REALIZEDENERGY

www.noonetime.com

Microsoft XML Programming

XML, ASP, and the DOM

XML 2001 Tutorial – Sunday December 9, 2001

Noonetime

Steve Schwedland

(steve@noonetime.com)

+1-303-291-0230

NOONETIME

Introduction

Agenda

This course will cover the following topics:

– Brief overview of XML with an emphasis on element relationships.
– VBScript and ASP Basics.
– DOM properties and methods.
– Real world examples.

Learning Objectives

- Understand the DOM approach of tree-based processing.
- Create XML and HTML from XML sources using Active Server Pages in combination with the DOM.
- Be able to use the basic DOM approach for more complex processing.

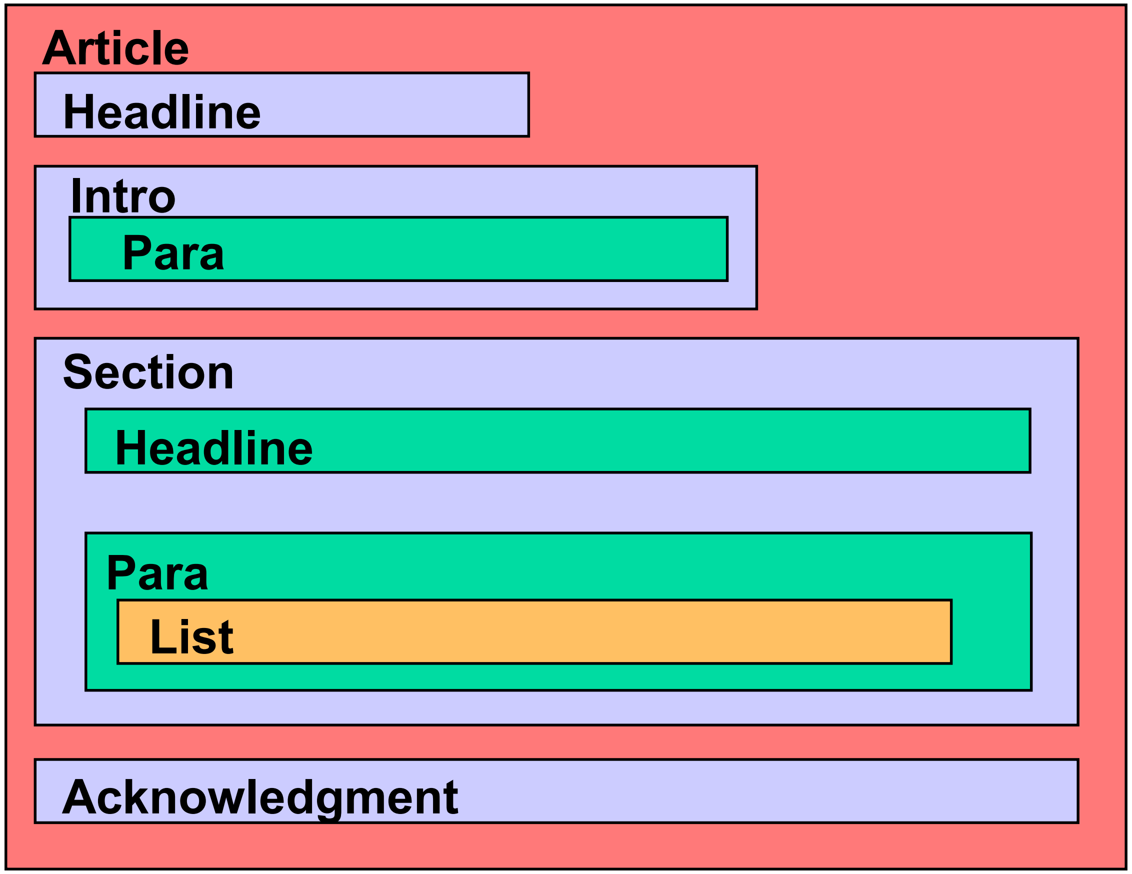
XML Syntax – Brief Overview

An XML Document is…

- A collection of parsed and unparsed pieces, called “entities”.
- Text and markup in seven-bit ASCII.
- Valid, or at least well-formed.

Ukrainian Dolls

Containers of Containers



According to the Rules…

- One unique root element.
  - The root element contains all other elements.
- Elements in the root occur sequentially and/or nested.
  - Elements follow one another, or appear inside one another, but may not overlap.
- All elements must have a start tag and an end tag.
  - Begin with <MyContainerName>
  - End with </MyContainerName>

A Note on Names

- XML is case-sensitive
  - <Para>, <PARA>, <para>, and <PaRa> all mark different elements.
- Element names
  - Must begin with a letter, an underscore (_), or a colon (:).
  - Names should not begin with “XML”; all case variations are reserved for the standard.
  - May then include any combination of letters, digits, periods (.), hyphens (-), underscores (_), or colons (:).

EMPTY Elements

From section 3.1 of the XML Recommendation:

– Have no content (nothing between the start tag and end tag).
– Most often used as point markers in the document, usually carrying information in attributes.
– May be indicated by:

```
<NameOfEmpty/>
```

OR

```
<NameOfEmpty></NameOfEmpty>
```

Attributes

- Information about the Information.
- Carry information about the element.
  - Information about figures.
  - IDs, authors, security levels.
- Must appear in the start tag, in the form:

```
<Tag MyAttribute="value">
```

OR

```
<Tag MyAttribute='value'>
```

Elements vs. Attributes

Elements
- May have sub-parts.
- May occur more than once.
- Order can be enforced.
- Parser does not check content for constraints.

Attributes
- May not have sub-parts.
- May not occur more than once per element.
- Order cannot be enforced.
- Parser does check content for constraints; some typing possible.

Character Entities

Character entities are:

– Used to represent parts of an extended character set, or characters that might otherwise “break” an XML document.

– Expressed as an “&” followed by the entity name and ending with a “;” for example…

Ω Omega symbol from the Symbol font character set.
&OHgr; from ISO Greek character set.

Σ Summation symbol from the Symbol font character set.
&sum; from the ISO Added Math Symbols character set.

Predefined Character Entities

For well-formed documents:

&	&amp;	(ampersand)
'	&apos;	(apostrophe)
>	&gt;	(greater than)
<	&lt;	(less than)
"	&quot;	(double quote)

Valid documents must explicitly define these entities before using them.

Processing Instructions

- Allow documents to contain instructions for applications.
- Not part of the document structure, but must be passed through to the application.
- Made of two parts; target and value.

```
<?Target OpenNewWindow?>
```

Comments

- Information for humans; should be ignored by the XML processor.
- Can be used to add additional information to DTDs and documents.

```
<article>
   <!--This isn't really necessary-->
   <para>Surfing is easy and fun.</para>
</article>
```

The DTD – Brief Overview

The DTD

- A DTD defines the structural rules for your data.
- Allows you to check your data against those rules using a parser.
- A collection of declarations:
  - Elements
  - Attributes
  - Entities
  - Notations

Element Declarations - Examples

```
<!ELEMENT book (title, Chapter+)                        >

<!ELEMENT book (title?, (para* | Section*) )            >

<!ELEMENT book (figure | graphic | table | list)+ >

<!ELEMENT book (title, subtitle?,
                 ((para | list | graphic)*,
                          (Group+ | Section+)))     >
```

Attribute Lists

Three things to define for every attribute:

```
<!ATTLIST Story
   copyright   NMTOKEN                                #REQUIRED
   keyword     CDATA                                  #IMPLIED
   type        (original | revised | adapted) "original"
   dtd-version NMTOKEN                                #FIXED "1.0"
   filename    ENTITY                                 #IMPLIED
   target      ID                                     #REQUIRED  >
```

Attribute name

List of possible values or an attribute type keyword

Default value or default value keyword

Implications of the Attribute Type

ID and IDREF

- Standard interfaces give the ability to process an element with a particular ID based on a found IDREF.

NOTATION

- Tells the parser *not* to try and read the contents of the element it's used on but also-
- Gives you an idea of how to process the contents of the element based on the associated notation declaration.

Implications of the Default Value

- Default values can be built into the data structure, not your application.
- #IMPLIED: Your application might need to assume a value.
- Enforces some basic checking at the parser level so you don't have to in your application.

ASP and VBScript

ASP

- ASP (Active Server Pages) is a language for creating interactive web sites that was created by Microsoft. It was designed to be easier to use, faster to develop in, and to perform better than Perl/CGI.

- Can use either VBScript or JScript as the programming language.

VBScript

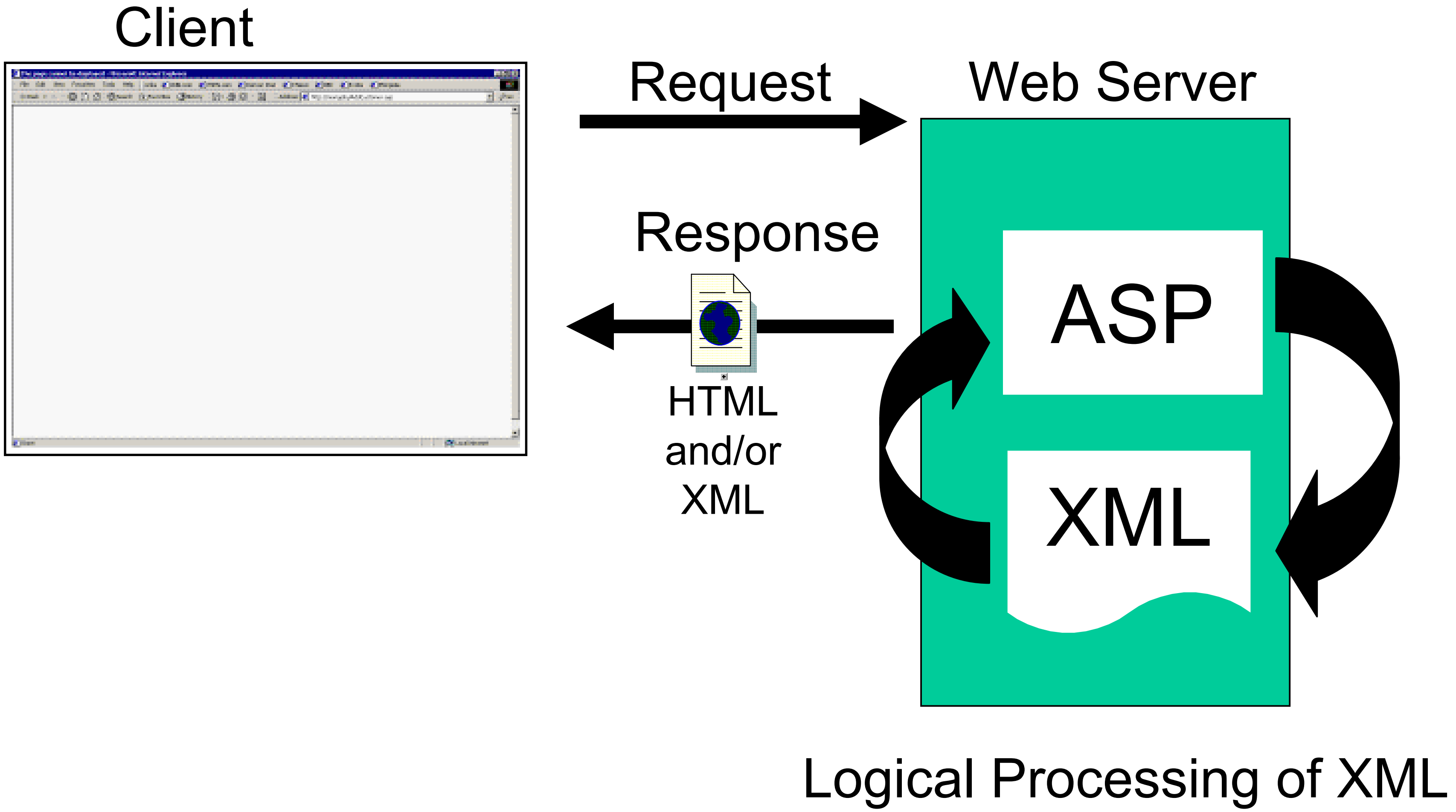
VBScript…

- is a scaled down version of Visual Basic.
- is a powerful, easy to learn tool that can be used to add interaction to your web pages.
- brings active scripting to a wide variety of environments, including web client scripting in Internet Explorer and web server scripting in Microsoft Internet Information Server.

Putting it all together…

REALIZEDENERGY

www.noonetime.com



XSL versus ASP

XSL(T), like ASP...

- Works on the DOM model of your data.
- Good for XML to XML operations.
- Can use on the server or client side to filter, rearrange and modify XML.

However...

- Not a regular expression, general purpose programming language.
  - Designed primarily for the very specific task of transforming XML to XML.
  - No dynamic variables.
  - Other limitations.

Standard Programming Interfaces

SAX and DOM

SAX = Simple API for XML

DOM = Document Object Model

- Both are standard interfaces for working with XML.
- Both "expose" XML to your application as a collection of objects with properties.
- Platform and language-neutral interfaces that allow programs and scripts to access the content and structure of XML documents.

SAX

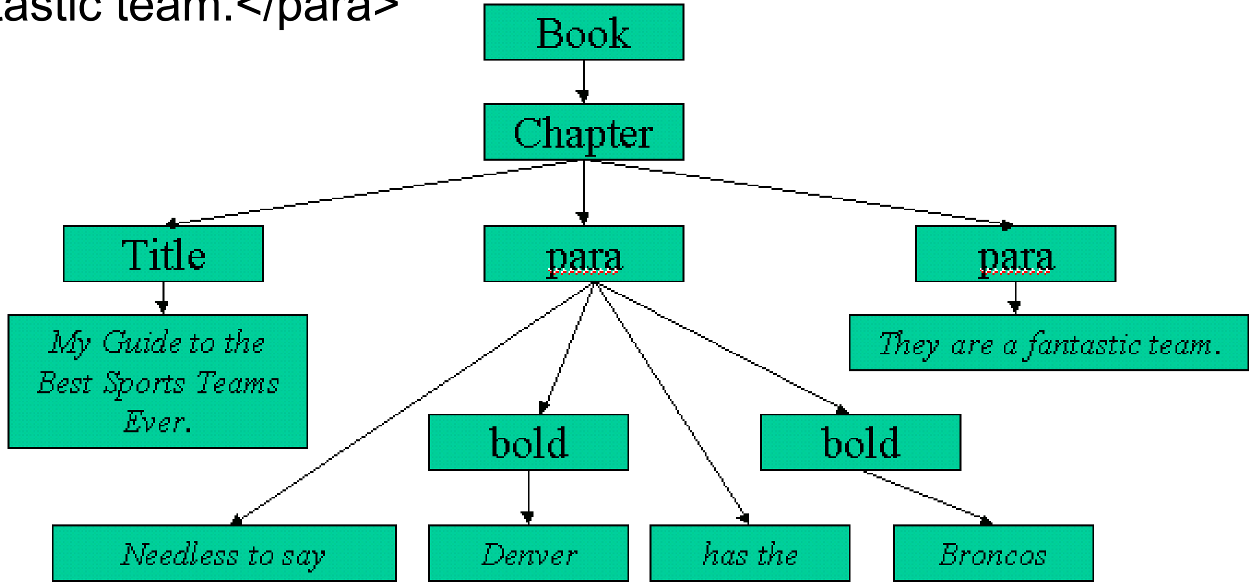
- Delivers data to your application as a stream of events.
  - "I found an element".
  - "I found an attribute and here is it's value".
  - "I found a processing-instruction".
- Event driven.
- Relatively resource efficient.
- Good choice for conversion of XML to other formats.

DOM

- Delivers data to your application as tree structure that represents the document.
- Allows access to the entire tree at once.
  - Query for arbitrary XML structures.
  - Easily rearrange a structure into a modified one.
- Relatively less efficient with system resources.
- Good choice when you need to create XML in your application, or modify an existing XML structure.

DOM Tree

```
<Book author="Joe Cool">
  <Chapter topic="sports">
    <title ID="X234Y">My Guide to the best sports teams ever.</title>
    <para>Needless to say <bold>Denver</bold>, has the
       <bold>Broncos</bold>.</para>
    <para>They are a fantastic team.</para>
  </Chapter>
</Book>
```



DOM Implementation Issues

- While the DOM offers a standard set of interfaces to XML, different DOM implementations can add proprietary extensions.

VB Script Review

Blockers

- <%…%> Blockers are used to delimit code in your ASP page.
- Code contained in these tags will be executed.
- Code not contained in these tags will be considered part of the literal result to the client.

Example:

```
<% @Language=VBScript %>
<html>
    <% ASP Code - VBScript %>
</html>
```

Comments

```
'This is a comment
```

- Everything to the right of the apostrophe will be ignored.
- No block commenting in VBScript; each line must start with an apostrophe.

Variables

- Implicit vs. Explicit declarations.
- Dim statement:

```
Dim Name
Dim Name, Address, City, State
```

- Variable naming rules:
  - Must begin with an alphabetic character.
  - Cannot contain embedded periods.
  - Must be unique within the same scope.
  - Must be no longer than 255 characters.

Assigning Variables

Use the following format to assign a value to a variable:

```
Name = “Steve Schwedland"
HoursWorked = 50
Overtime = True
```

Conditional Statements

- If…Then…Else
  - Used to choose one of the identified options based on a series of conditional tests, or a default if none of the tests are true.
  - Tests only one condition.
- Select…Case
  - Similar to if..then..else except it allows multiple conditions to be tested.

If...Then...Else

```
If *condition* Then
    *statements*
ElseIf *condition-n* Then
    *elseif statements*
Else
    *else statements*
End If
```

If…Then…Else

```
If x=0 Then
   response.write "You have no bananas."
ElseIf x=1 Then
   response.write "You have a banana."
Else
   response.write "You have " & x & " bananas."
End If
```

Select...Case

```
Select Case *testExpression*
    Case *expressionList-n*
        *statements-n*
    Case Else
        *elsestatements-n*
End Select
```

Select…Case

```
Select Case HairColor
    Case "Blonde"
        Call BlondeRoutine
    Case "Bald"
        Response.Write “You don’t have any hair!<BR>”
        Response.Write “Deal with it!<BR>”
    Case Else
        Call OtherHairColor
End Select
```

Looping Statements

- For…Next
  - Repeat a group of statements a set number of times.
- For Each…Next
  - Similar to For...Next except particularly useful for stepping through the elements of an array or collection.
- Do…Loop
  - Do While and Do Until.
  - Repeat a group of statements based on a condition that is tested with each iteration of the loop.

For...Next

```
For counter = start To end [Step step]
    statements
          Exit For
    statements
Next
```

For...Next

```
For I = 1 To 10
   For J = 1 To 10
        For K = 1 To 10
                . . .
        Next
   Next
Next
```

For Each...Next

```
For Each element In group
    statements
          Exit For
    statements
Next element
```

For Each...Next

```
Dim fso, f, f1, fc, str
Set fso=CreateObject("Scripting.FileSystemObject")
Set f=fso.GetFolder(folderspec)
Set fc=f.Files
For Each f1 in fc
   str = str & f1.name
   str = str & "<BR>"
Next
Response.Write str
```

Do…Loop

```
    Do {While | Until} condition
        statements
            Exit Do
        statements
    Loop
OR
    Do
        statements
            Exit Do
        statements
    Loop {While | Until} condition
```

Do...Loop

```
Do Until Resp = vbNo
    MyNum = Int (6 * Rnd + 1)
    Resp = MsgBox (MyNum & " Another number?", vbYesNo)
Loop
```

Creating a DOM Object

DOM Objects

- The four main objects exposed by the XML DOM are:
  - DOMDocument
  - XMLDOMNode
  - XMLDOMNodeList
  - XMLDOMNamedNodeMap
- Each of these objects exposes methods and properties that enable you to:
  - gather information about the instance of the object.
  - manipulate the value and structure of the object.
  - navigate to other objects within the tree.

Creating a DOMDocument Object

- The document can be created using one of two possible types.
  - Free-threaded
  - Rental-threaded model
- Rental-threaded documents exhibit better performance because the parser doesn't need to manage concurrent access among threads.
- Note that you cannot combine nodes or documents that are created using differing threading models.

Creating a DOMDocument Object (continued)

The document threading model is determined by the following settings (MSXML 3.0):

Setting Version	Rental-threaded model	Free-threaded model
Version independent ProgID	Msxml2.DOMDocument	Msxml2.FreeThreadedDOMDocument
ProgID	Msxml2.DOMDocument.3.0	Msxml2.FreeThreadedDOMDocument.3.0
ClassID	F6D90F11-9C73-11D3-B32E-00C04F990BB4	F6D90F12-9C73-11D3-B32E-00C04F990BB4
VB Class Name	DOMDocument30	FreeThreadedDOMDocument30

Creating a DOMDocument Object

The following example demonstrates how to create the two types of objects.

In VBScript:

```
Set objDoc = CreateObject("Msxml2.DOMDocument")
```

Or

```
Set objFTDoc = CreateObject("Msxml2.FreeThreadedDOMDocument")
```

Setting DOM Properties

Getting and Setting Parse Flags

The DOMDocument object exposes four properties that allow the user to change parsing behavior at run time:

- async
- validateOnParse
- resolveExternals
- preserveWhiteSpace

async Property

- The async property indicates whether asynchronous download is permitted.

```
objXMLDOMDocument.async = boolValue
```

- Set the value to True if asynchronous download is permitted, Otherwise set the value to False. The default is True.

Example:

```
xmlDoc.async = False
```

validateOnParse Property

- The validateOnParse Property Indicates whether the parser should validate this document.

```
objXMLDOMDocument.validateOnParse = boolValue
```

- If True, it validates during parsing. If False, it parses only for well-formed XML. The default is True.

Example:

```
xmlDoc.validateOnParse = True
```

resolveExternals Property

REALIZEDENERGY

www.noonetime.com

- The resolveExternals Property indicates whether external definitions are to be resolved at parse time, independent of validation.
  - Default attribute values and types from DTD.
  - Data types from an XML Schema.

```
objXMLDOMDocument.resolveExternals = boolValue
```

- When set to True, external definitions are resolved at parse time.

Example:

```
xmlDoc.resolveExternals = True
```

NOONETIME

preserveWhiteSpace Property

- Specifies the default white space handling.

  `objXMLDOMDocument.preserveWhiteSpace = boolVal`

- When True, all white space is preserved, regardless of any xml:space attributes. Same as having xml:space= "preserve" attribute on every element.
- When False, any xml:space attributes determine what space is preserved. Default value is False.

Example: `xmlDoc.preserveWhiteSpace = True`

Putting it together…

Here is the example code we have so far:

```
' Declare a variable and set it to an MSXML document object.
Dim objDoc
Set objDoc = CreateObject("Msxml2.DOMDocument")

' Set Parse Flags.
objDoc.async = False
objDoc.validateOnParse = False
objDoc.resolveExternals = False
objDoc.preserveWhiteSpace = True
xmlDoc.load("c:\myBook.xml") ' All ready to load some XML!
```

Reading in XML

Loading XML From a File

To load an XML file use the load method:

```
boolValue = XMLDOMDocument.load(url)
```

where *url* is a string containing the location of the XML file. The file can be local or on the web.

```
objDoc.load("http://www.example.com/reports.xml")
objDoc.load("C:\MyFile.xml")
objDoc.load(Server.MapPath("CDMusic.xml"))
```

The load method returns True if the load was a success and False otherwise.

Loading XML From a String

You can also load an XML file as a string using the loadXML method:

```
boolValue = objDoc.loadXML(string)
```

Where *string* is the actual XML that you want to load.

Example:

```
objDoc.loadXML("<author><first_name>Joe</first_name>
<last_name>Cool</last_name></author>")
```

Construct XML From an ADO

- Read data from a database.
- Build an XML document.
- Load it into a DOM object.

XML From ADO Example

```
DIM adoConnection
DIM adoRecordSet
Set adoConnect= Server.CreateObject("ADODB.Connection")
adoConnect.open "ADOTest"  ', "user_name", "user_pass"
Set adoRecordSet = adoConnect.Execute("SELECT * FROM Class")

Dim myXMLString
myXMLString = "<class>"
Do While Not adoRecordSet.EOF
    myXMLString = myXMLString & "<student>" + adoRecordSet("First") + "</student>"
    adoRecordSet.MoveNext
Loop
myXMLString = myXMLString & "</class>"
Response.ContentType = "application/xml"
Response.Write myXMLString
adoRecordSet.Close
set adoRecordSet = Nothing
```

Working With the DOM Object

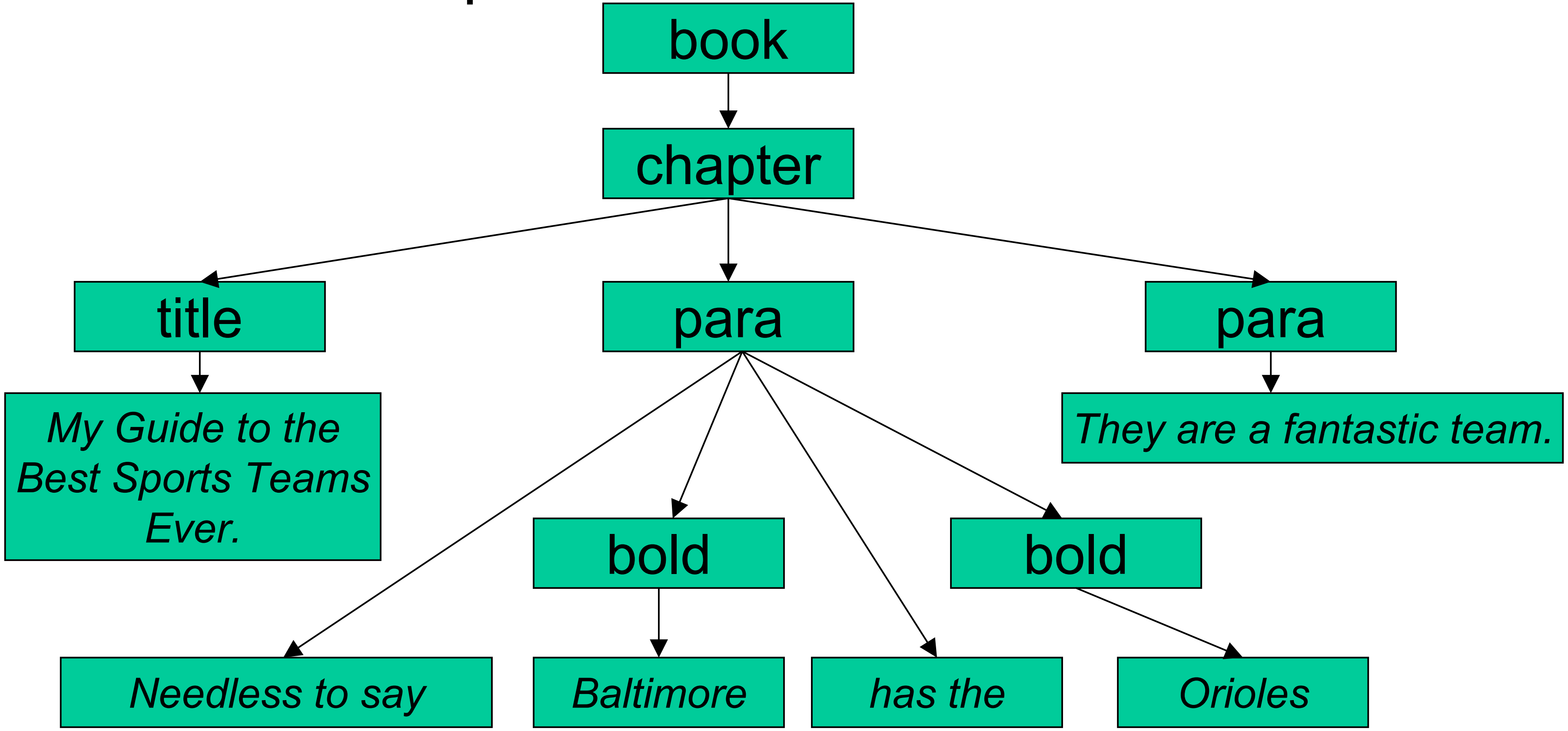
Accessing the Document Tree

In the DOM, documents have a logical structure which is very much like a tree. Consider the following XML structure for mybook.xml

```
<book author="Joe Cool">
  <chapter topic="sports">
    <title ID="X234Y">My Guide to the best sports teams ever.</title>
    <para> Needless to say <bold>Baltimore</bold> has the
       <bold>Orioles</bold>.</para>
    <para>They are a fantastic team.</para>
  </chapter>
</book>
```

Accessing the Document Tree (continued)

The DOM represents the XML like this:



Accessing the Document Tree (continued)

- You can access the tree for an XML document in two ways.
  - By beginning at the root element and walking down the tree or
  - By querying for a specific node or collection of nodes.
- You can navigate to the root element of the document by using the documentElement property. This property returns the root element as an XMLDOMNode object.

Accessing the Tree by the root element Example

You can walk the tree as in the following example using VBScript:

```
Dim root, xmlDoc, child
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load(Server.MapPath("CDMusic.xml"))
'Set root to the XML document's root element
Set root = xmlDoc.documentElement  'Walk the tree
For Each child in root.childNodes
   Response.Write "&lt;" & child.nodeName & "&gt;" & child.text
   Response.Write "&lt;/" & child.nodeName & "&gt;"
Next
```

Creating New Nodes

The DOMDocument object exposes methods to create nodes in one of two ways.

- The standard eight methods, that follow the standard Document Object Model (DOM) specification.
- The createNode method, which allows you to create a qualified node by supplying the node type, name, and an optional namespaceURI of the new node.

Standard DOM Methods

DOMDocument also exposes the following eight methods, which allow you to create specific Document Object Model (DOM) nodes.

- CreateAttribute
- CreateCDATASection
- CreateComment
- CreateDocumentFragment
- CreateElement
- CreateEntityReference
- CreateProcessingInstruction
- CreateTextNode

Standard DOM Method Example

```
Dim xmlDoc
Dim root
Dim newElem
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load (Server.MapPath("books.xml"))
Set root = xmlDoc.documentElement
Response.Write root.childNodes.item(1).xml
Response.Write "<BR/><BR/>"
Set newElem = xmlDoc.createElement("PAGES")
root.childNodes.item(1).appendChild newElem
root.childNodes.item(1).lastChild.text = "400"
Response.Write root.childNodes.item(1).xml
```

createNode Method

- Creates a node.
- Example:

```
Set NewNode = DOMObj.createNode(Type, name, namespaceURI)
```

  - NewNode is the name of variable to create the node in.
  - DOMObj is the name of the document object the node is being created for.
  - Type, name and namespaceURI are the parameters that define the new node.
- Return Value is the newly created node object.

Parameters

Type

– Variant. Value that uniquely identifies the node type. This can be specified using either the integer value or the string value.

Name

– String containing the value for the new node's nodeName property.

NamespaceURI

– String defining the namespace URI.

Creating New Nodes (continued)

The relationship between the node *name* and node *type* is summarized below.

NODE_ATTRIBUTE	The name of the attribute.
NODE_CDATA_SECTION, NODE_COMMENT, NODE_DOCUMENT, NODE_DOCUMENT_FRAGMENT, NODE_TEXT	The nodeName property for these node types is a constant value; the *name* parameter is ignored.
NODE_DOCUMENT_TYPE	The name of the document type; for example, the xxx in <!DOCTYPE xxx ...>.
NODE_ELEMENT	The name of the XML tag, with any namespace prefix included if present.
NODE_ENTITY	The name of the entity.
NODE_ENTITY_REFERENCE	The name of the entity referenced. Note that the name does not include the leading ampersand or the trailing semicolon. The name includes the namespace if one is present.
NODE_NOTATION	The name of the notation.
NODE_PROCESSING_INSTRUCTION	The target—the first token following the <? characters

Using the XMLDOMNode Object

Gathering Information About a Node

The following properties allow you to get information about the node:

- hasChildNodes
- namespaceURI
- nodeName
- nodeType
- nodeTypeString
- parsed
- specified
- xml

Gathering Information About a Node (continued)

You can access these properties to get information concerning:

- the type of the node,
- the name of the node,
- whether the node has any children,
- whether the node is explicitly specified in the XML file or implied within the Document Type Definition (DTD) or XML Schema,
- the namespace to which the node belongs,
- whether the node has been parsed,
- and the string representation of the node.

hasChildNodes Method

- Returns True if this node has children.

```
boolValue = oXMLDOMNode.hasChildNodes()
```

- Example:

```
Dim currNode
Set currNode = xmlDoc.documentElement
If currNode.hasChildNodes Then
     Response.Write "<H1>Length: " & currNode.childNodes.length & "</H1>"
Else
    Response.Write "no child nodes"
End If
```

namespaceURI

- Returns the URI (universal resource identifier) for the namespace.

```
strValue = oXMLDOMNode.namespaceURI
```

- This refers to the "uuu" portion of the namespace declaration xmlns:nnn="uuu".

nodeName Property

- Contains the qualified name of the element, attribute, or entity reference, or a fixed string for other node types.

```
strValue = oXMLDOMNode.nodeName
```

- Example:

```
Dim currNode
Set currNode = xmlDoc.documentElement
Response.Write currNode.nodeName
```

nodeType Property

- Specifies the XML DOM node type, which determines valid values and whether the node can have child nodes.

```
lValue = oXMLDOMNode.nodeType
```

- It indicates the type of the node. Use the nodeTypeString property to return the node type in string form.

nodeTypes

A node may be one of the following types:

- NODE_ELEMENT (1)
- NODE_ATTRIBUTE (2)
- NODE_TEXT (3)
- NODE_CDATA_SECTION (4)
- NODE_ENTITY_REFERENCE (5)
- NODE_ENTITY (6)
- NODE_PROCESSING_INSTRUCTION (7)
- NODE_COMMENT (8)
- NODE_DOCUMENT (9)
- NODE_DOCUMENT_TYPE (10)
- NODE_DOCUMENT_FRAGMENT (11)
- NODE_NOTATION (12)

nodeType Property (Example)

The following VBScript example creates an **XMLDOMNode** object and displays its type enumeration (in this case 1, for NODE_ELEMENT):

```
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.load(Server.MapPath("CDMusic.xml"))
xmlDoc.async = False
Dim currNode
Set currNode = xmlDoc.documentElement.childNodes.item(0)
Response.Write "Node Type: " & currNode.nodeType
```

nodeTypeString Property

- Returns the node type in string form.

```
strValue = oXMLDOMNode.nodeTypeString
```

- It contains the string version of the node type. To return the enumeration value, use the nodeType property.

nodeTypeString Property Types

The string name for the node types are:

NODE_ATTRIBUTE	attribute
NODE_CDATA_SECTION	cdatasection
NODE_COMMENT	comment
NODE_DOCUMENT	document
NODE_DOCUMENT_FRAGMENT	documentfragment
NODE_DOCUMENT_TYPE	documenttype
NODE_ELEMENT	element
NODE_ENTITY	entity
NODE_ENTITY_REFERENCE	entityreference
NODE_NOTATION	notation
NODE_PROCESSING_INSTRUCTION	processinginstruction
NODE_TEXT	text

nodeTypeString Property Example

- The following VBScript example creates an XMLDOMNode object and displays its node type in string form (in this case, "element"):

```
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.load(Server.MapPath("CDMusic.xml"))
xmlDoc.async = False
Dim currNode
Set currNode = xmlDoc.documentElement.childNodes.item(0)
Response.Write "Node Type String: " & currNode.nodeTypeString
```

nodeTypeString Property Example 2

- The following sample code tests to see if the node is of type "element."

```
if (xmlNode.nodeTypeString == "element")
        XMLDoc.loadXML(xmlNode.xml)
```

- If the node is of type "element," the string representation of the node is loaded into a document object through the loadXML method.

parsed Property

- Contains True if this node and all descendants have been parsed and instantiated; False if any nodes remain to be parsed.

```
boolValue = oXMLDOMNode.parsed
```

- During asynchronous access, not all of the document tree may be available. Before performing some operations, such as XSL transformations or pattern-matching operations, it is useful to know whether the entire tree below this node is available for processing.

parsed Property Example

- The following VBScript example displays whether or not the top-level node (root) and all its descendants are parsed:

```
Dim xmlDoc
Dim root
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = True
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Set root = xmlDoc.documentElement
Response.Write root.parsed
```

specified Property

- Indicates whether the node (usually an attribute) is explicitly specified or derived from a default value in the DTD or schema.

  boolValue = oXMLDOMNode.specified

- Returns:
  - True if the attribute is explicitly specified in the element.
  - False if the attribute value comes from the DTD or schema.

specified Property Example

- The following example creates an XMLDOMNode object from the specified item in an XMLDOMNamedNodeMap. It then displays whether or not the attribute was specified in the element rather than in a DTD or schema:

```
Dim xmlDoc, currNode, objNamedNodeMap, myNode
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Set currNode = xmlDoc.documentElement.childNodes.item(0)
Set objNamedNodeMap = currNode.attributes
Set myNode = objNamedNodeMap.item(0)
Response.Write myNode.specified
```

xml Property

- Contains the XML representation of the node and all its descendants.

```
strValue = oXMLDOMNode.xml
```

- Note that the xml property always returns a Unicode string.

xml Property (continued)

- The xml property for the DOMDocument object converts the document from its original encoding to Unicode. As a result, the original encoding attribute is removed.
- For example,

```
<?xml version="1.0" encoding="UTF-8"?>
```

appears in the xml property as:

```
<?xml version="1.0"?>
```

xml Property Example

- The following example creates an XMLDOMNode object (of type NODE_ENTITY), and then displays the object's XML value (including that of any of the object's child nodes):

```
Dim xmlDoc
Dim currNode
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Set currNode = xmlDoc.documentElement.childNodes.item(0)
Response.Write currNode.xml
```

Getting and Setting Data Within a Node

- The data marked up within the XML file is exposed in the DOM as node values. These values might be the value of an attribute, for instance, or the text within an XML element.
- There are a few ways to access the value of a node.
  - nodeValue
  - Using the text property
  - Access through the nodeTypedValue

nodeValue

- The nodeValue property gives you access to values of certain nodes. The nodes that contain a value that can be accessed through the nodeValue property are attributes, text nodes, comments, processing instructions, and CDATA sections.
- The following code sets the value of an attribute to "hello world":

```
newAttNode = XMLDoc.createAttribute("newAtt")
newAttNode.nodeValue = "hello world"
```

nodeValue and text property

- Navigate down to the element's children (the text nodes within) and call nodeValue on them. Then call the text property.
- This property returns the text within the element.
- The following sample code tests to see if the text within the element elem1 is equal to "hello world." If it is, it assigns that element the text "hi world."

```
if (elem1.text = "hello world") then
    elem1.text = "hi world"
end if
```

Navigating the Tree

- From the XMLDOMNode object, you can navigate directly to:
  - parent node (parentNode)
  - children (childNodes, firstChild, lastChild)
  - siblings (previousSibling, nextSibling)
  - the document object to which the node belongs (ownerDocument).

Navigating the Tree (Cont.)

- If the node is of type element, attribute, or entityReference, you can call the definition property to navigate to the schema definition of the node.
- In addition, if the node is of type element, processingInstruction, documentType, entity, or notation, you can navigate to the attributes on the node using the attributes property.

parentNode, firstChild, lastChild

- Create a DOMNode object from another node object's parent and displays its XML.

```
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Dim currNode
Dim newNode
Set currNode = xmlDoc.documentElement.childNodes.Item(1).childNodes.Item(0)
Set newNode = currNode.parentNode

Response.ContentType = "application/xml"
' Comment one of the two lines below and compare output.
'Response.Write currNode.xml
Response.Write newNode.xml
```

childNodes

- Contains a node list containing the children nodes.

```
Set currNode = xmlDoc.documentElement.childNodes.Item(1)
Set newNode = currNode.childNodes
```

ownerDocument

- Returns the root of the document that contains the node.

```
Set currNode = xmlDoc.documentElement.childNodes.Item(0).childNodes.Item(1)
Set owner = currNode.ownerDocument
```

Using the XMLDOMNodeList Object

Gathering Information About the Node List

- You can get the length of the node list from the length property. Among other things, the length can be used for iterating through the list of children.
- This example checks to see if the current node has children, if so it gets the number of children and outputs them to a message box.

```
Dim currNode
Set currNode = xmlDoc.documentElement
If currNode.hasChildNodes Then
    Response.Write currNode.childNodes.length
End if
```

Navigating the Node List

- You can get at a specific member of the node list by using the item method.
  - The item method takes a number corresponding to a node's position within the node list.
  - To get the first node in the node list, for instance, one would call the following method: item(0).
- It is also possible to navigate through the node list using the nextNode method.
  - The nextNode method returns the next node in the node list.
  - If the current node is the last node in the list or the node list has no members, nextNode returns null.

nextNode

- Returns the next node in the collection.
- Useful for iterating through a collection.

```
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Dim objNodeList, objNode
Set objNodeList = xmlDoc.getElementsByTagName("track")
For i = 0 To (objNodeList.length - 1)
   Set objNode = objNodeList.nextNode
   Response.Write objNode.Text & "<BR/>"
Next
```

Using the XMLDOMNamedNodeList Object

Navigating the Named Node Map

- Just like the XMLDOMNodeList object, the XMLDOMNamedNodeMap object exposes the length property.

- This property returns the number of members in the named node map.

Navigating the Named Node Map (continued)

- As with a node list, you can access members of the named node map by index using the item method. However, you can also access the members of a named node map name by using getNamedItem and getQualifiedItem.
- The getNamedItem method takes the name of the desired node as a parameter, whereas, the getQualifiedItem method takes the name and namespaceURI of the desired node. Each method returns an XMLDOMNode object.
- The XMLDOMNamedNodeMap object, like the node list object, exposes the nextNode property.

Using Collections

- In the XML DOM, collections are used to store lists of nodes or elements.
- Unlike an array, a collection is dynamic, meaning that the addition or removal of nodes, and changes within nodes, are immediately reflected in the collection.
- You can access the collections by two different ways:
  – By using the Item method or
  – Iterating through the list

Accessing Collection using Item

- The following code returns the first member of the collection:

```
objNodeList.item(0)
```

- As a shortcut, you can access a node by simply appending the appropriate index number in parentheses to a collection reference, like so:

```
objNodeList(0)
```

Iterating through Collections

- In Visual Basic, you can iterate through an XML DOM collection with a **For Each…Next** statement, like so:

```
Dim Item
For Each Item in objNodeList
    Response.Write Item.text
Next
```

- Or you can use a **For…Next** statement:

```
Dim i
Dim members
members = objNodeList.length
For i = 0 To (members-1)
   Response.Write objNodeList(i).xml
Next
```

XML DOM Persistence

- Information stored in an XML Document Object Model (DOM) can be serialized to an XML file, and information stored in an XML file can be opened into a DOM.
- For scripting purposes, persistence is handled by the following methods and property:
  - load method
  - loadXML method
  - save method
  - xml property

Retrieving Document Properties

Gathering Document Information

- The following properties allow you to retrieve information about the document: doctype, implementation, parseError, readyState, and url.
- You can access these properties to retrieve the:
  - Document Type Declaration (DTD) accompanying the document
  - implementation for the document
  - last error to occur while parsing
  - information about the state of the XML document
  - URL, if any, of the XML file being loaded and parsed.

doctype Property

- Contains the document type node that specifies the DTD for this document.

  `objXMLDOMDocumentType = oXMLDOMDocument.doctype`
- For XML, it points to the node of type NODE_DOCUMENT_TYPE that specifies the document type definition (DTD). It returns Null for XML documents without a DTD and for HTML documents.
- An XML document can contain a document type declaration before the first element in the document. It starts with the tag <!DOCTYPE> and can specify an external DTD.

doctype Property Example

- The following VBScript example creates an XMLDOMDocumentType object, and then displays the name property of the object:

```
Dim xmlDoc
Dim MyDocType
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load("c:\mybook.xml")
Set MyDocType = xmlDoc.doctype
MsgBox (MyDocType.name)
```

implementation Property

- Contains the XMLDOMImplementation object for this document.

```
objXMLDOMImplementation = oXMLDOMDocument.implementation
```

- An XML DOM application can use objects from multiple implementations. This property provides access to the XMLDOMImplementation object that handles this document.
- The XMLDOMImplementation object provides methods that are independent of any particular instance of the document object model.

implementation Property Example

- The following VBScript example creates an XMLDOMImplementation object:

```
Dim xmlDoc
Dim i
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load("c:\mybook.xml")
Set i = xmlDoc.implementation
```

parseError Property

- Returns an XMLDOMParseError object that contains information about the last parsing error.

```
objXMLDOMParseError = oXMLDOMDocument.parseError
```

- The XMLDOMParseError object returns detailed information about the last parse error, including the error number, line number, character position, and a text description.

parseError Property Example

- The following VBScript example uses the document's **parseError** property to create an **XMLDOMParseError** object. It then tests the error and displays a message if one occurs:

```
xmlDoc.load(Server.MapPath("CDMusic.xml"))
Set myErr = xmlDoc.parseError
If (myErr.errorCode <> 0) Then
    Response.Write "A parse error occurred on line "
    Response.Write xmlDoc.parseError.Line
    Response.Write " at position " & xmlDoc.parseError.linepos
End If
```

readyState Property

- Indicates the current state of the XML document.

  lValue = oXMLDOMDocument.readyState

- It returns a value that indicates the instantiation and download state of the XML document object. The value can be one of the following:
  - LOADING (1) The load is in progress
  - LOADED (2) Reading of the persisted properties completed
  - INTERACTIVE (3) Some data has been read and parsed, and the object model is now available on the partially retrieved data set.
  - COMPLETED (4) The document has been completely loaded, successfully or unsuccessfully.

url Property

- The url PropertyReturns the canonicalized URL for the last loaded XML document.

  strValue = oXMLDOMDocument.url

- It returns the URL from the last successful load. If the document is being built in memory, this property returns Null.
  - NOTE: The **url** property is not updated after saving a document with the save method. To update the **url** property in this case, reload the document using the load method

url Property Example

- The following VBScript example creates a DOMDocument object and displays the value of its url property:

```
Dim xmlDoc
Set xmlDoc = CreateObject("Msxml2.DOMDocument")
xmlDoc.async = False
xmlDoc.load("c:\mybook.xml")
MsgBox (xmlDoc.url)
```

Writing Out XML

Saving XML

- To save an XML file use the load Method:

```
objXMLDOMDocument.save(objTarget)
```

- Where objTarget is a string containing the location to save the XML file.

Loading and Saving XML Example

- The following VBScript example creates a DOMDocument object and uses the load method to load a local XML file:

```
Dim xmlDoc
Set xmlDoc = CreateObject(“Msxml2.DOMDocument")
xmlDoc.load("c:\myBook.xml")
```

- The document object, XMLDoc, now contains a tree consisting of the parsed contents of reports.xml.
- And to save the XML file use the Save method:

```
xmlDoc.save(Server.MapPath(“myBook.xml"))
```

Summary

References

Microsoft's XML SDK Documentation

- msdn.microsoft.com/library/en-us/xmlsdk30/htm/xmmscxmloverview.asp

Microsoft's VBScript Reference

- msdn.microsoft.com/scripting/

World Wide Web Consortium

- www.w3.org

Good Programming Resources

- www.devguru.com
- www.topxml.com
- www.hotscripts.com

Thank You