Java installed (and even if we could some people would disable it anyway). Also, even if they did have it installed, it might not be the most up-to-date version, or a version recent enough to run the applet. It also means if we want to run Java applets on a web browser without native Java support, we must install the Java plug-in first via the `codebase` attribute. However, as downloading any component via the `codebase` attribute is a lengthy process, we suggest that if you really intend using Java (even if you're just browsing Java applets) then you should use a browser that contains the Java Virtual Machine.

Ok, we're going to borrow a very simple example Java Clock applet from the Sun Microsystems site, for which Sun holds the copyright, but the license allows it to be freely distributed. This is located at: http://java.sun.com/products/plugin/1.3/demos/applets/Clock/Clock2.java.

This should be saved as `Clock2.java` on your own machine, after which you need to use the JDK 1.3 to compile it as `Clock2.class`.

You can compile it using the command prompt/UNIX command line using the following code:

```
> javac Clock2.class
```

We're not interested in the internal workings of this code, so we're not going to explain how it works. It is enough to know that it will display a vector graphic clock on our web page. What we are interested in doing is embedding this applet in a web page, within the `<object>` element, so that it works in as many browsers as possible.

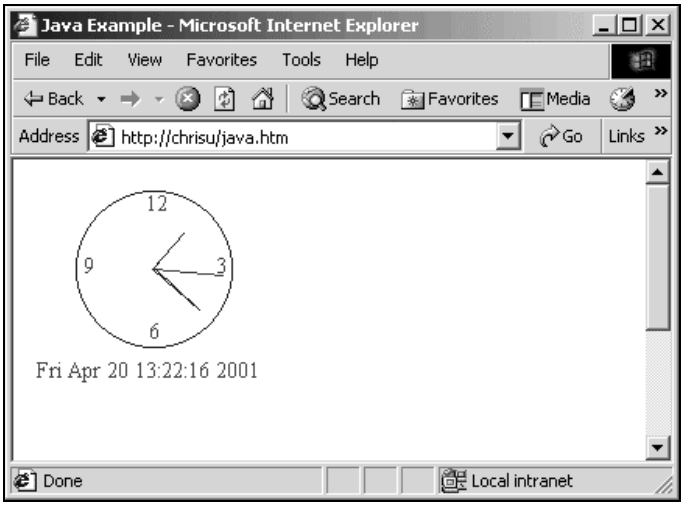
With Internet Explorer it's enough to supply the details in the `classid` attribute:

```
<object codetype="application/java"
        classid="java:Clock2.class" width="400" height="200">
  Your browser is ignoring the "object" element.
</object>
```

However it is possible to specify the latest version of the JDK as follows:

```
<object codetype="application/java;version=1.3"
        classid="java:Clock2.class" width="400" height="200">
  Your browser is ignoring the "object" element.
</object>
```

In which case we are dependent on having a version of IE with a recent enough JDK. In both cases it should result in something along the lines of:



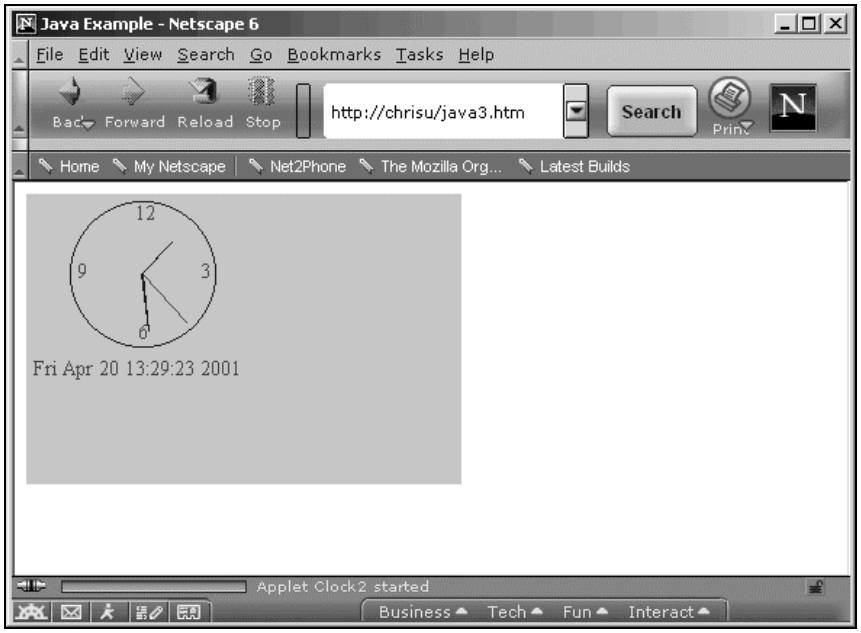
However, for the browsers that don't include native Java support this won't be enough, and we will also need to indicate the location of a Java plug-in to be able to run Java applets. We need to specify the `codebase` as follows – this will be ignored if the browser already has native Java support. In IE versions without the Java SDK the code would be as follows:

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="300"
        height="200" codebase="http://java.sun.com/products/
        plugin/1.3/jinstall-13-win32.cab#Version=1,2,0,0">
  <param name="code" value="Clock2.class">
  <param name="type" value="application/x-java-applet;version=1.3">
  Your browser is ignoring the "object" element.
</object>
```

However, the `.cab` executable specified is something unique to IE, and to get it work in Netscape 6 we actually need to specify the Sun Java plug-in instead:

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="300"
        height="200" codebase="http://java.sun.com/products/plugin/
        1.3/plugin-install.html">
  <param name="code" value="Clock2.class">
  <param name="type" value="application/x-java-applet;version=1.3">
  Your browser is ignoring the "object" element.
</object>
```

Here we've used a version of Netscape 6, without the Java SDK included, and the output will look like this:



Of course this still leaves the problem of what to do with browsers that don't support the `<object>` element entirely, such as Netscape 4. In such cases, we can tuck the `<applet>` tag into the code as follows, making sure to comment it out (for Internet Explorer):

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        WIDTH=300 height=200 codebase="http://java.sun.com/products/
        plugin/1.3/jinstall-13-win32.cab#Version=1,2,0,0">
  <param name="code" value="Clock2.class">
  <param name="type" value="application/x-java-applet;version=1.3">
  <comment>
    <applet code="Clock2.class" width="170" height="150">
    </applet>
  </comment>
</object>
```

The IE-specific `<comment>` tag is the only one available, and if it isn't used, then both the `<object>` and the `<applet>` elements are interpreted by IE, and the applet will be displayed twice. This is direct contradiction to what we said earlier about objects; the expected behavior should be to ignore the `<applet>` element. However, with the `<applet>` element in IE, there seems to be this anomaly, which requires the use of the `<comment>` tag. There is more on the `<applet>` element in the Chapter 15.

Inserting a Flash Animation with the Object Element

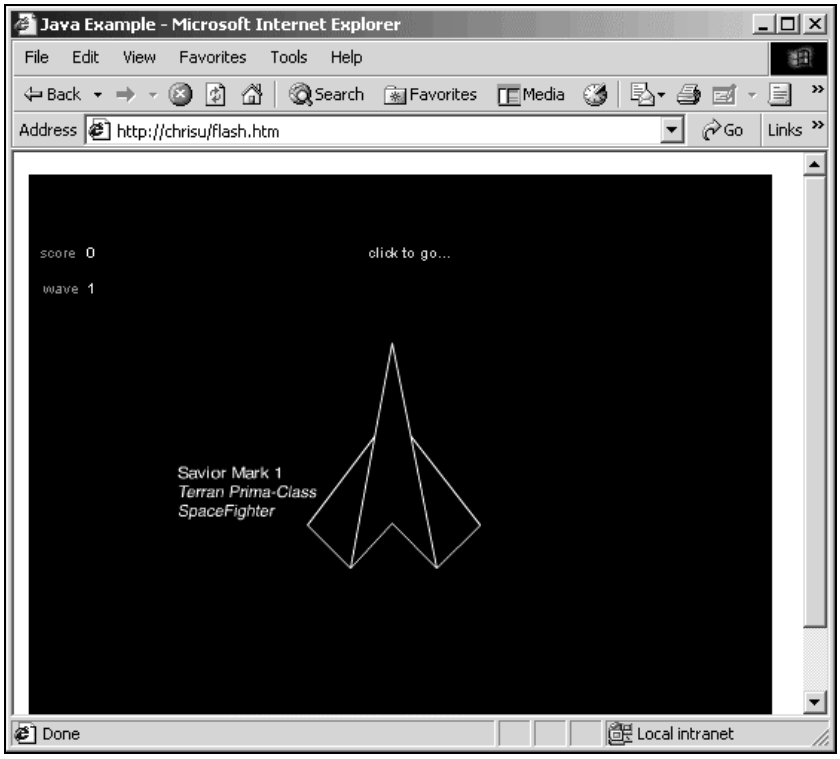
As mentioned earlier in this chapter, the way to insert a Flash animation into our web page is with the `<object>` element. Normally with Flash, we'd use the publishing tool (accessed in Flash via the File | Publish menu) to create the HTML for us, and so we wouldn't have to get involved with actual coding, as the publishing tool does this all automatically for us. However there are times when we might want to manually place a Flash animation into our web pages.

It's also worth mentioning that the code created by the publishing tool uses `<object>` to insert the animation in Internet Explorer, and within the `<object>` element, there is an `<embed>` element to insert the animation so that the Netscape 4 and Opera browsers can run it as well. The Netscape 6 browser seems unable to execute it via the `<object>` element – once again Microsoft `.cab` files are required to run the Flash tool within `<object>`, and so Netscape 6 relies on the `<embed>` element. Now let's look at an example.

We've borrowed a Flash file (`savior.swf` – see the code download) courtesy of Friends of Ed (http://www.friendsofed.com), which allows us to play a simple arcade game. Once again the code that creates the embedded application is not important, just the method by which we are inserting it. Flash is now commonly included in the latest browsers, but to be on the safe side we've added a `codebase` attribute to point to the flash download screen.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/
        cabs/flash/swflash.cab#version=5,0,0,0" width="500"
        height="400">
  <param name="movie" value="savior.swf">
  <param name="quality" value="high">
  <param name="bgcolor" value="#FFFFFF">
  Flash animations not supported with "object"
</object>
```

Assuming the `savior.swf` file is located in the same directory as the web page, which has the above information included, we should see the following in Internet Explorer:



To get this animation to work in Netscape and Opera, we need to enclose the following `<embed>` element code with the `<object>` element code:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/
        cabs/flash/swflash.cab#version=5,0,0,0" width=500
        height=400>
  <param name="movie" value="savior.swf">
  <param name="quality" value="high">
  <param name="bgcolor" value="#000000">
  <embed src="savior.swf" quality="high" bgcolor="#000000" width=550
         height="400" type="application/x-shockwave-flash"
         pluginspace="http://www.macromedia.com/shockwave/download/
         index.cgi?P1_Prod_Version=ShockwaveFlash">
  </embed>
</object>
```

The `<embed>` element is not even deprecated as it never formed part of any HTML standard, although IE, Netscape, and Opera all support it. The `pluginspace` attribute is equivalent to the IE `codebase` attribute. There is more on the `<embed>` element in Chapter 16. One last thing to notice is that we don't require the `<comment>` tag here, as IE will only embed one occurrence of the Flash animation here (as it is meant to), despite the fact that technically there are two specified, because it ignores what is inside an `<object>` element, as it should if it handles the tag itself.

Inserting an Excel Spreadsheet

It is also possible to insert simple items in Internet Explorer, such as spreadsheets and Word documents. We could use a tool such as FrontPage to do the basic work for us, but all FrontPage is actually doing is creating the requisite `<object>` element and inserting the necessary data as `param` attributes. The following code would insert a blank Excel spreadsheet into our web page.

```
<object classid="CLSID:0002E510-0000-0000-C000-000000000046">
  <param name="DisplayTitleBar" value="false">
</object>
```

This unfortunately won't produce anything on Netscape or Opera, as they don't support COM objects.

Including HTML in Another HTML Document

The HTML 4.01 standard also allows us to use the `<object>` element to "insert HTML documents into our web pages". Inserting web pages into web pages might seem like a pointless pastime, but there are two reasons why it might be very useful:

- First, it acts as an alternative to the `<iframe>` element, which is good, given that `<iframe>` isn't officially part of the HTML 4.01 standard. However, there are problems with its use for this purpose. An `<iframe>` element can be targeted by a link or form, and can be focused and printed, while an `<object>` element doesn't support these features.
- Secondly, it enables us to easily reuse a single section of code (either HTML or script) many times throughout a site, without having to reproduce the code in its entirety.

We can embed a document using the `data` attribute as follows:

```
<object data="embedded_document.htm">
  Couldn't include document specified.
</object>
```

However, there has been a security hole associated with this ability in IE 5.5. It allows the execution of arbitrary programs using `object type="text/html"` and parsing `index.dat`, by revealing the location of the temporary internet files folder, which could possibly lead to a hacker taking full control over user's computer. There is a patch available to rectify this. Details of this security issue can be found at http://www.microsoft.com/technet/security/bulletin/MS00-055.asp, and details of the patch at http://www.microsoft.com/windows/ie/download/critical/patch11.htm. The patch basically stops this from working. It also doesn't work in Netscape. The only surefire browser to support this is Opera 5.x.

Microsoft also provides the ability to embed script code within HTML documents as **Behaviors**. Behaviors are platform-independent components that can be deployed on the Web. Formerly their predecessor (known as Scriptlets) used the `<object>` element to achieve this. However, this was very much an evolving technology, and it proved to be a flawed way of inserting them, so Behaviors are now embedded using style sheets instead. As a result this is very much a technology beyond the scope of this book and one we won't discuss any further.

Summary

In this chapter, we've introduced and explained how to add objects to our web pages. We gave a brief overview of the `<object>` element. We introduced the different attributes and showed how it could be used to add images to HTML documents, as well as objects such as videos and MP3 files. This is the method that is recommended by the HTML 4.01 standard, and the method that should be used, if possible.

We rounded off the chapter with an overview of how we can embed some of the most popular components within our web pages. However, we suggested that as Internet Explorer doesn't properly support the `<object>` element for images, it is unlikely to displace the `<img>` element in popular usage, but for all other types of media, it is very useful.