Document Object Model

Objectives

To understand what the Document Object Model is
To understand and be able to use the major DOM features
To use JavaScript to manipulate an XML document
To use Java to manipulate an XML document
To become familiar with DOM-based parsers

Introduction

DOMs are to manipulate the contents of an XML document.

XML documents, when parsed, are represented as a hierarchical tree structure in memory. This tree structure contains the document's elements, attributes, content, etc. XML was designed to be a live, dynamic technology - a programmer can modify the contents of the tree structure, which essentially allows the programmer to add data, remove data, query for data, etc. in a manner similar to a database.

The W3C provides a standard recommendation for building a tree structure in memory for XML documents called the *XML Document Object Model (DOM)*. Any parser that adheres to this recommendation is called a *DOM-based parser*. Each element, attribute, **CDATA** section, etc., in an XML document is represented by a *node* in the DOM tree. For example, the simple XML document

```
<?xml version = "1.0"?>
<message from = "Paul" to = "Tem">
   <body>Hi, Tem!</body>
</message>
```

results in a DOM tree with several nodes. One node is created for the **message** element. This node has a *child node* that corresponds to the **body** element. The **body** element also has a child node that corresponds to the text **Hi, Tem!.** The **from** and **to** attributes of the **message** element also have corresponding nodes in the DOM tree.

A DOM-based parser *exposes* (i.e., makes available) a programmatic library - called the *DOM Application Programming Interface* (API) - that allows data in an XML document to be accessed and modified by manipulating the nodes in a DOM tree.

Portability:

The DOM interfaces for creating and manipulating XML documents are platform and language independent. DOM parsers exist for many different languages, including Java, C, C+ +, Python and Perl.

Another API - JDOM-provides a higher-level API than the W3C DOM for working with XML documents in Java. Because JDOM is an API that is specific to the Java programming language, it can take advantage of features in Java that make it easier to program. JDOM is still in the early stages of development (visit *www.jdom.org* for more information on the JDOM API.

In order to use the DOM API, programming experience is required. Although the DOM API is available in many languages (e.g., C, Java, VBScript, etc.), JavaScript and Java will be emphasized.

DOM Implementations

DOM-based parsers are written in a variety of programming languages and are usually available for download at no charge. Many applications (such as Internet Explorer 5) have built-in parsers. Example 1 lists six different DOM-based parsers that are available at no charge:

Parser	Description
JAXP	Sun Microsystem's *Java API for XML Parsing*: **java. Sun. com/xml**
XML4J	IBM's *XML Parser for Java*: **www.alphaworks.ibm.com/tech/xml4j**
Xerces	Apache's *Xerces Java Parser*: **xml.apache.org/xerces**
msxml	Microsoft's *XML parser* (version 2.0) is built-into *Internet Explorer 5.5*, version 3.0 is also available: **msdn.microsof t.com/xml**
4DOM	*4DOM is a parser* for the Python programming language: **fourthought.com/4Suite/4DOM**
XML::DOM	*XML::DOM is a Perl module*: **Www-4.ibm.com/software/developer/library/xml-perl2**

DOM with JavaScript

To introduce document manipulation with the XML Document Object Model, a simple scripting example that uses JavaScript and Microsoft's msxml parser is introduced. This example takes an XML document (example 2) that marks up an article and uses the DOM API to display the document's element names and values. Example 3 lists the JavaScript code that manipulates this XML document and displays its content in an HTML page.

```
<?xml version = "1.0"?>
<article>
<title>Simple XML</title>
<date>December 6, 2000</date>
<author>
<fname>Tem</fname>
<lname>Nieto</lname>
</author>
<summary>XML is pretty easy.</summary>
<content>Once you have mastered HTML,
    XML is easily learned. You must remember
    that XML is not for displaying information but
    for managing information.
</content>
</article>
```

Example 2. Article marked up with XML tags.

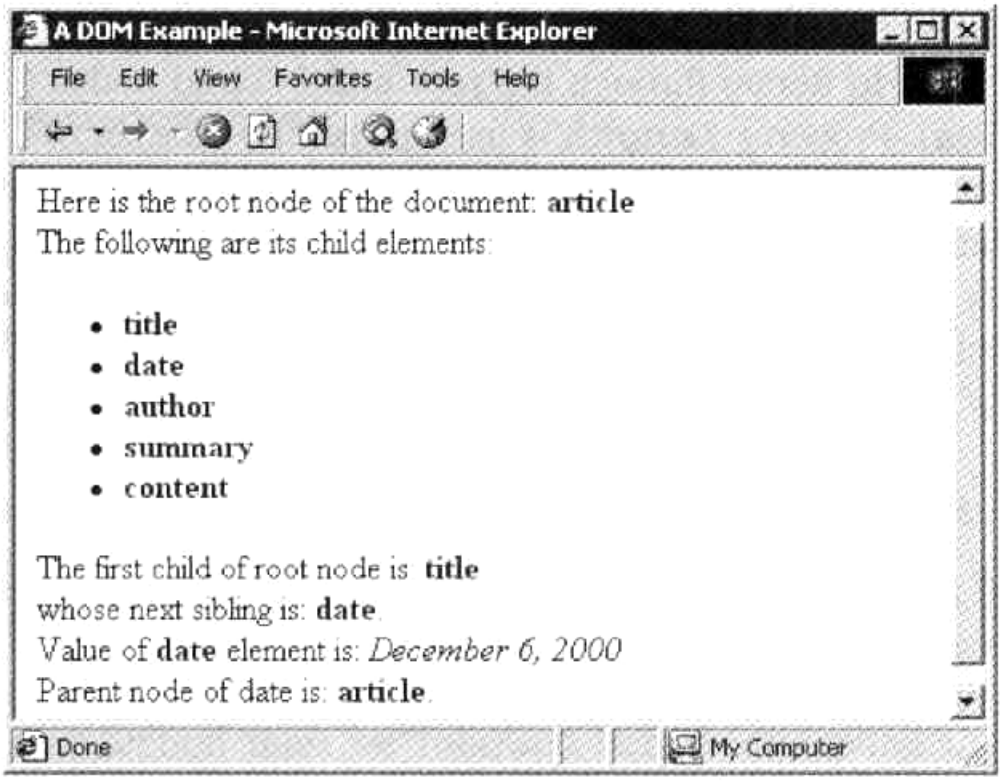
Traversing the article of example 2 (file *article.xml* with Javascript:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>A DOM Example</title>
```

```
</head>
<body>
<script type = "text/javascript" language = "JavaScript">
var xmldocument = new ActiveXObject( "Microsoft.XMLDOM");
xmldocument.load( "article.xml" );
var element = xmlDocument.documentElement;
document.writeln( "<p>Here is the root node of the document:" );
document.writeln("<strong>"+element.nodeName + "</strong>" );
document.writeln( "<br>The following are its child elements:" );
document.writeln( "</p><ul>" );
 for ( i = 0; i < element.childNodes.length; i++ ) {
 var curnode = element.childNodes.item( I );
 document.writeln( "<li><strong>" + curNode.nodeName + "</strong></li>" );
 }
document.writeln( "</ul>" );
var currentnode = element.firstchild;
document.writeln( "<p>The first child of root node is:" );
document.writeln( "<strong>" + currentNode.nodeName + "</strong>" );
document.writeln( "<br>whose next sibling is:" );
var nextsib = currentNode.nextSibling;
document.writeln("<strong>"+nextSib.nodeName + "</strong>" );
document.writeln( "<br>Value of <strong>" + nextSib.nodeName + "</strong>" element is:" );
var value = nextSib.firstChild;
document.writeln( "<em>" + value.nodevalue + "</em>" );
document.writeln( "<br>Parent node of " );
document.writeln( "<string>" + nextSib.nodeName + "</strong> is:" );
document.writeln( "<strong>" + nextSib.parentNode.nodeName + "</strong>.</p>" );
```

</script>
</body>
</html>

Example 3: Traversing *article.xml* with Javascript



The explanation of Javascript text:

<script type = "text/javascript" language = "JavaScript">

is the opening ***script*** tag, which allows the document author to include scripting code. Attribute **type** indicates that the script element is of media type **text:/javascript.** JavaScript is the most popular client-side (e.g., browser) scripting language used in industry. If the browser does not support JavaScript, **script's** contents are treated as text. Attribute **language** indicates to the browser that the script is written in the ***JavaScript:*** scripting language.

var xmlDocument = new ActiveXObject("Microsoft.XMLDOM");

instantiates a Microsoft XML Document Object Model object and assigns it to reference **xmlDocument.** This object represents an XML document (in memory) and provides methods for manipulating its data. The statement simply creates the object, which does not yet refer to any specific XML document.

xmlDocument.load("article.xml");

calls method **load** to load **article.xml** into memory. This XML document is parsed by msxml and stored in memory as a tree structure.

var element = xmlDocument.documentElement;

assigns the root element (i.e., **article)** to variable **element.** Property **documentElement** corresponds to the document's root element. The root element is important because it is used as a reference point for retrieving child elements, text, etc.

document.writeln(," + element.nodename + ,");
place the name of the root element in a **strong** element and write it to the browser where it is rendered. Property ***nodename*** corresponds to the name of an attribute, element, etc. which are collectively called nodes. In this particular case, **element** refers to the root node named **article.**

for (i = 0; i < element.childNodes.length; i++) {

uses a **for** loop to iterate through the root node's child nodes (accessed using property ***childNodes).*** Property **length** is used to get the number of child nodes of the document element.

Individual child nodes are accessed using the **item** method. Each node is given an integer index (starting at zero) based on the order in which they occur in the XML document. For example in example 2 **title** is given the index 0, **date** is given the index 1, etc.

var curNode = element.childNodes.item(i);

calls method **item** to return the child node identified by the index **i.** This node is assigned to variable **curNode.**

var currentnode = element.firstchild;

retrieves the root node's first child node (i.e., **title)** using property **firstChild.** This expression is a more concise alternative to

var currentnode = element.childNodes.item(0);

Nodes at the same level in a document (i.e., that have the same parent node) are called *siblings.* For example, **title, date, author, sunmiary** and **content** are all sibling nodes. Property ***nextSibling*** returns a node's next sibling.

var nextSib = currentNode.nextSibling;

assigns **currentNode's** (i.e., **title**) next sibling (i.e., **date)** to **nextSib.**

In addition to elements and attributes, text (e.g., **Simple XML**) is also a node.

var value = nextSib.firstChild;

assigns **nextSib's** (i.e., **date)** first child node to **value.** In this case, the first child node is a text node. The **nodeValue** method retrieves the value of this text node. The value of a text node's value is the text it contains. Element nodes have a value of **null** (i.e., the absence of a value).

document.writeln("" + nextSib.parentNode.nodeName + ".</p>");

retrieve and display **nextSib's** (i.e., **date)** parent node (i.e., **article).** Property **parentNode** returns a node's parent node.

Setup

In successive sections, Java applications to illustrate the DOM API will be used. The software needed to run these Java applications are presented. To be able to compile and execute the examples, it is needed to do the following:

- Download and install the Java 2 Standard Edition from **www.java.sun.com/j2se**

For step-by-step installation instructions, visit

www.deitel.com/faq/java3install.htm

- Download and install JAXP from **java.sun.com/xml/download.html.**

Installation instructions are provided at the Web site and HTML files are included with the download. Examples are also available for download from

www.deitel.com

The steps outlined in this section must be followed before attempting to execute any example.

DOM Components

Java, JAXP and the XML-related Java packages described in example 4 will be used to manipulate an XML document. Before discussing our first Java-based example, summary of several important DOM classes, interfaces and methods will be given. Due to the number of DOM objects and methods available, this is only a partial list of these objects and methods.

For a complete list of DOM classes and interfaces, browse the HTML documentation **(index.html** in the **api** folder) included with JAXP.

Class/Interface	Description
Document interface	Represents the XML document's top-level node, which provides access to all the document's nodes-including the root element.
Node interface	Represents an XML document node.
NodeList interface	Represents a read-only list of **Node** objects.

Element interface	Represents an element node. Derives from **Node.**
Attr interface	Represents an attribute node. Derives from **Node.**
Character Data interface	Represents character data. Derives from **Node.**
Text interface	Represents a text node. Derives from **CharacterData.**
Comment interface	Represents a comment node. Derives from **CharacterData.**
Processing Instruction interface	Represents a processing instruction node. Derives from **Node.**
CDATA Section interface	Represents a **CDATA** section. Derives from **Text.**

Table 4: DOM classes and interfaces.

The **Document** interface represents the top-level node of an XML document in memory and provides a means of creating nodes and retrieving nodes. Table 5 lists some **Document** methods.

Table 6 lists the methods of class **XmlDocument,** including the methods inherited from **Document.** Class **XmlDocument** is part of the JAXP internal APIs and its methods are not part of the W3C DOM recommendation.

Interface **Node** represents an XML document node. Table 7 lists the methods of interface **Node.**

Method Name	Description
CreateElement	Creates an element node.
CreateAttribute	Creates an attribute node.
CreateTextNode	Creates a text node.
CreateComment	Creates a comment node.
CreateProcessingInstructio	Creates a processing

n	instruction node.
CreateCDATASection	Creates a **CDATA** section node.
GetDocumentElement	Returns the document's root element.
AppendChild	Appends a child node.
GetChildNodes	Returns the child nodes.

Table 5: Some **Document** methods.

CreateXmlDocument	Parses an XML document.
Write	Outputs the XML document.

Table 6: XmlDocument methods.

Appendchild	Appends a child node.
Clonenode	Duplicates the node.
Getattributes	Returns the node's attributes.
GetChildNodes	Returns the node's child nodes.
GetNodeName	Returns the node's name.
GetNodeType	Returns the node's type (e.g., element, attribute, text, etc.)
GetNodeValue	Returns the node's value.
GetParentNode	Returns the node's parent.
HasChilciNodes	Returns **true** if the node has child nodes.
Removechild	Removes a child node from the node.
Replacechild	Replaces a child node with another node.
SetNodeValue	Sets the node's value.
Insertbefore	Appends a child node in front of a child node.

Table 7 Node methods.

Table 8 lists some node types that may be returned by method **getNodeType.** Each type in table 8 is a **static f inal** *constant) member of class* **Node. Element** represents an element node. Table 9 lists some **Element** methods.

Node type	Description
Node.ELEMENT NODE	represents an element node
Node.ATTRIBUTE NODE	represents an attribute node
Node.TEXT NODE	represents a text node
Node.COMMENT-NODE	represents a comment node
Node.PROCESSING-INSTRUCTION NODE	represents a processing instruction node
Node.CDATA SECTION NODE	represents a CDATA section node

Table 8. Some node types.

Method name	**Descripton**
getAttribute	Returns an attribute's value.
getTagName	Returns an element's name.
removeAttribute	Removes an element's attribute.
setAttribute	Sets an attribute's value.

Table 9: Element methods

Internet and World Wide Web Resources

www.w3.org/DOM
W3C DOM home page.

www.w3schools.com/dom
The W3Schools DOM introduction, tutorial and links site.

www.oasis-open.org/cover/dom.html
The Oasis-Open DOM page contains a comprehensive overview of the Document Object Model with references and links.

dmoz.org/Computers/Prograna ning/Internet/W3C_DOM
This is a useful set of DOM links to different locations and instructional matter.

www.w3.org/DOM/faq.html
Answers to Frequently Asked DOM Questions.

www.jdom.org
Home page for the JDOM XML API in Java.

Summary

- XML documents, when parsed, are represented as a hierarchal tree structure in memory. This tree structure contains the document's elements, attributes, text, etc. XML was designed to be a live, dynamic technology - the contents of the tree structure can be modified by a programmer. This essentially allows the programmer to add data, remove data, query for data, etc., in a manner similar to a database.
- The W3C provides a standard recommendation for building a tree structure in memory for XML documents called the XML Document Object Model (DOM). Any parser that adheres to this recommendation is called a DOM-based parser.
- A DOM-based parser exposes (i.e., makes available) a programmatic library-called the DOM Application Programming Interface (API) that allows data in an XML document to be accessed and manipulated. This API is available for many different programming languages.
- DOM-based parsers are written in a variety of programming languages and are usually available for download at no charge. Many applications (such as Internet Explorer 5) have built-in parsers.
- A Microsoft XML Document Object Model object (i.e., ***Microsoft.XMLDOM*)** represents an XML document (in memory) and provides methods for manipulating its data.
- Property **documentelement** returns a document's root element. The root element is important because it is used as a reference point for retrieving child elements, text, etc.
- Property **nodename** returns the name of an attribute, element, etc.-which are collectively called nodes.
- Property **childnodes** contains a node's child nodes. Property **length** returns the number of child nodes.
- Individual child nodes are accessed using the **item** method. Each node is given an integer value (starting at zero) based on the order in which they occur in the XML document.
- Property **firstchild** retrieves the root node's first child node.
- Nodes at the same level in a document (i.e., that have the same parent node) are called siblings. Property **nextsibling** returns a node's next sibling.
- A text node's value is its text, an element node's value is **null** (which indicates the absence of a value) and an attribute node's value is the attribute's value.
- Property **parentnode** returns a node's parent node.
- The **Document** object represents the top-level node of an XML document in memory and provides a means of creating nodes and retrieving nodes.
- Interface **Node** represents Lin XML document node.
- **Element** represents an element node.
- Sun Microsystems, the creator of Java, provides several packages related to XML. Package **org.w3c.dom** provides the DOM-API programmatic interface (i.e., classes, methods, etc.). Package javax.xml.parsers provides classes related to parsing an XML document. Package **com.sun.xml.tree** contains classes and interfaces from Sun Microsystem's internal API, which provides features (e.g., saving an XML document) currently not available in the DOM recommendation.
- A DOM-based parser may use an event-based implementation (i.e., as the document is parsed events are raised when starting tags, attributes, etc. are encountered) to help create the tree structure in memory. A popular event-based implementation is called the Simple API for XML (SAX). Package **org. xml. sax** provides the SAX programmatic interface.
- Class **DocumentBuilderFactory** (package **javax. xmi. parsers)** obtains an instance of a parser.
- Method **setvalidating** specifies whether a parser is validating or nonvalidating.

- Method **parse** loads and parses XML documents. If parsing is successful, a **Document** object is returned. Otherwise, a **SAXException** is thrown.
- Method **getDocumentElement** returns the **Document's** root node. The **Document's** root node represents the entire document - not the root element node.
- Method **getNodeType** retrieves the node's type.
- Elements in the XML document are retrieved by calling method **getElementsByTagName**. Each element is stored as an item (i.e. a **Node** in a **NodeList**. The first item added is stored at index 0, the next at index 1, and so forth. This index is used to access an individual item in the **NodeList.**
- Interface **Text** represents an element or attribute's character data.
- Method **replacechild** replaces a **Node.**
- Method **write** is a member of **XmlDocument,** which requires casting a **Document** to **XmlDocument.** This internal API class is used because **Document** does not provide a method for savin, **an** XML document.
- **SAXParseException** and **SAXException** contain information about errors and warnings thrown by the parser. Class **SAXParBeException** is a subclass **ol'SAXException** and includes methods for locating the location of the error.
- By default, JAXP does not throw any exceptions when a document fails to conform to a DTD. The programmer must provide their own implementation, which is registered using method **setErrorHandler.**
- Interface **ErrorHandler** provides methods **fatalError, error** and **warning** for fatal errors (i.e., errors that violate the XML 1.0 recommendations parsing is hafted), errors (e.g., such as validity constraints that do not stop the parsing process) and warnings (i.e., not classified as fatal errors or errors and that do not stop the parsing process), respectively.
- Method **newDocument** creates a new **Document** object, which can he used to build an XML document in memory.
- Method **createComment** creates a comment.
- Method **createProcessingInstruction** creates a processing instruction and method **createCDATASection** creates a **CDATA** section.