

Internet Explorer 4 JScript and Object Model

Diversity is often a good thing. But when you are trying to develop scripted Web applications that work on all browsers, diversity can become a hurdle to overcome. That's the case when authors learn scripting in one browser environment only to discover too late that some object, property, method, or event is not supported or is supported differently in the other browser.

By including "JavaScript" in the title of this book, I signal that the focus of the book is Netscape's implementation of the language in Navigator. But I cannot ignore Internet Explorer and its JScript language, which is derived from JavaScript. The purpose of this chapter is not to teach you everything you need to know about scripting in JScript for Internet Explorer-only applications. Instead, I want to provide an overview of the pieces in Internet Explorer 4 that differ. Some of what you read about here may eventually come to Navigator in a future version.

Core Language

Internet Explorer's internal architecture clearly distinguishes the core scripting language from the document object model. This is necessary in Internet Explorer because the browser supports two scripting languages: VBScript and JScript. While it is unlikely that a document would contain scripts in both languages, nothing prevents such a situation. Each `<SCRIPT>` tag set can be assigned a different language name in the `LANGUAGE` attribute, and the interpreter of only the named language is invoked for the script.

44

CHAPTER



In This Chapter

Internet Explorer 4's object containment structure

The wide range of Internet Explorer 4 element objects

How to determine the version of JScript installed in the browser



The core language consists of the normal kinds of programming constructs: how variables work; conditional constructions; data types; and similar items that any programming language needs, regardless of the object environment. Microsoft, Netscape, and other participants established a common denominator standard in 1997 under the auspices of the European Computer Manufacturer's Association (ECMA), a Switzerland-based standards body (<http://www.ecma.ch>). As mentioned in earlier chapters, the result is a core language description called ECMAScript.

ECMAScript is based on JavaScript Version 1.1, which was built into Navigator 3. Only a few minor differences exist between ECMAScript and JavaScript 1.1. The JScript implementation in Internet Explorer 4 is compatible with the ECMAScript standard (and with the Navigator JavaScript 1.2 core language, as well). But, because most standards define a common denominator, they don't preclude a supporter of the standard from adding proprietary extensions for a single platform. JScript includes some Internet Explorer 4-only extensions, including access to ActiveX constructs and language version checking.

Document Object Model

Figure 44-1 is a diagram of the document object model and containment hierarchy for Internet Explorer 4. Many elements are the same as Navigator, but I should point out the differences that may lead to confusion in terminology. I recommend that you also keep your finger in Appendix B's Navigator Object Road map to examine compatibility of properties, methods, and event handlers for the objects discussed in this section.

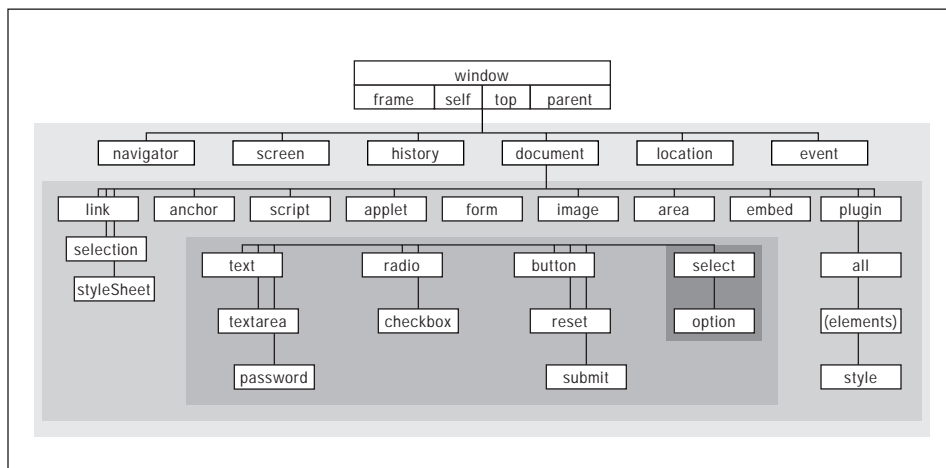


Figure 44-1: Internet Explorer 4 object containment hierarchy

Starting at the top of the hierarchy, Internet Explorer 4 encapsulates the navigator and screen objects inside the window object. Navigator, on the other hand, treats these objects independently of the document object model. Even with this apparent discrepancy, if you reference these objects without the `window` reference (as is often

the case when referring to objects contained by a window), the same statements work in both browsers. Not all properties of these two objects are the same, however.

Diving one level inward to the document object and its components, you may recognize that Internet Explorer 4 has several more objects defined at this level than Navigator 4. The new items consist of the following objects: `all`; `embed`; `plugin`; `script`; `selection`; and `styleSheet`.

You met the `all` keyword in Chapters 41 and 43 in working with style sheets. This keyword is the gateway to objects defined by HTML tags that are not otherwise part of the object hierarchy. The `embed` and `plugin` objects are the same thing: objects reflecting items loaded into the document via the `<EMBED>` tag. Don't confuse this `plugin` object with Navigator's `navigator.plugin` object, which looks into the browser at the plug-ins installed in the browser.

A `script` object reflects the item created by a `<SCRIPT>` tag set. You can literally change the content of a script tag after the page loads. A `select` object is created whenever the user selects text in the body of the document. And a `styleSheet` object is added to a document for each `<STYLE>` tag set or each style sheet imported via the `<LINK>` tag.

Element objects

The most intriguing aspect of the Internet Explorer 4 document object model is that virtually every HTML tag in a document becomes an object, called an *element*. The name of each element is the name of the tag (without angle brackets), but for scripting purposes, you probably want to assign an `ID` attribute to tags you intend to address via scripts. This lets you build a convenient reference to the element through the `document.all` object. For example, you can define a paragraph tag with an `ID` as follows:

```
<P ID="quotation">...</P>
```

The `ID` attribute may be the same as a style defined earlier in the document to be applied to the paragraph block. Either way, you can access the object through a reference that begins

```
document.all.quotation
```

Every element has a number of properties, although most, by virtue of being element objects, have a wide range of properties in common. Among those properties are items such as `offsetLeft`, `offsetTop`, `outerHTML`, `outerText`, and `style`. Elements that are block elements (that is, they have start and end tag pairs) also have properties for `innerHTML` and `innerText`, which allow retrieval and modification of the text inside the block or the entire block itself (including the tags).

Table 44-1 lists all the element object names. Consult the Internet Explorer 4 SDK documentation for details about the properties, methods, and events for each object.

Table 44-1
Internet Explorer 4 Element Objects

A	COLGROUP	H4	MENU	SUB
ACRONYM	COMMENT	H5	META	SUP
ADDRESS	DD	H6	NEXTID	TABLE
APPLET	DEL	HEAD	OBJECT	TBODY
AREA	DFN	HR	OL	TD
B	DIR	HTML	OPTION	TEXTAREA
BASE	DIV	I	P	TFOOT
BASEFONT	DL	IFRAME	PLAINTEXT	TH
BGSOUND	DT	IMG	PRE	THEAD
BIG	EM	INPUT	Q	TITLE
BLOCKQUOTE	EMBED	INS	S	TR
BODY	FIELDSET	KBD	SAMP	TT
BR	FONT	LABEL	SCRIPT	U
BUTTON	FORM	LEGEND	SELECT	UL
CAPTION	FRAME	LI	SMALL	VAR
CENTER	FRAMESET	LINK	SPAN	XMP
CITE	H1	LISTING	STRIKE	
CODE	H2	MAP	STRONG	
COL	H3	MARQUEE	STYLE	

Collections

Microsoft uses a new term to refer to multiple objects of the same kind in a document: the *collection*. In Navigator terms, a collection is the same as an array of similar objects. For example, when a document contains one or more images, you can refer to one of them using array syntax, as follows:

```
document.images[0]
```

In Internet Explorer terminology, the document contains an images collection. Such a collection may consist of zero or more objects of a particular type, and every collection has a length property your scripts can use to determine the number of objects of that type in the document.

While you can refer to these objects in cross-compatible array syntax, Internet Explorer 4 lets you replace the brackets with parentheses. Arguments for the parentheses can be an index value or a name assigned to the object via the object's ID attribute. Therefore, to access the first image of a document, Internet Explorer 4 accepts the following syntax:

```
document.images(0)
```

For named index references, you have your choice of two formats in Internet Explorer 4. One that is compatible with Navigator (all versions) lets you place the name inside brackets, as in

```
document.images["myCat"].src // Navigator and Internet Explorer 4
format
```

In Internet Explorer 4, however, named references may also go inside parentheses, not brackets, as follows:

```
document.image("myCat").src // IE4 format
```

The cross-compatible way is to address objects by numeric index (in array syntax) or by directly naming the object in the reference:

```
document.myCat.src
```

Sometimes, however, a script must assemble the name of an object from other values. For example, the following loops set the URLs of state map image objects. Each image object is named with the state abbreviation plus the word “map.” The statement inside the loop assembles the map name from an entry in a lookup table. Due to differences in bracket and parenthesis syntax for each browser, the assembly must be done differently in Navigator and Internet Explorer 4:

```
// Navigator syntax
for (var i = 0; i < stateDB.length; i++) {
    document.images[stateDB[i].abbrev + "map"].src = stateDB[i].URL
}
// IE4 syntax
for (var i = 0; i < stateDB.length; i++) {
    document.image(stateDB[i].abbrev + "map").src = stateDB[i].URL
}
```

Brackets are okay for numeric index values in both browsers, but named index values must be treated differently. You can, however, script around this incompatibility by having the `eval()` function create an object reference that works with both browsers:

```
// Compatible syntax
var obj
for (var i = 0; i < stateDB.length; i++) {
    obj = eval("document." + stateDB[i].abbrev + "map")
    obj.src = stateDB[i].URL
}
```

It requires a little more effort, but at least a compatible solution exists that works with both browsers — and is even compatible with older versions of all browsers back to Navigator 2 and Internet Explorer 3.

Table 44-2 lists all of the collections in the Internet Explorer 4 document object model. Most collections are properties of the document object. Some, however, are specific to other objects within a document, such as the cells collection, which belongs to a TR (table row) object.

Table 44-2
Internet Explorer 4 Collections

<i>Collection</i>	<i>Description</i>
all	Exposes all element objects in the document
anchors	All document anchors
applets	All document applets
areas	All areas defined for client-side image maps
cells	All cells of a row in a table (collection of a TR object)
children	All direct descendants of an object
elements	All elements in a form (collection of a form object)
embeds	All document embeds
filters	All filters associated with an element
forms	All document forms
frames	All window or document frames
images	All document images
imports	All imported style sheets (collection of a styleSheet object)
links	All document links
options	All options of a select object
plugins	All embeds of a document (same as embeds collection)
rows	All rows of a table (collection of a table object)
rules	All rules in a styleSheet object
scripts	All document scripts
styleSheets	All document style sheets
tbodies	All TBODY elements of a table object

Events

Chapter 39 describes the basic difference between Navigator's object capture and Internet Explorer 4's object bubbling mechanisms. The difference in document object models also plays a role in the event model differences. When you consider that all of those element objects have event handlers associated with them, you see that Internet Explorer tightly binds events to objects. In addition to specifying event handlers within a tag in the traditional way via a tag attribute, Internet Explorer offers another way to bind an event handler to an object. You can create a

<SCRIPT> tag set whose contents are executed when a particular event for a particular object fires. The syntax for this construction is as follows:

```
<SCRIPT FOR=objectReference EVENT="eventHandlerName([params...])"
  LANGUAGE="JavaScript">
    statements to execute (no function name needed)
</SCRIPT>
```

Notice that no function definitions exist in the code within the script. Raw statements are entered inside the tag but they are executed only when the referenced object receives the event named in the `EVENT` attribute. Beware that this construction works only for Internet Explorer 4. If you attempt to include one of these `<SCRIPT>` tags in documents for other browsers, the script statements will execute as the page loads, because the extra attributes are ignored, and the tag is treated like any `<SCRIPT>` tag.

Few system event handlers pass parameters, but some, such as `onError`, do. These parameters are passed along with the event, as if passed with a function call. Given Internet Explorer 4's window-based event object, additional information about the event is available from inside the script by accessing the `window.event` object properties without needing an event object passed, as it is in Navigator 4.

Importantly, the kind of event binding using the special `<SCRIPT>` tag occurs only after the document has loaded. Therefore, if you believe users may try to activate form elements (for example, click a button) before the page has completed loading, then you should define the event handler as an inline event handler inside the tag, even if it must call a function defined earlier in the document.

Internet Explorer 4 also supports setting object event handlers as properties of the object, offering two syntaxes for setting such properties. This is made compatible with Navigator by assigning a function reference to the event property, as in the following example:

```
document.forms[0].clicker.onclick = processClick
```

Internet Explorer also offers a variation on this theme, combining the syntax of object assignment with the syntax of inline event handler assignments, as in the following example:

```
document.forms[0].clicker.onclick = "processClick()"
```

This latter approach requires that the function include its parentheses and that the entire name be in quotes. You can also pass parameters with this method, but if you intend to pass form-related values (for example, the form object or a property of the object), you are better off doing this in an inline event handler, which is also compatible with all browsers.

Scripting Engine Versions

Internet Explorer is structured in such a way that the scripting engine is a separate component of the browser. As happened during the lifetime of Internet Explorer 3 for Windows, the company released two versions of the JScript language. In Windows, the component is a .dll file named `jscript.dll`. Each major

version of the .dll adds to the functionality of the language in the browser, independent of the document object model.

Starting with jscript.dll Version 2, the JScript language comes with its own global functions that reveal the .dll version as well as the default scripting engine. Table 44-3 shows the relevant functions you can use to determine which JScript version is installed in the visitor's browser. Because it is possible that a visitor may be running jscript.dll Version 1, I recommend not using these functions with Internet Explorer 3. An attempt to run these functions with the original .dll file will yield a script error.

Table 44-3
Internet Explorer 4 JScript Versioning Functions

<i>Function</i>	<i>Description</i>
ScriptEngine()	Returns a string of the engine ("JScript")
ScriptEngineBuildVersion()	Integer of the build number of the engine's .dll file
ScriptEngineMajorVersion()	Integer of the value to the left of the decimal of the version number (3 for first release of jscript.dll in Internet Explorer 4)
ScriptEngineMinorVersion()	Integer of the value to the right of the decimal of the version number (0 for the first release of jscript.dll in Internet Explorer 4)

The JScript versioning information is useful only if you are using language constructs that became part of the language after a certain version. For general browser versioning, you should use the navigator object properties, such as `navigator.appVersion` and `navigator.userAgent`.

