33 CHAPTER

XML Objects

In This Chapter

Treating XML elements as objects

Creating IE XML data islands

Accessing XML element attributes

XML (eXtensible Markup Language) is an undeniably hot topic in the Internet world. Not only has the W3C organization formed multiple working groups and recommendations for XML and its offshoots, but the W3C DOM recommendation also has XML in mind when it comes to defining how elements, attributes, and data of any kind — not just the HTML vocabulary — are exposed to browsers as an object model. Most of the arcana of the W3C DOM Core specification — especially the structure based on the node — are in direct response to the XML possibilities of documents that are beginning to travel the Internet.

While XML documents can stand alone as containers of structured data in both IE5+ and NN6, the Windows version of IE5+ permits XML data to be embedded as "islands" in an HTML document. Such islands are encased in an XML element — an IE-specific extension of HTML.

It's important to distinguish between "the" XML element — the element generated in a document by the IE-specific `<XML>` tag set — and a generic XML element that is a part of the XML data island. Generic XML elements have tag names that are meaningful to a data application, and they are usually defined by a separate Document Type Declaration (DTD) that contains a formal specification of the element names, their attributes (if any) and the nature of the data they can contain. Out of necessity, this book assumes that you are already familiar with XML such that your server-based applications serve up XML data exclusively, embed XML islands into HTML documents, or convert database data into XML. The focus of this chapter, and an extended application example of Chapter 57, is how to access custom elements that reside inside an IE XML element.

Elements and Nodes

Once you leave the specialized DOM vocabulary of HTML elements, the world can appear rather primitive — a highly granular world of node hierarchies, elements, element attributes, and node data. This granularity is a necessity in an environment in which the elements are far from generic and the structure of data in a document does not have to follow a format handed down from above. One Web application can describe an individual's contact information with one set of elements, while another application uses a completely different approach to element names, element nesting, and their sequence.

Fortunately, most, if not all, scripting you do on XML data is on data served up by your own applications. Therefore, you know what the structure of the data is — or you know enough of it to let your scripts access the data.

The discussion of the W3C DOM in Chapter 14 should serve as a good introduction to the way you need to think about elements and their content. All relevant properties and methods are listed among the items shared by all elements in Chapter 15.

Note

Microsoft has created a separate document object model exclusively for XML documents. To distinguish between the DOMs for XML and HTML documents, Microsoft calls the former the XML DOM and the latter the DHTML DOM. Specifications for the two DOMs overlap in some terminology, but the two models are not interchangeable. Read more about the Microsoft XML DOM at `http://msdn.microsoft.com`.

An XML *data island* is a hierarchy of nodes. Typically, the outermost nodes are elements. Some elements have attributes, each of which is a typical name/value pair. Some elements have data that goes between the start and end tags of the element (such data is a text node nested inside the element node). And some elements can have both attributes and data. When an XML island contains the equivalent of multiple database records, an element container whose tag name is the same as each of the other records surrounds each record. Thus, the `getElementsByTagName()` method frequently accesses a collection of like-named elements.

Once you have a reference to an element node, you can reference that element's attributes as properties; however, a more formal access route is via the `getAttribute()` method of the element. If the element has data between its start and end tags, you can access that data from the element's reference by calling the `firstChild.data` property (although you may want to verify that the element has a child node of the text type before committing to retrieving the data).

Of course, your specific approach to XML elements and their data varies with what you intend to script with the data. For example, you may wish to do nothing more with scripting than enable a different style sheet for the data based on a user choice. The evolving XSL (eXtensible Stylesheet Language) standard is a kind of (non-JavaScript) scripting language for transforming raw XML data into a variety of presentations. But you can still use JavaScript to connect user-interface elements that control which of several style sheets renders the data. Or, as demonstrated in Chapters 52 and 57, you may wish to use JavaScript for more explicit control over the data and its rendering, taking advantage of JavaScript sorting and data manipulation facilities along the way.

Table 33-1 is a summary of W3C DOM Core objects, properties, and methods that you are most likely to use in extracting data from XML elements. You can find details of all of these items in Chapter 15.

Table 33-1 **Properties and Methods for XML Element Reading**

Property or Method	***Description***
`Node.nodeValue`	Data of a text node
`Node.nodeType`	Which node type
`Node.parentNode`	Reference to parent node
`Node.childNodes`	Array of child nodes
`Node.firstChild`	First of all child nodes
`Node.lastChild`	Last of all child nodes
`Node.previousSibling`	Previous node at same level
`Node.nextSibling`	Next node at same level
`Element.parentNode`	Reference to parent node
`Element.childNodes`	Array of child nodes
`Element.firstChild`	First of all child nodes
`Element.lastChild`	Last of all child nodes
`Element.previousSibling`	Previous node at same level
`Element.nextSibling`	Next node at same level
`Element.tagName`	Tag name
`Element.getAttribute(`*name*`)`	Retrieves attribute (Attr) object
`Element.getElementsByTagName(`*name*`)`	Array of nested, named elements
`Attr.name`	Name part of attribute object's name/value pair
`Attr.value`	Value part of attribute object's name/value pair

XML Element Object

For HTML element properties, methods, and event handlers, see Chapter 15.

Properties	***Methods***	***Event Handlers***
`src`		
`XMLDocument`		

Syntax

Accessing XML element object properties or methods:

```
(IE5+)     [window.]document.all.elementID.property | method([parameters])
```

About this object

The XML element object is the primary container of an XML data island within an HTML page. If your scripts intend to traverse the node hierarchy within the element, or simply access properties of nested elements, then you should assign an identifier to the `ID` attribute of the XML element. For example, if the XML data contains results from a database query for music recordings that match some user-entered criteria, each returned record might be denoted as a RECORDING element as follows:

```
<XML ID="results">
    <SEARCHRESULTS>
        <RECORDING>
            ...elements with details...
        </RECORDING>
        <RECORDING>
            ...elements with details...
        </RECORDING>
        <RECORDING>
            ...elements with details...
        </RECORDING>
    </SEARCHRESULTS>
</XML>
```

Your script can now obtain an array of references to RECORDING elements as follows:

```
var recs = document.getElementById("results").getElementsByTagName("RECORDING")
```

While it is also true that there is no known HTML element with the tag name `RECORDING` (which enables you to use `document.getElementsByTagName("RECORDING")`), the unpredictability of XML data element names is reason enough to limit the scope of the `getElementsByTagName()` method to the XML data island.

Interestingly, the W3C DOM Level 2 does not define an XML element object within the HTML section. You cannot simply embed an XML document inside an HTML document: The standards clearly indicate that a document can be one or the other, but not both. While the NN6 DOM can recognize custom elements, the browser understandably gets confused when custom elements have tag names that already belong to the HTML DTD. Therefore, I do not recommend attempting to embed custom elements into an HTML document for NN6 unless it some day implements a mechanism similar to IE's XML data islands.

Note IE5/Macintosh does not support XML data islands.

Properties

`src`

Value: String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

The `src` property represents the `SRC` attribute of the XML element. The attribute points to the URL of an external XML document whose data is embedded within the current HTML document.

`XMLDocument`

Value: Object Reference Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility								✓	✓

The `XMLDocument` property returns a reference to Microsoft's proprietary XML `document` object and the object model associated with it (the so-called XML DOM). A lot of this object model is patterned after the W3C DOM model, but access to these properties is via a rather roundabout way. For more details, visit

`http://msdn.microsoft.com/xml/reference/xmldom/start.asp`

✦ ✦ ✦