CHAPTER 4

The Document Object Model (DOM)

W<small>E HAVE SEEN HOW AN</small> XML document can be transformed into an HTML docu-
ment for presentation and into another XML document. What else can we do
with an XML document? The next natural thing we would like to do is to manipu-
late the data and structure in an XML document. The basic functions would
include insertion of new information, modification or deletion of existing data in
the document. In this chapter, we introduce DOM programming as a technique
for achieving this objective.

Introduction to DOM

The Document Object Model (DOM) provides an application programming
interface (API) that is platform- and language-neutral. Application developers
can access and manipulate the data in an XML document through these inter-
faces using their favorite scripting or programming languages and without
having to worry about the platform on which the scripts will run.

> **NOTE** *The freedom in choosing the platform and language
> is, however, dependent on the parser to be used to expose the
> XML DOM implementation of an XML document. We will
> look at DOM parsers in a moment.*

The World Wide Web Consortium (W3C) releases the DOM specifications
according to levels:

- The DOM Level 1 specification was released as W3C Recommendation
  in October 1998. The working draft of the second edition is submitted in
  September 2000 to incorporate errata changes of the earlier version.

- The DOM Level 2 modules were released as W3C Recommendations in
  November 2000.

- The DOM Level 3 working drafts were submitted in the first half of 2001. The latest working draft available at the point of this writing is dated September 13, 2001.

DOM-based APIs are being developed in the specifications for the Mathematical Markup Language (MathML), Scalable Vector Graphics (SVG), and Synchronized Multimedia Integration Language (SMIL).

DOM is a tree-based API to documents that requires the entire XML document to be represented in memory while processing it. An alternative to DOM is the event-based Simple API for XML (SAX), which can be used to process large XML documents with limited memory available for processing.

NOTE   *SAX is discussed in Chapter 7.*

XML DOM Parsers

XML DOM parsers are software that are able to interpret an XML document as a DOM instance, typically representing it as a tree of nodes in memory. We shall call this tree the document tree as used in the DOM specifications or simply DOM tree.

Note that the DOM specifications do not specify the data structure for implementation, but typically a tree structure is most natural for representing a DOM implementation of an XML document. A tree structure enables ease of access to its various types of nodes by their relative position. The tree traversal order is typically top-down and left-to-right.

XML DOM parsers include the MSXML that is in the Microsoft's Internet Explorer, Oracle's XML Parser for Java v2, and Xerces of the Apache Software Foundation.

All conforming implementations of DOM must fully implement the fundamental interfaces of the DOM specification.

Support of DOM in MSXML3

MSXML3 implements the fundamental and extended interfaces of DOM Level 1 specification.

The DOM Level 1 specification consists of two parts, namely Core and HTML. The Core DOM provides fundamental interfaces for representing any structured document and also extended interfaces for representing an XML document. The HTML DOM provides high-level interfaces that are used with the fundamental Core DOM to provide a representation of an HTML document.

All conforming implementations of DOM Level 1 must fully implement the fundamental interfaces and exception listed in Table 4-1.

Table 4-1. DOM Level 1 Fundamental Interfaces and Exception

FUNDAMENTAL INTERFACE/EXCEPTION	DESCRIPTION
Interface DOMImplementation	Provides methods for performing operations that are independent of any specific instance of the document object model.
Interface DocumentFragment	A lightweight Document that is used to represent a portion of a document tree or a new fragment of a document.
Interface Document	Represents an entire XML document. An entry point for accessing the entire document is provided as the root of the document tree. Note that this root is not the root element of the represented XML document. This root is above the latter in the tree hierarchy.
Interface Node	Represents a node in the document tree. A node in a tree can represent different valid components in an XML document such as element, attribute, text, comment and processing instruction, document type, as well as DOM-specific components such as a document instance and document fragment.
Interface NodeList	An ordered (based on the document tree) collection of nodes, where the individual nodes may be addressed by an index, starting from 0 for the first node in the collection.

Table 4-1. DOM Level 1 Fundamental Interfaces and Exceptions (continued)

FUNDAMENTAL INTERFACE/EXCEPTION	DESCRIPTION
Interface NamedNodeMap	As in NodeList, NamedNodeMap refers to a collection of nodes. However, in this case, the nodes may be addressed by an index as well as by name. It is typically used for attribute nodes.
Interface CharacterData	This is an extension of the Node interface, specifically for handling any character data.
Interface Attr	This is another extension of the Node interface, specifically for handling an attribute.
Interface Element	This is an extension of the Node interface, specifically for handling an element.
Interface Text	This is an extension of the CharacterData interface, specifically for handling a text node.
Interface Comment	This is an extension of the CharacterData interface, specifically for handling a comment node.
Exception DOMException	An instance of DOMException is used to raise an exception when an undesirable or exceptional circumstance occurs. An example of such exceptional circumstance is an unsuccessful attempt to access a nonexistent node in a given context.

In addition, MSXML3 also implements extended interfaces, which are shown in Table 4-2.

Table 4-2. DOM Level 1 Extended Interfaces

EXTENDED INTERFACE	DESCRIPTION
Interface CDATASection	This is an extension of the Text interface, specifically for handling a CDATA section.
Interface DocumentType	This is an extension of the Node interface, specifically for handling the document type (through the <!DOCTYPE> element) of an XML document.
Interface Entity	This is for representing a parsed or unparsed entity.
Interface EntityReference	This is for representing an XML entity reference such as and ".
Interface Notation	This is an extension of the Node interface, specifically for handling a notation.
Interface ProcessingInstruction	This is an extension of the Node interface, specifically for handling a processing instruction.

Each interface and exception may consist of one or more of the following types of components:

- Predefined constants

- Properties, also called attributes (which should not be confused with the attributes used in an XML document that are associated with elements)

- Methods, also called functions

Refer to Appendix B for a complete list of properties and methods that are associated with each interface and exception in Table 4-1 and Table 4-2.

Representing XML Document as a Tree

As mentioned earlier, the MSXML parser represents an XML document as a tree of nodes in memory when the document is loaded. The DOM library provides programmers with APIs to manipulate the tree that is built in the memory.

Figure 4-1 depicts the role of the parser in facilitating the programming that can be incorporated into an application to access and edit data in an XML document.
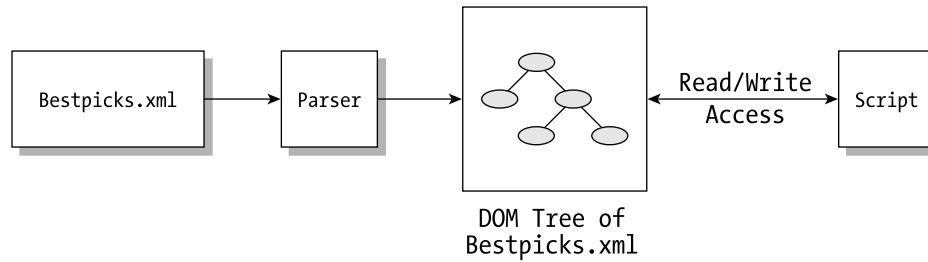
Figure 4-1. Transforming an XML document into a DOM tree for read/write access by an application

Each of the components in an XML document can be mapped onto a *node*. Hence we would find different types of nodes such as element node, attribute node, comment node, and processing-instruction node. On top of that, there is an additional node, which alludes to the entire document and is called the `Document` node of the tree structure.

Consider the following XML document, Bestpicks.xml:

```
<?xml version="1.0"?>
<BOOKS xmlns:apress="http://www.apress.com">
  <apress:BOOK ISBN="1893115860" Pages="357" Type="SOFT">
      <!--This is the first book-->
      <TITLE>A Programmer's Introduction to C#</TITLE>
      <AUTHOR>Eric Gunnerson</AUTHOR>
      <PRICE>34.95</PRICE>
  </apress:BOOK>
  <BOOK ISBN="189311595X" Pages="380" Type="HARD">
      <TITLE>Cryptography in C &amp; C++</TITLE>
      <AUTHOR>Michael Welschenbach</AUTHOR>
      <PRICE>49.95</PRICE>
  </BOOK>
  <BOOK ISBN="1893115763" Pages="400" Type="SOFT">
      <TITLE>C++ for VB Programmers</TITLE>
      <AUTHOR>Jonathan Morrison</AUTHOR>
      <PRICE>49.95</PRICE>
  </BOOK>
</BOOKS>
```

The Bestpicks.xml document can be depicted as a tree as shown in Figure 4-2, which for the sake of simplicity, shows only the first two <BOOK> elements due to space. The attribute nodes are represented as ellipses with dotted lines and the text nodes are represented as rectangles.
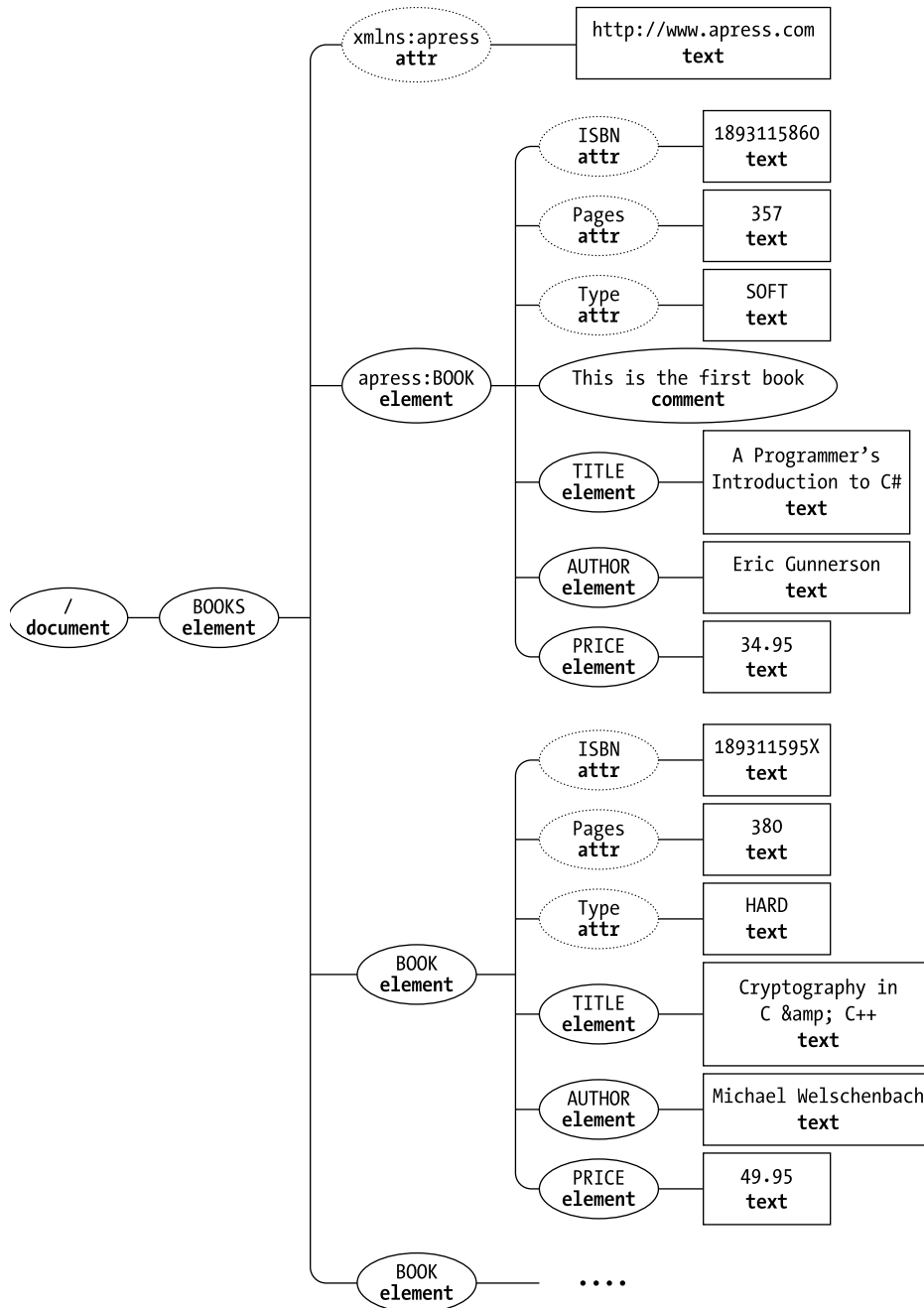
Figure 4-2. Partial tree representation of Bestpicks.xml

Creating a DOM Tree of an XML Document in Memory

The creation of a DOM representation of an XML document in the memory before further manipulating it through the DOM API involves three basic steps:

1. Create an instance of the Document interface, i.e., a DOM instance or DOM object

2. Specify the asynchronization requirement

3. Load content into the instance of DOM created in the first step

As shown in the following example, the basic steps can be coded in VBScript where data is loaded from the document, Bestpicks.xml, using the MSXML parser at the client:

```
<script language="VBScript">
<!--
    Dim docObj
    Set docObj = CreateObject (MSXML2.DOMDocument)
    docObj.async = false
    docObj.load ("Bestpicks.xml")
//-->
</script>
```

To load data from the same XML document at the server using ASP and VBScript, do the following:

```
<%
    Dim docObj
    Set docObj = Server.CreateObject (MSXML2.DOMDocument)
    docObj.async = false
    docObj.load (Server.MapPath("Bestpicks.xml"))
%>
```

The client-side script allows us to easily add a display statement after the loading statement to check the loaded contents of the DOM instance:

```
    msgbox docObj.xml
```

The content of the document, to which docObj points, appears as shown in Figure 4-3.

```
VBScript                                                    ×

<?xml version="1.0"?>
<BOOKS xmlns:apress="http://www.apress.com">
        <apress:BOOK ISBN="1893115860" Pages="357" Type="SOFT">
                <!--This is the first book-->
                <TITLE>A Programmer's Introduction to C#</TITLE>
                <AUTHOR>Eric Gunnerson</AUTHOR>
                <PRICE>34.95</PRICE>
        </apress:BOOK>
        <BOOK ISBN="189311595X" Pages="380" Type="SOFT">
                <TITLE>Cryptography in C &amp; C++</TITLE>
                <AUTHOR>Michael Welschenbach</AUTHOR>
                <PRICE>49.95</PRICE>
        </BOOK>
        <BOOK ISBN="1893115763" Pages="400" Type="SOFT">
                <TITLE>C++ for VB Programmers</TITLE>
                <AUTHOR>Jonathan Morrison</AUTHOR>
                <PRICE>49.95</PRICE>
        </BOOK>
</BOOKS>

                            OK
```

Figure 4-3. Display of the DOM object for Bestpicks.xml using a message box

We next look at each of the three steps in greater detail.

Creating an Instance of DOM

The DOM Level 1 does not provide a way to create a Document instance. That is, the creation of a Document instance is an implementation-specific operation. The Microsoft-specific APIs that are used for creating Document instances are shown in the following code snippet; the first is used in client-side script while the second is used in server-side script:

```
Set docObj = CreateObject (MSXML2.DOMDocument)
```

and

```
Set docObj = Server.CreateObject (MSXML2.DOMDocument)
```

Setting Asynchronization Flag

The asynchronization requirement is specified either as `true` (default value) or `false` before the document object just created is populated with content. A `false` value indicates that the next call for loading content into the document object is a blocked operation. That is, loading must be completed before the script that follows is processed.

On the other hand, a `true` value to the `async` property allows processing of the script that follows to continue after the load operation is set to action.

For the latter case, MSXML3 provides properties for use with a document object to check its loading state before processing the DOM tree. This can be done using the event handler `onDataAvailable` or through explicitly testing the `readyState` property of the Document instance.

For example, the following causes the function `processData()` to be executed when data becomes available:

```
docObj.onDataAvailable = processData()
```

There are several constant values defined for the `readyState` property to indi-cate the current loading state of the XML document (DOM) object. They are as listed in Table 4-3.

Table 4-3. Possible Values of `readyState`

VALUE OF *READYSTATE*	DESCRIPTION
1	Loading of data is in progress.
2	Data has been loaded. Reading and parsing of data are in progress.
3	Some data has been read and parsed, which is available for read-only access.
4	The entire loading of the document is completed. The result may be a success or failure. Further check is needed to ensure successful loading.

The parseError object holds error information regarding a DOM object (e.g., `docObj`) and checks if the error code is 0, which indicates successful loading, or otherwise:

```
if docObj.readystate = 4 then
    set loadingErr = docObj.parseError
    if  loadingErr.errorCode = 0 then
```

```
        msgbox "successful loading"
    else
        msgbox "loading failed"
    end if
end if
```

Loading XML Data into a DOM Instance

In our example, the load() method is used to load the XML data or content into the DOM instance, docObj. Specifically, the XML content is loaded from an XML document named Bestpicks.xml. A DOM instance loaded from an XML document would have a non-null URL name that can be accessed using the URL property of the DOM instance. The following code:

```
msgbox (docObj.url)
```

will display a message box that is similar to the one that appears in Figure 4-4.
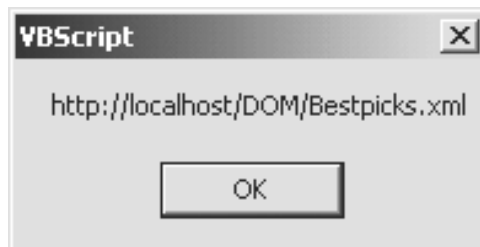
Figure 4-4. Displaying URL of data source for a DOM object

If the source of XML data is a string, the loadXML() method is used for both client- and server-side scripts, as illustrated in the following example:

```
docObj.loadXML ("<?xml version='1.0'?>" &_
                "<BOOKS><BOOK>" &_
                "<TITLE>A Programmer's Introduction to C#</TITLE>" &_
                "</BOOK></BOOKS>")
```

If the source of XML data for loading is passed to an ASP script through a request, we should load the data from the Request object as defined in the ASP programming model:

```
docObj.load Request
```

> **NOTE** *We look at an example on loading from the Request object later in this chapter.*

Saving a DOM Tree

You can save a DOM representation residing in the memory into an XML document as a text file with extension *.xml,* which is in turn stored in some permanent storage. This is appropriate only in server-side processing for obvious security reasons; no client would freely let a script create a file locally.

The following server-side ASP script fragment shows how a DOM tree that is referred to as `docObj` is saved as a text stream into a file named mysample.xml through the `save()` method:

```
docObj.save (Server.MapPath("mysample.xml"))
```

After this line is executed, check the server directory and you should see the file, mysample.xml, created in the same folder as the current ASP script. If you want to save the data into a subdirectory, named *samples,* of the folder containing the current ASP script, simply modify the path as shown here:

```
docObj.save (Server.MapPath("samples/mysample.xml"))
```

Alternatively, you may retrieve and save the DOM tree into a temporary string variable in an application using the `xml` property of the document object. To accomplish this either on the client or server side, simply incorporate the following VBScript statement:

```
tempString = docObj.xml
```

In addition, we may want to display the string in a message box by incorporating the following VBScript statement into a client-side script:

```
msgbox docObj.xml
```

We have seen how a DOM object defined in the DOM Level 1 specification, also known as a DOMDocument object as used in MSXML3, is created and loaded with data. We are now ready to manipulate the DOM tree that is built by the MSXML3 parser. The rest of the chapter demonstrates the use of some of the DOM Level 1 interfaces to access and manipulate the DOM tree.

Fundamental APIs for Processing a DOM Tree

In this section, we introduce the properties and methods of some of the core interfaces implemented by the MSXML parser. The interfaces discussed include the Document, Node, NodeList, Element, and Attr interfaces.

Reference to the DOM Tree

When the DOM tree is first created, the reference to the DOM tree is the root node of the tree, which is a Document node, or sometimes also called a DOM node. In the previous section, we saw how a DOM tree is created to represent the document Bestpicks.xml through the loading method:

```
docObj.load ("Bestpicks.xml")
```

In this example, the reference to the DOM tree is the Document node named docObj.

Reference to the Document Root Element

To start traversing to the next level after the root of the DOM tree for Bestpicks.xml, we need to reference the node representing the document root element, <BOOKS>, which can easily be accomplished by using the documentElement property of the Document node, docObj:

```
Set BOOKSnode = docObj.documentElement
```

The variable BOOKSnode now references the <BOOKS> element of the DOM tree that is shown in Figure 4-5.
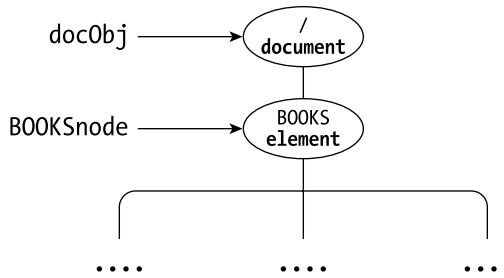
Figure 4-5. References to root node and document element

Fundamentals of a Node

Every node in the DOM tree is an instance of the Node interface. Each node has properties and methods as defined in DOM Level 1.

In addition, each specific type of node, such as an element node, may further extend the Node interface with additional properties and methods that pertain to that node type.

What Do We Know about a Node?

Each node of the DOM tree has a name, type, and value. We can find out the name, type, and value of a node, such as BOOKSnode, using the following properties:

```
BOOKSnode.nodeName
BOOKSnode.nodeType
BOOKSnode.nodeValue
```

The values of the properties nodeName, nodeType, and nodeValue of a node depend on the type of the node as described in Table 4-4.

Table 4-4. Name, Type, and Value of a Node

TYPE OF NODE	VALUE OF *NODENAME*	CONSTANT AND NUMERIC VALUES OF *NODETYPE*	VALUE OF *NODEVALUE*
Element	Name of the element tag	ELEMENT_NODE(1)	null
Attr	Name of the attribute	ATTRIBUTE_NODE (2)	Value of the attribute
Text	#text	TEXT_NODE (3)	Text value of the node
CDATA Section	#cdata-section	CDATA_SECTION_NODE (4)	CDATA content
Entity-Reference	Name of the referenced entity	ENTITY_REFERENCE_NODE (5)	null
Entity	Entity name	ENTITY_NODE (6)	null
Processing-Instruction	target	PROCESSING_INSTRUCTION_NODE (7)	Value of the in-struction portion
Comment	#comment	COMMENT_NODE (8)	Value of the comment node
Document	#document	DOCUMENT_NODE (9)	null
Document Type	Name of document type	DOCUMENT_TYPE_NODE (10)	null
Document Fragment	#document-fragment	DOCUMENT_FRAGMENT_NODE (11)	null
Notation	Notation name	NOTATION_NODE (12)	null

The following is a list of other properties of a Node object as defined by DOM Level 1. More detailed descriptions of the properties are given in Appendix B.

- parentNode

- childNodes

- firstChild

- lastChild

- previousSibling

- nextSibling

- attributes

- ownerDocument

MSXML extends the property set defined in DOM Level 1 with several other useful properties such as the nodeTypeString and xml properties. In fact, we have seen the latter earlier in this chapter when we discussed the saving of a DOM tree. The nodeTypeString property refers to the type of node by character string, e.g., "element" instead of the value 1.

Consider Listing 4.1 (NodeProperties.html), which contains a script to access some of the properties of some nodes in the DOM tree representing the XML document Bestpicks.xml:

Listing 4-1. NodeProperties.html

```vbscript
<html>
<script language="vbscript">
<!--
    'Initializing and loading a DOM object
    Dim docObj
    Set docObj = CreateObject("Msxml2.DOMDocument")
    docObj.async = false
    docObj.load "Bestpicks.xml"

    'Initializing 2 Node objects
    Set BOOKSnode = docObj.documentElement
    Set firstBOOK = BOOKSnode.firstChild

    'Displaying name, type and value properties of docObj
    document.write "<h2>Name, type and value of <em>docObj</em></h2>"
    document.write "<p>nodeName: <b>" & docObj.nodeName & "</b>"
    document.write "<br>nodeType: <b>" & docObj.nodeType &_
                    " (" & docObj.nodeTypeString & ")</b>"
    document.write "<br>nodeValue: <b>" & docObj.nodeValue & "</b>"

    'Displaying name, type and value properties of BOOKSnode
    document.write "<h2>Name, type and value of <em>BOOKSnode</em></h2>"
    document.write "<p>nodeName: <b>" & BOOKSnode.nodeName & "</b>"
    document.write "<br>nodeType: <b>" & BOOKSnode.nodeType &_
                    " (" & BOOKSnode.nodeTypeString & ")</b>"
    document.write "<br>nodeValue: <b>" & BOOKSnode.nodeValue & "</b>"
```

```
    'Displaying XML content of the first BOOK
    msgbox firstBOOK.xml
//-->
</script>
</html>
```

Figure 4-6 displays a page of contents and a message box that is produced by NodeProperties.html when it is loaded using Internet Explorer 5.0.
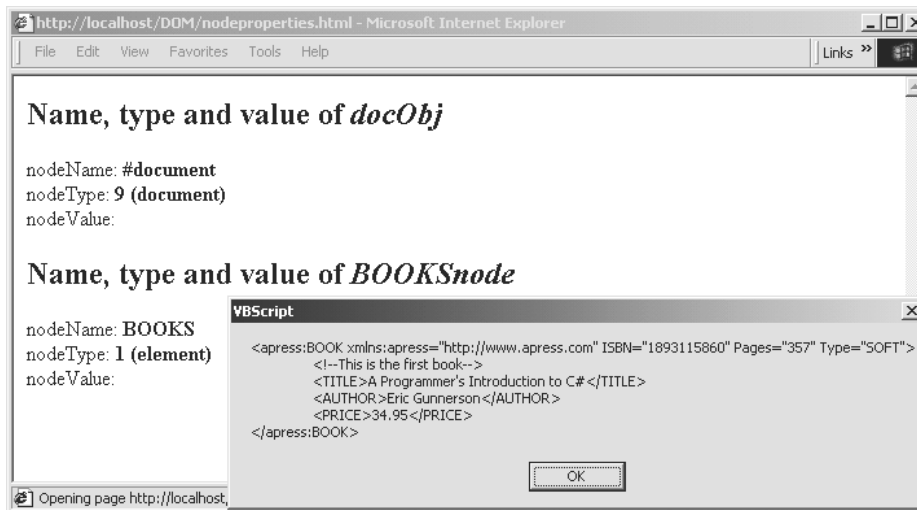

Figure 4-6. Display produced by NodeProperties.html using Internet Explorer 5.0

What Can We Do with a Node?

We have seen earlier how you can access the first child node of a given node by using the `firstChild` property. There are occasions when you would like to test if the node in question has any child node before addressing a specific child node. The `hasChildNodes()` method provides a way to check if a node contains any child node. It returns `true` if the answer is positive.

For example, if `docObj` and `BOOKSnode` are defined in NodeProperties.html, adding either of the following lines of script

```
    msgbox docObj.hasChildNodes()
    msgbox BOOKSnode.hasChildNodes()
```

yields the message box that is shown in Figure 4-7 in which the Boolean result is `True` since each of the nodes consists of at least one child node.
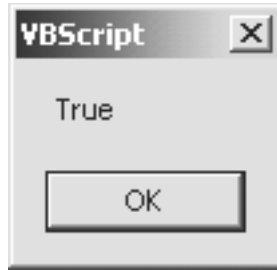
Figure 4-7. Response to `hasChildNodes()` *for* `docObj` *or* `BOOKSnode`

MSXML provides the following additional methods for the Node interface as listed in Table 4-5.

Table 4-5. MSXML Methods to the Node Interface

METHOD	DESCRIPTION
selectNodes (p)	This method returns a list of descendant nodes of this context node that matches the pattern specified by p, which is a valid XPath expression.
selectSingleNode (p)	This method returns the first descendant node of this context node that matches the pattern specified by p, which is a valid XPath expression.
transformNode (s)	This method returns the result, in string, of transformation applied on this node and its children using the supplied XSLT stylesheet DOM object, s.
transformNodeToObject (s, r)	This method processes this node and its children using the supplied XSLT stylesheet DOM object, s, and returns the resulting transformation in the supplied object, r.

Adding the following two lines of script in NodeProperties.html

```
msgbox "Title of first <BOOK>: " &_
        docObj.selectSingleNode("//TITLE").firstChild.nodeValue
```

yields the display that is shown in Figure 4-8.

Figure 4-8. Displaying the first title in the DOM object, docObj

Since <TITLE> is an element, its node value is null. In order to display the title, we must traverse down the tree shown in Figure 4-2 from the <TITLE> node of the first <BOOK> node, which gives us a text node next. The content of the text node is "A Programmer's Introduction to C#." Since the nodeValue of a text node displays the content of the node, we can use this property to print out the title of the book concerned.

We have seen how transformNode() is used in Chapter 3:

```
Response.write (xml.transformNode(xsl))
```

The DOM object (xml) representing an XML data document was transformed by the stylesheet represented as the DOM object, xsl. The result is a string, which is output as a stream from the server to the client.

By using transformNodeToObject() we are able to realize the same transformation and output the result to an object such as a DOM object as follows:

```
Set output = CreateObject(MSXML2.DOMDocument)
xml.transformNodeToObject (xsl, output)
```

Fundamentals of a NodeList

The NodeList interface is different from the Node interface in that it is representing a set of nodes instead of just one single node. There is only one property and one method defined for the NodeList interface by the World Wide Web Consortium (W3C) DOM Level 1. They are the length property and the item() method, and they are discussed in the next two subsections.

Size of a NodeList

Assuming that `BOOKSnode` represents the <BOOKS> element node of the tree that is shown in Figure 4-2, we can find out the number of <BOOK> child nodes that the tree contains by using the following line of script:

```
msgbox "No. of <BOOK> nodes under <BOOKS>: " &_
        BOOKSnode.selectNodes("BOOK").length
```
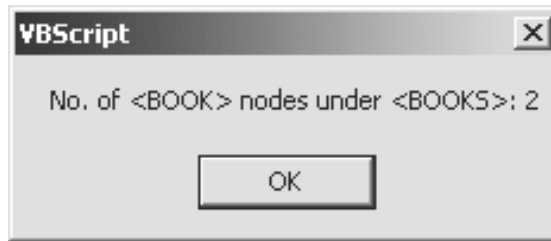

Figure 4-9. Display the number of <BOOK> nodes under the <BOOKS> node

The expression, `BOOKSnode.selectNodes("BOOK")`, returns a NodeList object, which is an ordered set of descendant nodes of the <BOOKS> node and which has the tag name, `BOOK`. The length property of a node list returns the number of nodes it contains, which is two in this case since there are two such <BOOK> nodes under the <BOOKS> node.

Getting to the Individuals of a NodeList

The `item()` method takes in a parameter, say `i`, which is used as the index of the requested node in a given node list. In short, it returns the $(i+1)^{th}$ node in the list for the specified index parameter, `i`, since the index starts with 0.

Using the same reference node that is indicated by `BOOKSnode`, as shown in Figure 4-5, consider the following lines of script:

```
Set BOOKnodes = BOOKSnode.selectNodes("BOOK")
msgbox BOOKnodes.item(1).xml
```

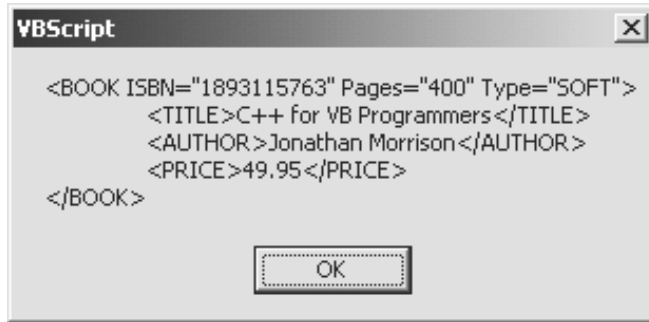The script displays the subtree that is the second <BOOK> element of Bestpicks.xml.

Figure 4-10. XML data for the second <BOOK> element in Bestpicks.xml

The expression, `BOOKnodes.item(1)`, can be simplified to `BOOKnodes(1)`.

Fundamentals of an Element Node

The Element node is a special case of node that extends the Node interface with an additional property and some other methods pertaining to elements. In addition, MSXML also defines some useful properties and methods to this interface.

Names of an Element Node

The first thing we can find out about an element node is its name.

Let's assume that the same definition applies to `firstBOOK` (as we have seen earlier), which is depicted in Figure 4-11.
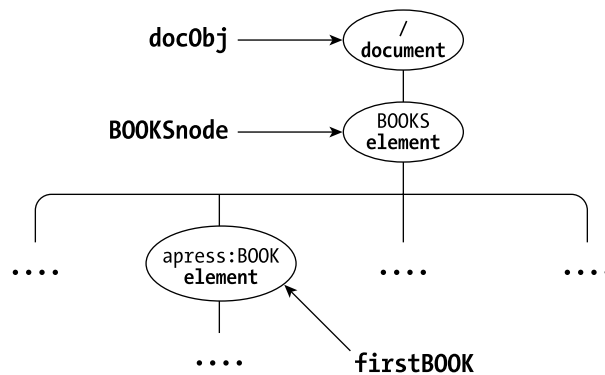

Figure 4-11. Reference to the first child node of the <BOOKS> node

Consider the following script fragment:

```
msgbox "nodeName: " & firstBOOK.nodename & vbCRLF &_
        "tagName:   " & firstBOOK.tagname & vbCRLF &_
        "baseName: " & firstBOOK.basename & vbCRLF &_
        "prefix:    " & firstBOOK.prefix & vbCRLF &_
        "namespaceURI: " & firstBOOK.namespaceURI
```

The properties that are displayed in Figure 4-12 refer to naming information of an element, in which the last three properties are MSXML extensions to the DOM Level 1 definition.
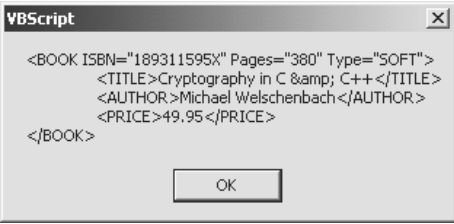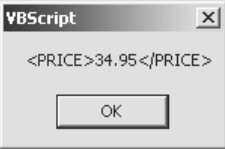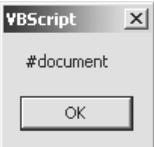

Figure 4-12. Naming information of the first book in Bestpicks.xml

Relating to Other Element Nodes

Let's try to traverse the DOM tree using firstBOOK as the reference node (Table 4-6).

Table 4-6. Examples of Traversal to Other Element Nodes from the Node Referenced by `firstBOOK`
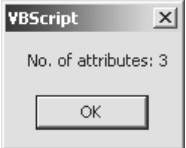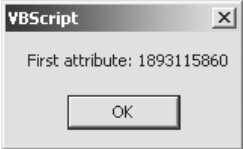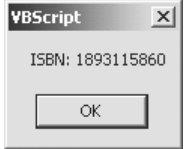
DESCRIPTION	MESSAGE BOX SCRIPT & DISPLAY
Displaying the name of parent node of `firstBOOK`.	Msgbox firstBOOK.parentNode.nodeName

| Displaying the contents of the node that is the next sibling (to the right) of `firstBOOK`. | Msgbox firstBOOK.nextSibling.xml |

| Displaying the contents of the node that is the last child element node of `firstBOOK`. | Msgbox firstBOOK.lastChild.xml |

| Displaying the name of the Document node that represents the document containing `firstBOOK`. | Msgbox firstBOOK.ownerDocument.nodeName |

Learning about Attributes of an Element Node

Let's investigate the attributes of `firstBOOK` (Table 4-7).

Table 4-7. Retrieving Information of Attributes of the Node Referenced by `firstBOOK`

DESCRIPTION	MESSAGE BOX SCRIPT & DISPLAY
Displaying the number of attributes associated with `firstBOOK`.	Msgbox "No. of attributes:" & firstBOOK.attributes.length
	[VBScript message box: "No. of attributes: 3", OK]
Displaying the value of the first attribute of `firstBOOK`.	Msgbox "First attribute:" & firstBOOK.attributes(0).nodeValue
	[VBScript message box: "First attribute: 1893115860", OK]
Displaying the value of the ISBN attribute of `firstBOOK`.	Msgbox "ISBN:" & firstBOOK.getAttribute("ISBN")
	[VBScript message box: "ISBN: 1893115860", OK]

It should be highlighted that `firstBOOK.attributes` returns a NamedNodeMap object, which is an unordered list of nodes, whose individual nodes may be retrieved by their names, such as using the `getAttribute()` method.

Fundamentals of an Attr Node

We have just seen how an attribute of an element node is accessed. We look at some of the properties and methods of an `Attr` (i.e., attribute) node here.

Since the `Attr` interface extends the Node interface, most of the properties and methods that we discussed under the Node interface also apply to an `Attr` node.

Consider the following script fragment, which first sets a reference to the `Pages` attribute of `firstBOOK` using the `getAttributeNode()` method of an element node, and then displays some properties pertaining to the attribute node, as shown in Figure 4-13 and Figure 4-14, respectively:
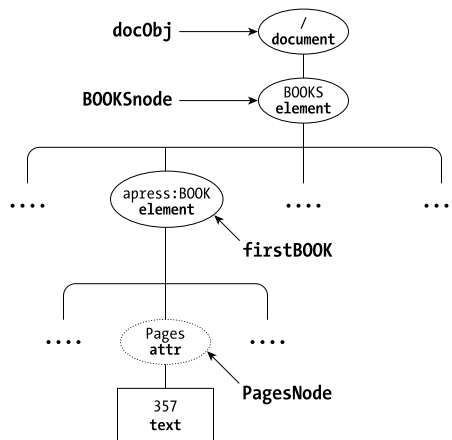
Figure 4-13. Reference to the Pages *attribute of the node referenced by* firstBOOK
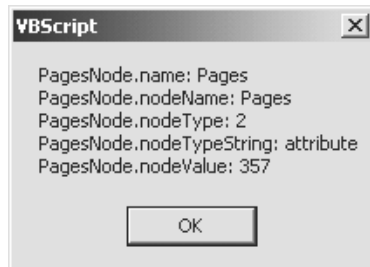
Figure 4-14. Naming information of the Pages *attribute of the node referenced by* firstBOOK

```
Set PagesNode = firstBOOK.getAttributeNode("Pages")
msgbox "PagesNode.name: "        & PagesNode.name & vbCRLF &_
        "PagesNode.nodeName: "   & PagesNode.nodeName & vbCRLF &_
        "PagesNode.nodeType: "   & PagesNode.nodeType & vbCRLF &_
        "PagesNode.nodeTypeString: " & PagesNode.nodeTypeString & vbCRLF &_
        "PagesNode.nodeValue: " & PagesNode.nodeValue
```

The name property is an extended property to the Node interface. The rest of the properties are not new to us, as they had been mentioned earlier when we introduced the fundamentals of the Node interface.

Client-Side DOM Programming—Shopping Cart

In this section, we make use of a case study to demonstrate DOM programming on the client side. We introduce the use of more properties and methods of the various interfaces for accomplishing the functionality of the application.

We extend the application requirements to include the server-side DOM programming in a later section.

Client-Side Application Requirements

Let's implement a shopping cart that keeps track of the items selected by the user from a list of best-sellers of some bookstore. The shopping cart can expand or shrink as the user adds or drops items from his or her shopping cart.

The following describes the client-side requirements from the user's perspective:

1. The user is presented with a selection list of best-sellers from a bookstore.

2. The user can select an item and specify the quantity to add to the shopping cart. The user can remove the item from the cart by specifying the quantity to be zero.

3. The user can check out and submit the shopping cart to a backend server through clicking on a button.

To simplify our implementation, we will not provide a mechanism for the user to view the shopping cart since the coding techniques that are required for this functionality are similar to some of those that are used for implementing the three requirements specified in the preceding list. The user interface design is not of utmost concern here and we will try to keep a simple interface to avoid cluttering the essential code for manipulating the DOM objects, which is relevant to this chapter. Also, we will skip the script necessary for validating user's input, as that does not add much value to illustrating programming techniques with DOM.

Initialization

At this point, we can identify a few initialization tasks:

*   Declaring global variables

*   Loading Bestpicks.xml as a DOM tree

*   Creating initial DOM tree for the shopping cart

*   Invoking the loading of a form for user to do ordering

We will not show all of the global variables at once; instead we will mention them when they are introduced as the need arises.

We will put all the initialization script in a VBScript subroutine named `initialize`.

Loading an XML Document into a DOM Tree

To access the information enclosed in the XML document, Bestpicks.xml, we first create a DOM object and load it with data from the document. We will also initialize a variable (`BOOKSnode`) to point to the document root (<BOOKS> element):

```
Dim booksDoc, BOOKSnode
Sub initialize
    Set booksDoc = CreateObject("MSXML2.DOMDocument")
    booksDoc.async = false
    booksDoc.load "Bestpicks.xml"
    Set BOOKSnode = booksDoc.documentElement
End Sub
```

Here, `booksDoc` and `BOOKSnode` are declared as global variables as they should be accessible throughout most of the other subroutines.

Creating a DOM Tree for the Shopping Cart

We need to create the DOM tree for the shopping cart from scratch. This can be accomplished by the following code inserted into the `initialize` subroutine, where `orderDoc` and `ORDERSnode` are declared as global variables:

```
    Set orderDoc = CreateObject("Msxml2.DOMDocument")
    orderDoc.async = false
    orderDoc.loadXML "<?xml version='1.0'?>" &_
                     "<ORDERS></ORDERS>"
    Set ORDERSnode = orderDoc.documentElement
```

After the initialization, the cart will have the following DOM tree structure, as shown in Figure 4-15.
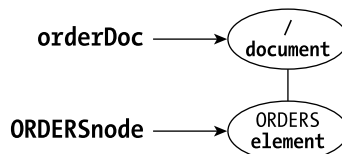
Figure 4-15. Initial DOM structure of the shopping cart (`orderDoc`*)*

We can also display the XML contents (Figure 4-16) using a message box during development, such as:

```
msgbox orderDoc.xml
```


*Figure 4-16. Initial XML content of the shopping cart (*orderDoc*)*

Invoking the Loading of an Order Form

We will make use of a subroutine loadOrderForm to load an order form. To invoke the loading, we merely include the following call statement as the last line of code in the initialize subroutine:

```
Call loadOrderForm
```

Creating an Order Form

We will obtain the information of the best sellers from Bestpicks.xml and present them as a selection list in an order form as shown in Figure 4-17.

Figure 4-17. The order form

Traversing All BOOK Nodes of the DOM Tree

Since we need to present all the books, we need to traverse the entire DOM tree
representing Bestpicks.xml in search of book element nodes. When the DOM
tree was first created, the reference to the tree is the root node of the tree, that is,
booksDoc, which is a Document node. To start traversing the tree, we need to ref-
erence the node representing the document root, <BOOKS>, which was
accomplished in the initialization subroutine presented earlier. The object name
used to reference the <BOOKS> node is BOOKSnode.

To iterate through each child node (which may be a <BOOK> node or an
<apress:BOOK> node) of BOOKSnode in the DOM tree and present it as an option
of a selection list, we first gather the list of child nodes using the childNodes
property.

```
Set bestpicks = BOOKSnode.childNodes
```

We can insert this statement into the initialize subroutine.

> **NOTE** *We cannot collect all the nodes representing
> the books in the DOM tree based on a given element
> tag name by using either the DOM Level 1 method,*
> getElementsByTagName(), *or the* selectNodes() *method pro-
> vided by MSXML. This is because not all the eligible book
> elements have the same tag name.*

We will now write the subroutine to load the order form. In this subroutine,
we will construct a loop to perform information presentation n times where n is
the number of books in the node list, bestpicks. We will present each title as an
option in the selection list. Also, each option is associated with a value, which is
the book's ISBN. The following shows the construction of the selection list:

```
Sub loadOrderform
    document.write "<p><form name='orderForm'>"
    document.write "<select name='selectedbook' size='1'>"
    lastBookIndex = bestpicks.length - 1
    for i = 0 to lastBookIndex
        document.write "<option value='" &_
                        bestpicks(i).getAttribute("ISBN") & "'>" &_
                        bestpicks(i).selectSingleNode("TITLE").text
```

```
        next
        document.write "</select> "
        document.write "</form> "
End Sub
```

Note that the text property used to display the title is an extension provided by MSXML. Without this property, the display of the title is achieved through a longer expression, such as the following:

```
        bestpicks(i).selectSingleNode("TITLE").firstChild.nodeValue
```

Other Form Elements

To complete the form as shown previously in Figure 4-17, we need to add an input box for the user to specify the number of copies and two buttons. The first button is an order button to invoke the adding of the item to the shopping cart. The second button will lead to the submission of the shopping cart to the server for further processing.

Following is an example for coding the three items immediately after the output of the end-tag of the selection list (i.e., </select>):

```
    document.write "<p><input type='text' name='num' " &_
                    "size='3' maxlength='2'> copies"
    document.write "<p><input type='button' name='order' value='Order'>" &_
                    "   " &_
                    "<input type='button' name='checkout' value='Check Out'>"
```

Note that we have assigned the name, `order`, to the order button. Hence, upon clicking on the button, the subroutine, order_onClick, will be invoked to add the selected book into the shopping cart. Similarly, clicking on the check-out button will invoke the subroutine, checkout_onClick.

Updating the Shopping Cart

This functionality is invoked through the clicking of the order button in the order form. That is, the subroutine for handling this task is order_onClick, which we have just introduced.

In this subroutine, we need to update the DOM object representing the shopping cart, which is referenced by `orderDoc`, based on the input of the user through the order form.

The shopping cart referenced by orderDoc is first checked to see if the selected item exists in the cart. If so, we merely need to update the number of copies. However, if the user specified a quantity of 0 or less for an existing item, we will remove the corresponding node of the selected book from the shopping cart.

If the selected book is not found in the DOM object for the shopping cart, we need to add the necessary book element into the DOM tree unless the user specified 0 copy for the selected book.

```
1  Sub order_onClick
2      orderISBN = document.orderForm.selectedbook.value
3      orderQtty = document.orderForm.num.value
4
5      Set bookOrders = ORDERSnode.childNodes
6      found = 0
7      for each order in bookOrders
8          if order.selectSingleNode("ISBN").text = orderISBN then
9              found = 1
10             if orderQtty > 0 then
11                 'update the number of copies ordered
12                 order.selectSingleNode("QTTY").text = orderQtty
13             else
14                 'remove the order
15                 ORDERSnode.removeChild order
16             end if
17             exit for
18         end if
19     next
20
21     if found = 0 and orderQtty > 0 then
22         'add in new ORDER node
23         Set newOrderElement = orderDoc.createElement("ORDER")
24         Set newISBNElement = orderDoc.createElement("ISBN")
25         Set newQttyElement = orderDoc.createElement("QTTY")
26
27         Set newISBN = orderDoc.createTextNode(orderISBN)
28         Set newQtty = orderDoc.createTextNode(orderQtty)
29
30         newISBNElement.appendChild(newISBN)  'create <ISBN>xxx</ISBN>
31         newQttyElement.appendChild(newQtty)  'create <QTTY>yy</QTTY>
32
33         newOrderElement.appendChild(newISBNElement)
34         newOrderElement.appendChild(newQttyElement)
35         ORDERSnode.appendChild(newOrderElement)
36     end if
37 End Sub
```

We will highlight the new features that are introduced in the preceding subroutine:

- Lines 7 to 19 handle the case when the selected book has already been added into the shopping cart. Line 12 updates the quantity ordered if the user keys in a positive number. Line 15 uses the `removeChild` method to drop the selected book from the shopping cart by removing the corresponding <ORDER> node from the DOM tree.

- Lines 21 to 36 handle the case when the selected book is not found in the existing cart and that the user specified a positive value for the quantity.

- Lines 23 to 25 use the `createElement` method to create three new elements: <ORDER>, <ISBN>, and <QTTY> with no content.

- Lines 27 and 28 use the `createTextNode` method to create two text nodes for holding the ISBN and quantity values captured from the order form.

- Line 30 uses the `appendChild` method to associate the text node created in line 27 to the <ISBN> element created in line 24 as a child node to the latter. Hence, if the user selected the first book whose ISBN is 1893115860, then line 30 will result in the following element: <ISBN>1893115860</ISBN>

- Similarly, if the user input the quantity as 3, then line 31 will result in the following element: <QTTY>3</QTTY>

  Let's insert the following line of code for displaying the new order element immediately after line 34, as shown in Figure 4-18:

```
msgbox newOrderElement.xml
```
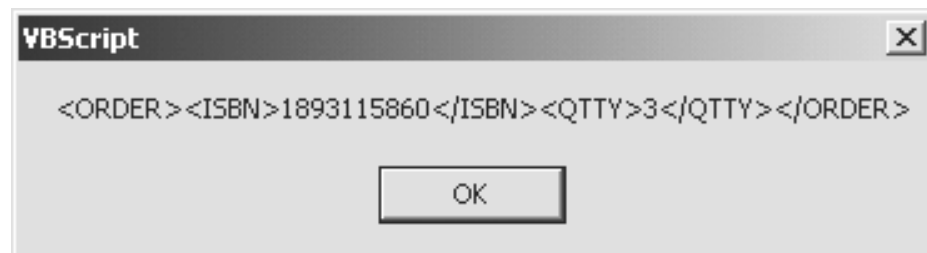

Figure 4-18. Contents of the new order just created

- Lines 33 and 34 use the <ISBN> and <QTTY> elements that were just created as building blocks to construct a higher-level <ORDER> element: <ORDER><ISBN>1893115860</ISBN> <QTTY>3</QTTY></ORDER>

- Finally, the new <ORDER> element constructed was appended as the last order under the <ORDERS> element referenced by `ORDERSnode`.

Checking Out

This functionality is invoked through clicking on the check out button in the order form and it involves the following final tasks to be developed on the client side:

- Prompts user to enter a user ID, which is then inserted as an attribute value of the <ORDERS> node in the DOM tree for the shopping cart.

- Creates an XMLHTTP object for delivering the shopping cart DOM object over the HTTP from the client browser to the targeted ASP script residing on an HTTP server.

- Waits for server's response and displays it on the browser.

- Re-initializes the DOM object representing the shopping cart.

Prompting for User ID

When the user chooses to check out, we would like the application to prompt the user for his user ID and insert into the DOM tree of the shopping cart as an attribute of the document root, which is <ORDERS> in our case.

We can make use of the `inputBox()` function to capture the user ID, as shown in Figure 4-19:

```
userID = inputBox ("Key in user ID:", "Check out")
```

Figure 4-19. Input box prompting for user ID

We will then create an attribute node with the `userID` value and associate that value with the <ORDERS> node that is referenced by the variable `ORDERSnode`, as shown in the following line:

```
ORDERSnode.setAttribute "userID", userID
```

If the user keys in the user ID Cust0001, the preceding will result in changing the start-tag to <ORDERS userID="Cust0001">, as shown in Figure 4-20.
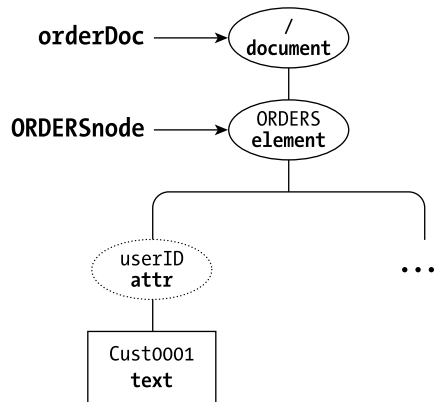

Figure 4-20. Adding the `userID` *attribute to the root element* `ORDERS`

Passing DOM Object to HTTP Server

First, we will create an XMLHTTP object to communicate with the backend HTTP server where the server script for processing the shopping cart can be reached.

This will enable us to pass an XML packet to the server using the HTTP connection between the client and the server.

The XMLHTTP object can request to open a connection with an HTTP server, specifying the HTTP request method, such as GET or POST, the resource it is looking for, such as an ASP script for processing the shopping cart, the asynchronization flag, as well as optional user id and password. If the asynchronization flag is set to `false`, then further execution will not occur at the client side until a response from the server is received. The final step is to send the XML packet out to the server using the `send` method.

An example of the corresponding code that is used for doing the tasks that were just described is shown in the following lines of script:

```
Set postObj = CreateObject ("MSXML2.XMLHTTP")
postObj.open "POST", "process.asp", false
postObj.send orderDoc
```

Displaying the Server's Response

If the response from the server is expected to be a text string, then we can retrieve the response through the `responseText` property of the XMLHTTP object. If we are expecting the response to be parsed XML content, we should use the `responseXML` property instead.

For example, if we want to display the server's text string response in a message box, we can achieve that through the following line of code:

```
msgbox postObj.responseText
```

Alternatively, we may want to receive the response as XML content and assign it to a newly created DOM object as shown here:

```
Set responseDoc = CreateObject ("MSXML2.DOMDocument")
Set responseDoc = postObj.responseXML
```

Re-initializing the Shopping Cart

After sending the shopping cart to the server, the content of the cart at the client side was not changed. If a different user would like to start ordering from the same ordering page left behind by the previous user, we need to make sure that the new user starts with an empty cart. To ensure that the cart is empty, we can include the following lines to re-initialize the shopping cart to its original state:

```
        orderDoc.loadXML "<?xml version='1.0'?>" &_
                      "<ORDERS></ORDERS>"
    Set ORDERSnode = orderDoc.documentElement
```

Complete Listing of the Client-Side Script

Now that we have completed the "bottom-up" discussion of the various function-alities that are performed on the client side, we present in Listing 4-2 the complete listing of the source code for the client. We have also included appropriate inline comments to help you understand the code as you glance through it:

Listing 4-2. Client-side script for shopping cart example

```
<html>
<head>
<script language="vbscript">
<!--
    Dim booksdoc, BOOKSnode, orderDoc, ORDERSnode
    Dim bestpicks

sub initialize
'Loading Bestpicks.xml
    Set booksDoc = CreateObject("Msxml2.DOMDocument")
    booksDoc.async = false
    booksDoc.load "Bestpicks.xml"
'Setting references to document root and list of books
    Set BOOKSnode = booksDoc.documentElement
    Set bestpicks = BOOKSnode.childNodes

'Creating new DOM tree for a shopping cart
    Set orderDoc = CreateObject("Msxml2.DOMDocument")
    orderDoc.async = false
    orderDoc.loadXML "<?xml version='1.0'?>" &_
                      "<ORDERS></ORDERS>"
    Set ORDERSnode = orderDoc.documentElement

'Loading the form for ordering
    call loadOrderForm
end sub
```

```
sub loadOrderForm
    document.write "<h3>Choose a book to add to your shopping cart:</h3>"
    document.write "<p><form name='orderForm'>"
    document.write "<select name='selectedbook' size='1'>"
    lastBookIndex = bestpicks.length - 1
    for i = 0 to lastBookIndex
        document.write "<option value='" & bestpicks(i).getAttribute("ISBN") &_
                        "'>" & bestpicks(i).selectSingleNode("TITLE").text
    next
    document.write "</select> "
    document.write "<p><input type='text' name='num' &_
                    "size='3' maxlength='2'> copies"
    document.write "<p><input type='button' name='order' value='Order'>" &_
                    "   " &_
                    "<input type='button' name='checkout' value='Check Out'>"
    document.write "</form>"
end sub


sub order_onClick
    orderISBN = document.orderForm.selectedbook.value
    orderQtty = document.orderForm.num.value

    Set bookOrders = ORDERSnode.childNodes
    found = 0
    for each order in bookOrders
        if order.selectSingleNode("ISBN").text = orderISBN then
            found = 1
            if orderQtty > 0 then
            'update the number of copies ordered
                order.selectSingleNode("QTTY").text = orderQtty
            else
            'remove the order
                ORDERSnode.removeChild order
            end if
            ' break
        end if
    next

    if found = 0 and orderQtty <> 0 then
        'add in new node
        Set newOrderElement = orderDoc.createElement("ORDER")
        Set newISBNElement = orderDoc.createElement("ISBN")
        Set newQttyElement = orderDoc.createElement("QTTY")
```

```
            Set newISBN = orderDoc.createTextNode(orderISBN)
            Set newQtty = orderDoc.createTextNode(orderQtty)

            newISBNElement.appendChild(newISBN)
            newQttyElement.appendChild(newQtty)

            newOrderElement.appendChild(newISBNElement)
            newOrderElement.appendChild(newQttyElement)
            ORDERSnode.appendChild(newOrderElement)
        end if
    end sub

    sub checkout_onClick
    'Prompting for user ID
        userID = InputBox("Key in user ID:- ", "Check out")
        ORDERSnode.setAttribute "userID", userID
        msgbox orderDoc.xml

    'Posting an XML packet to server and displaying server's string response
        Set postObj = CreateObject("Microsoft.XMLHTTP")
        postObj.open "POST", "process.asp", false
        postObj.send orderDoc
        msgbox postObj.responseText

    'Reinitializing orderDoc and ORDERSnode
        orderDoc.loadXML "<?xml version='1.0'?>" &_
                          "<ORDERS></ORDERS>"
        Set ORDERSnode = orderDoc.documentElement
    end sub

    //-->
    </script>
    </head>

    <body>
    <h1>Welcome to BookPick!</h1>
    <script language="VBScript">
        initialize
    </script>
    </body>
    </html>
```

Server-Side DOM Programming—Shopping Cart

We will now move over to the server side to illustrate how the shopping cart is received and how the information is extracted from the XML packet sent to it through the HTTP protocol. We will then generate a string as a response to the client. We will name our ASP script, process.asp.

For our purposes there is no need to dwell further on the processing of the shopping cart's content, such as updating sales database table, generating invoices, and payment, which are no doubt essential in a real-life e-commerce transaction.

As such, we can simplify our ASP script to perform only the following:

- Sets the content type of the response to be generated to the client

- Creates a DOM instance and loads it with the shopping cart's contents

- Retrieves userID attribute information and total number of orders—each selected item is considered one order regardless of the quantity specified for the same selected item

- Generates an acknowledgement string

Before we end this section, we will also mention an important issue that involves validating XML content received by the server.

Setting the Content Type

The purpose of setting the content type is to insert a content-type field into the header of the HTTP response message to be sent to the client, specifying the type of content the client is receiving.

If our intention is to generate a string of plain text to be sent back to the client, we can use the following line to set the content type:

```
<% Response.ContentType = "text/plain" %>
```

If the application may send back either plain text or HTML content, then we should specify the following content type:

```
<% Response.ContentType = "text/html" %>
```

If we wish to send XML content back to the client, we should do the following:

```
<% Response.ContentType = "text/xml" %>
```

The content types just described are not exhaustive. It really depends on what you want to send back to the client as a response.

> **NOTE**   *Response is an object specified in the ASP program-ming model for generating and sending response messages to the client.*

Receiving a DOM Object from the Client

We need to create a DOM instance and load it with the incoming XML content from the ASP Request object as illustrated in the following code:

```
<%
    Dim receivedDoc
    Set receivedDoc = Server.CreateObject (MSXML2.DOMDocument)
    receivedDoc.async = false
    receivedDoc.load Request
%>
```

After the loading is completed, we would have created a DOM tree of the shopping cart at the server side. We can apply the same programming techniques used at the client side to manipulate the tree.

Retrieving the Shopping Cart's Contents

This is a simple task as it involves only appropriate traversal of the DOM tree (receivedDoc) to access the information we would like to have.

Based on our requirements specification, we need to get the user ID and find out the number of orders made.

The first task is straightforward as we can simply use the selectSingleNode() method and specify the XPath expression for the userID attribute node in the document. To obtain the second piece of information, we must first collect a list of <ORDER> nodes, which can easily be done using the selectNodes() method. We can then use the length property of the node list to find out the number of orders the user had just made.

The code for achieving the retrieval of the userID attribute node and the <ORDER> node list is shown here:

```
<%
'Retrieving user ID
    Set idNode = receivedDoc.selectSingleNode("//@userID")
    userid = idNode.nodeValue
'Retrieving orders
    Set orderNodes = receivedDoc.selectNodes("//ORDER")
%>
```

Next, we will generate a response that makes use of this information we have just retrieved.

Generating a Response to the Client

For illustration purposes, we will generate an acknowledgement string, which will be displayed at the client side using a message box (decided by the client script), as shown in Figure 4-21.
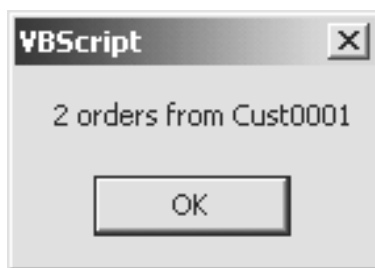
Figure 4-21. Client display of sample response string generated by the server

Here is the code for generating the short string as shown in the preceding illustration :

```
<%
    if orderNodes.length < 2 then
        Response.write orderNodes.length & " order from " & userID
    else
        Response.write orderNodes.length & " orders from " & userID
    end if
%>
```

Complete Listing of the Server-Side ASP Script

Listing 4-3 shows the source code of the server-side script process.asp, in which you can see the complete listing instead of fragments:

Listing 4-3. Process.asp

```
<%
'Setting content type
    Response.ContentType = "text/html"

'Creating a new DOM object and loading contents from the ASP's Request object
    Set receivedDoc = CreateObject("MSXML2.DOMDocument")
    receivedDoc.async = false
    receivedDoc.load Request

'Retrieving userID and the orders
    Set idNode = receivedDoc.selectSingleNode("//@userID")
    userid = idNode.nodeValue
    Set orderNodes = receivedDoc.selectNodes("//ORDER")

'Generating acknowledgement string to user (i.e., client)
    if orderNodes.length < 2 then
        Response.write orderNodes.length & " order from " & userID
    else
        Response.write orderNodes.length & " orders from " & userID
    end if
%>
```

Useful Web Links

- DOM Level 1 at

  http://www.w3.org/TR/REC-DOM-Level-1/

- DOM Level 3 Core Specification Working Draft at

  http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/

- Mathematical Markup Language at

  http://www.w3.org/TR/MathML2

- Basic DOM interfaces for Scalable Vector Graphics at

`http://www.w3.org/TR/SVG/svgdom.html`

- DOM interfaces for Synchronized Multimedia Integration Language at

`http://www.w3.org/TR/smil-boston-dom/`

- Xerces – DOM Paser at

`http://www.w3.org/TR/2000/REC-xml-20001006`

- Oracle XML Parser for Java v2 at

`http://technet.oracle.com/tech/xml/parser_java2/`

- XML DOM Objects/Interfaces supported in MSXML 3.0 at

```
http://msdn.microsoft.com/library/default.asp?url=/library/
  en-us/xmlsdk30/htm/xmmscxmldomobjects.asp
```

Summary

In this chapter, we introduced DOM programming by illustrating the properties and methods of the different types of interfaces that may be used to make accessing and manipulation of the XML data in an XML document possible. We also presented a mini-case study in which a shopping cart was created and manipulated on the client side, and illustrated how the server could receive and retrieve an XML DOM object from the client.

The concepts that we covered in this chapter should enable you to jumpstart an application using DOM programming. Since we are unable to examine all the APIs for DOM programming in one chapter, we encourage you to refer to the list in Appendix B. We also hope you find the list of links that we provided in the preceding section useful.