Using XML in InduSoft Web Studio

Implementation Specifications or Requirements

Category	Item	
Software	IWS Version:	6.0 and later
	Service Pack:	N/A
	Windows Version:	WinXP/2000/NT and Windows CE
	Web Thin Client:	Yes
Equipment	Panel Manufacturer:	N/A
	Panel Model:	N/A
	Other Hardware:	N/A
	Comm. Driver:	All
	Controller (e.g.: PLC)	All
	Application Language:	N/A
Software Demo Application	N/A	

Summary

XML has become a ubiquitous technology standard used to transport data in a cross-platform environment. It is a companion technology to HTML, but one does not replace the other. HTML is a standard used for displaying data (e.g. web pages) whereas XML is a standard used for structuring and transferring data.

In an InduSoft Web Studio (IWS) application, Recipes can be saved to or loaded from a file that is in XML format. These XML recipe files can be created in a separate computer and made available to the PC (or PCs) using the Recipe Manager.

In addition to the Recipe Manager, IWS can create, read from, and write to XML files from a VBScript program. This is a powerful capability that can be utilized, for example, to transfer production data records (e.g. summary data, sample data) and reports from a PC running IWS to another PC.

This Application Note provides an overview of XML, discusses how IWS creates XML recipe files, and gives examples using VBScripting with Microsoft's XML Parser (XMLDOM), part of Microsoft's XML Core Services (MSXML), to create, load and manipulate XML files. An example is also included of converting a database file (Recipe file stored in Microsoft Excel format), read into a RecordSet using ADO.NET, into an XML file format that is compatible with InduSoft's Recipe Manager.

XML Overview

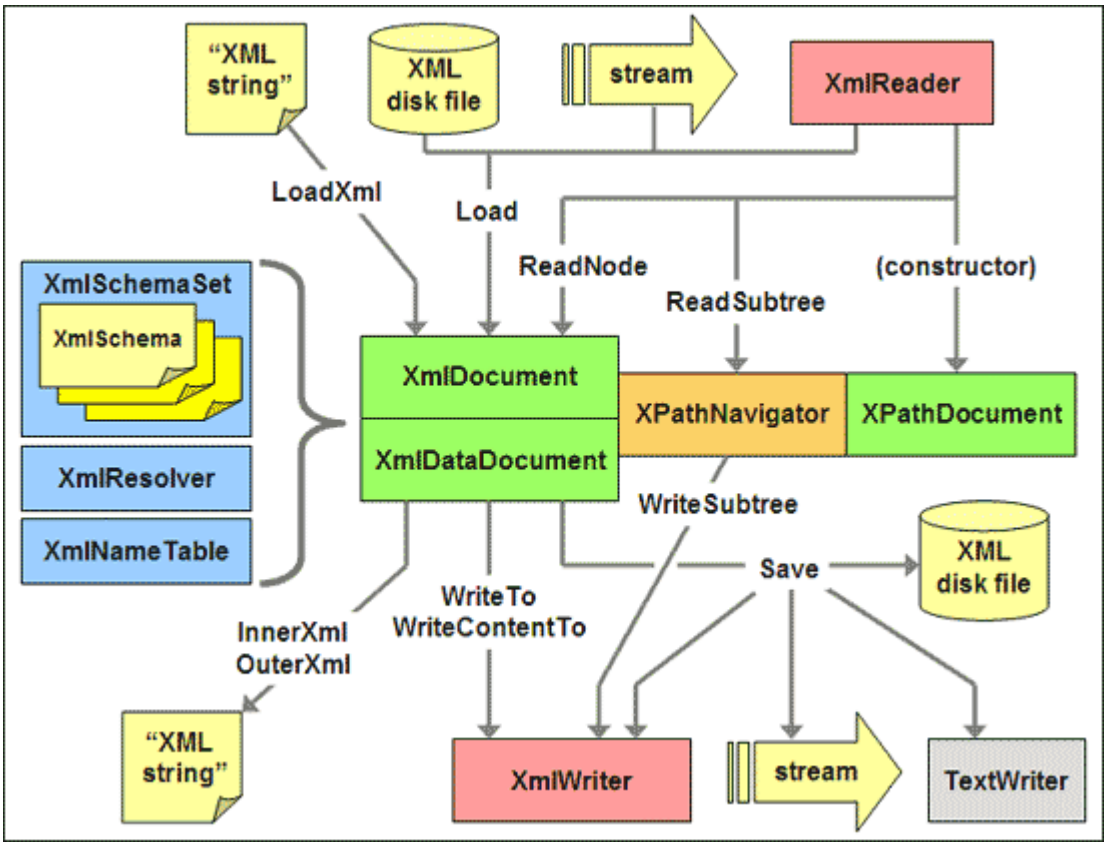
XML stands for E**X**tensible **M**arkup **L**anguage, a cross-platform language designed to describe data in a simple yet flexible text format intended for document publishing and data exchange applications. Today, XML is widely used for transmitting data over the Internet to XML compatible browsers such as Microsoft's Internet Explorer. XML is based on a suite of technical recommendations developed by working groups of the World Wide Web Consortium (W3C)[1]. Some of the components in the XML suite are:

XSD A XML schema definition language used to create XML schemas (e.g. used to validate XML documents, define data types)

Parser An API used to process XML data. Uses DOM or SAX.

XSLT Extensible Stylesheet Language Transformation that describes the formatting used to display XML data on a browser. XSLT can make XML data appear as if is HTML.

XPath Used by XSLT and other XML components to query and filter data in an XML document



Overview of XML
Source: http://www.15seconds.com/issue/050601.htm

[1] www.w3.org

It is important to understand that an XML document does not do anything by itself. It is merely structured data. An application program is required to make use of this structured data. Since its introduction in 1998, XML has become widely used in a number of areas, especially Web-based applications and document processing. Microsoft Office Products (e.g. Word, Excel) can save their files in XML format. Since an XML document contains data, it can also be used with XPath as a lightweight database. XPath is analogous to SQL and the XML document is analogous to the SQL database.

Since an IWS application is primarily concerned with using XML to load/store data (e.g. Recipe files, database exchange), we can limit the scope of our review of XML to a few areas; namely the XML language and the Parser. When used with an IWS application, XML data will be stored in files. This is true whether the XML data was created by the IWS Recipe Manager task, or by a VBScript program. While web applications can have XML data stored as "data islands" inside HTML pages, this is not how we will use XML.

The W3C defines processing (Parsing) APIs used to manipulate XML files, with DOM and SAX being the two most widely used APIs. DOM, or Document Object Model, is an object-oriented API that parses XML data into a well defined tree structure, and allows manipulation of its contents (load, store, as well as adding, changing or deleting elements). DOM requires that the XML data (a DOM document) be stored in memory, allowing access to the document's elements in a random manner. By contract, SAX, or Simple API for XML, is a set of APIs to access and manipulate XML documents in a sequential manner. Since XML files are simply text files, you do not need to use DOM or SAX to create or manipulate XML files, but they greatly simplify the task. For our purposes, we will focus on DOM since DOM provides the greatest flexibility. Microsoft provides services for both, however.

The Microsoft XML Core Services[2] (MSXML) is a collection of XML services to build XML applications that provide a high degree of interoperability with other applications that adhere to the W3C XML 1.0 standard. Strict adherence to the W3C recommendation is not practical, since the W3C recommendation does not include a standard function for loading XML files (big problem). Microsoft's MSXML has been periodically upgraded to be consistent with changes to W3C recommendations. The most recent release of MSXML is Version 6.0, released July, 2006. To insure interoperability with other Windows XP applications, MSXML Version 4.0 or later be sufficient. For Windows CE.NET and Windows CE 5.0 applications, the XML Core Services functionality is based on MSXML Version 3.0 Service Pack 1, a subset of MSXML Version 4.0 functionality[3]. So a review of the differences may be warranted.

IMPORTANT NOTES ABOUT MSXML

- **MSXML is Microsoft's XML Core Services**
- **Windows XP supports Versions 4, 5, & 6. Version 4 is widely used. If using XML to interact with a database such as Microsoft SQL Server 2005, verify which version of MSXML is required.**
- **Windows CE supports MSXML Version 3.0 SP1. This is a subset of Version 4.0 functionality. The MSDN website lists the differences between the Versions. If using XML to interact with a database, check with the database vendor to determine which version of MSXML is required.**

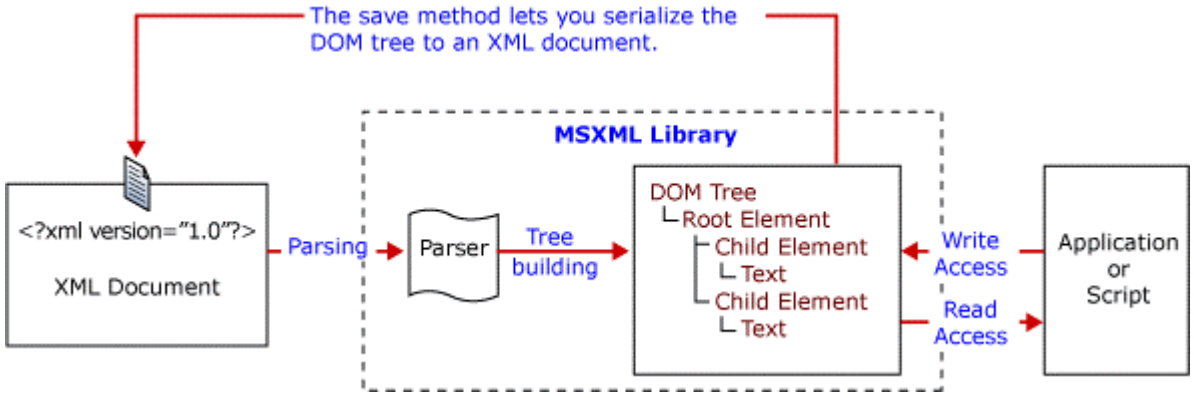
[2] See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/6139a6fe-7d4c-449d-9e36-60b68be1de71.asp
[3] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceappservices5/html/wce50orixmlcoreservicesanddocumentobjectmodel.asp

Microsoft's XML Parser is a COM object called XMLDOM, part of MSXML. XMLDOM is compatible with both IWS VBScript and Microsoft Internet Explorer (5.0 or later), and is compatible with W3C XML Version 1.0. Microsoft exposes the XMLDOM via COM interfaces in MSXML.dll. In VBScript, the XMLDOM object is instantiated by one of the following statements:

```
Set Mydoc = CreateObject("Microsoft.XMLDOM")            'Uses the latest version of the XMLDOM Parser

Set Mydoc = CreateObject("MSXML2.DOMDocument.4.0")  'Uses the MSXML 4.0 Parser SP2 (XMLDOM)
```

As a COM object, XMLDOM has properties, methods and events. The following diagram shows the tasks involved in parsing an XML document and presenting the information to an application or script.



An XML document is comprised of text data (letters, numbers, punctuation…anything that is within the bounds of Unicode) as well as binary data, and the markup that describes the data. The markup is enclosed in arrows (< and >), and valid markups include processing instructions, references, comments, and tags. The following letter from Jill to Jack illustrates how data might be structured in an XML file:

Example 1

```
<note>
    <to>Jack</to>
    <from>Jill</from>
    <heading>Reminder</heading>
    <body>Be careful when carrying the pail of water down the hill. Love, Jill.</body>
</note>
```

Just to be clear, an XML tag is not the same thing as an IWS tag. XML tags define the type of information contained within it, and are similar to an HTML tags except that XML tags are not predefined whereas HTML tags are. Since XML tags are not predefined, the tag definitions must be common between the producer and the consumer of the XML data. The combination of the start tag, the content, and the ending tag is called an element. In the above example, the XML file has sender and receiver information, as well as a heading and message body.

XML Syntax

Before we start programming XML applications in IWS, we need to know the syntax portion of the markup language (the ML of XML). The key components are:

- Prolog
- Declaration
- Processing Instructions
- Comments
- Tags
- Attributes
- Elements
- Character Data
- Character and Entity References
- CDATASection
- Entities

Prolog

Example 1 shows an example of what would be an XML document. However, in a real XML file there may be additional information needed to be sent to the application processing the XML data. This information can be contained in an XML declaration and one or more XML processing instructions. The XML declaration and any processing instructions that refer to XML style sheets must be placed in the beginning of the XML file, before the XML document. This section is called the Prolog.

IMPORTANT NOTES ABOUT THE XML PROLOG

- **Located at the beginning of an XML file before the document section**
- **Can contain an optional XML declaration, optional processing instructions and comments.**

Declaration

The XML declaration is a specific type of processing instruction that allows you to specify which "version" of the W3C specification the markup language conforms to. The XML declaration is not required, but if it is used it must be the first line in the document and no other content or white space (blank spaces) can proceed it.

If you use the XML declaration, you must include the version attribute. The version attribute must appear first. W3C has defined XML versions 1.0 and 1.1, although 1.0 is much more widely used at this time. However, Microsoft currently supports only Version 1.0, so you must use this.

There is an optional Encoding attribute that can be specified, which tells the parser which character format to use (all XML Parsers support UTF-8 8-bit and UTF-16 16-bit Unicode formats). §4.3.3 of the W3C XML specification provides a list of encoding types. Some examples include:

"Shift-JIS"	Japanese Encoding method Shift-JIS
"ISO-8859-1"	ISO-8859-1 (Latin1/Western European)

There is an optional Standalone declaration attribute that indicates whether the XML document relies on information from an external source such as an external Document Type Definition (DTD). The values for this attribute are “yes” or “no” (the default value if the attribute is left out).

Examples of valid XML declarations are:

```
<?xml version=“1.0”?>
<?xml version=“1.0” encoding =“UTF-8” standalone =“yes”?>
```

IMPORTANT NOTES ABOUT XML DECLARATIONS

- **All XML declarations must begin with <? and end with ?>**
- **All attribute values must be in quotes, either single or double quotes. If the attribute value contains double quotes, then it must be enclosed in single quotes. If the attribute value contains single quotes, then it must be enclosed in double quotes.**

Processing Instructions

Processing instructions include specific information meant for the application that uses the XML document. Processing instructions are not used by the XML Parser and can appear anywhere in the XML document after the declaration (if used). However, processing instructions must appear outside of another markup. The first item in a processing instruction is called a target, which must begin with a letter or an underscore. The rest of the target name can consist of letters, digits, hyphens, underscores, periods and colons. After the target, any combination of valid XML text characters can appear.

Once specific type of processing instruction is the style sheet processing instruction. Style sheets are stored in a file with an xsl file extension. They are used by Microsoft Internet Explorer to define how XML data is to be displayed by the web browser. If a style sheet processing instructin is used, it must be placed before the XML document or root element but after the declaration (if there is one).

Style sheet processing instructions usually have the text attribute set to one of two values:

type = “text/css”	Style sheet is a CSS type (cascading style sheet)
type = “text/xsl”	Style sheet is an Extensible style sheet (XSL), which is the type used by the IWS Recipe Manager

An example of a style sheet processing instruction

```
<?xml-stylesheet type="text/xsl" href="recipe.xsl"?>
```

IMPORTANT NOTES ABOUT XML PROCESSING INSTRUCTIONS

- **All XML processing instructions must begin with <? and end with ?>**
- **Style sheet processing instructions must appear after the declaration and before the document or root element**
- **W3C recommends against using colons “:” in processing instruction names**
- **All attribute values must be in quotes, either single or double quotes. If the attribute value contains double quotes, then it must be enclosed in single quotes. If the attribute value contains single quotes, then it must be enclosed in double quotes.**

Comments

Comments are content not intended for the XML parser but are included as notes to improve readability or other purposes. Comments can be on one line or have many lines. All comments must begin with a <!-- and end with a -->. An example of an XML comment is:

```
<!--  This section contains the recipe IWS tags and tag values -->
        <TAG>
          <NAME>Cake.flour</NAME>
          <VALUE>4000</VALUE>
        </TAG>
        <TAG>
          <NAME>Cake.fruit</NAME>
          <VALUE>2000</VALUE>
        </TAG>
        <TAG>
          <NAME>Cake.milk</NAME>
          <VALUE>3000</VALUE>
        </TAG>
        <TAG>
          <NAME>Cake.sugar</NAME>
          <VALUE>4000</VALUE>
        </TAG>
        <TAG>
          <NAME>Cake.yeast</NAME>
          <VALUE>50</VALUE>
        </TAG>
<!--  This section is finished -->
```

IMPORTANT NOTES ABOUT XML COMMENTS

- **All XML comments must begin with <!-- and end with -->**
- **You should avoid using a "-" or a "--" in the text of the comment as it can be confusing to the Parser**
- **Never place a comment within a tag or an attribute value**
- **You can comment out blocks of XML elements with the use of a comment**

Tags

Not to be confused with IWS tags, all XML documents contain tags. These XML tags generally occur in pairs; i.e. a start tag and an end tag. Tags establish boundaries around data content. The start and end tags, along with any contained data content, comprise XML elements. The start tag gives a name to the element.

Start tags may have attributes in the tag name, but end tags may not. There is a special type of type of tag called an empty tag. An empty tag can be used when there is no textual content to the element. An empty element can be defined by a start tag and an end tag with no white spaces or text content in between, or by an empty tag.

Some tag syntax examples are:

<elementName>	A start tag with no attributes
<elementName atr1Name=”atr1val” atr2Name=”atr2val” …>	A start tag with attributes
</elementName>	An end tag
<elementName></elementName>	Empty element with start & end tags
<elementName/>	Empty element with empty tag

IMPORTANT NOTES ABOUT XML TAGS

- **The start tag defines the element name**
- **The start tag name is case sensitive and must start with a letter or underscore. It can contain letters, digits, hyphens, underscores and periods. Do not use colons in the tag name.**
- **Tag names occur in pairs, unless it is an empty tag**
- **A start tag may contain attributes, but an end tag may not**
- **An empty tag may contain attributes, but the element does not contain any contents**

Attributes

Attributes allow you to add information about an element that generally not considered the content of an element (although there are no hard rules for this). An attribute can also be used with processing instructions.

All attributes must have both a name and a value, i.e. a name-value pair. Each attribute name must be unique and like element names, and attribute names are case sensitive. Attribute names must start with a letter or underscore. The rest of the name can contain letters, digits, hyphens, underscores and periods. Attribute values can contain text, or be a character reference or an entity reference. However, attribute values cannot be an element markup or a CDATA section.

Attribute values must be enclosed in single or double quotes, but the same type of quotation mark must be used on each side on the attribute value. If the attribute value contains embedded double quotes, then it must be enclosed in single quotes or you need to use the &quot entity reference. If the attribute value contains embedded single quotes, then it must be enclosed in double quotes or you need to use the &apos entity reference.

<myElement contraction='isn't' />	Embedded single quotes
<myElement question="They asked "Why?"" />	Embedded double quotes

IMPORTANT NOTES ABOUT XML ATTRIBUTES

- **Attributes can appear in start or empty tags, but not in end tags.**
- **All attributes must have a name and a value. You cannot have a name without a value.**
- **Attribute names are case sensitive.**
- **Cannot have two attributes with the same name**
- **All attribute values must be in quotes, either single or double quotes. If the attribute value contains double quotes, then it must be enclosed in single quotes or you need to use the &quot entity reference. If the attribute value contains single quotes, then it must be enclosed in double quotes or you need to use the &apos entity reference. Can use different quotes on different attribute values within the same element.**
- **You cannot put a comment in an attribute value**
- **Data can be stored in either child elements or in attributes. However, attributes are not as flexible since they cannot contain multiple values and do not describe data structures.**

Elements

Elements form basic units of XML content. Elements consist of a start tag and an end tag, and everything in between the two. The text between the start and end tags is called “character data” while the text within the tags is called “markup”. Elements can also contain attribute names and values. These elements create structures that can be manipulated with programs or style sheets.

Element relationships can be formed between multiple elements in an XML document and are usually described in tree (root, branches, leaves) or family metaphors (parent, child, subchild, ancestor, descendant, sibling), or most often a combination of both. These metaphors refer to the relationships relative to each other, but not necessarily to the entire document. In the tree metaphor, leaves refer to elements at the end of a branch that do not contain other elements. The following shows two examples of a relationship between elements which are identical:

Tree Structure

```
<a>
  <b>
    <c>
      <d/>
      <e/>
      <f/>
    </c>
  </b>
</a>
```

Non-tree structure

```
<a><b><c><d/><e/><f/></c></b></a>
```

Using family metaphors, the <c> element is a parent of <d>, <e> and <f>, who in turn are siblings or child elements of <c>. In turn, <c> is a child element of <b>, and <b> is the parent of element <c>. The <a> element is commonly called the root element, switching to the tree metaphor. All XML documents are required to have at least one root element. The root element is often referred to as the document tag, and follows the prolog. The root element must be a non-empty tag that encompasses the entire document.

XML elements can have different types of content. The different content types are:

- Empty
  This means that there is no information contained in the element.
- Element content
  This means that the element contains other elements (sub elements)
- Simple content
  This means that the element contains only text.
- Mixed content
  This means that the element contains text and other elements.

And finally, a note about element naming conventions. There are no reserved names in XML. Any valid name can be used. Names should be descriptive for legibility purposes, but this is not a requirement. If XML documents are used with a database, it is good practice to have database fields correspond to element names. Some additional rules for element naming are:

- Names can contain letters, numbers and other characters
- Names must not start with a number or punctuation character
- Names must not start with an xml (regardless of case)
- Names must not contain any spaces

IMPORTANT NOTES ABOUT XML ELEMENTS

- **All elements must have a name.**
- **All XML documents must have a root element that encompasses the entire document.**
- **There must be an end (closing tag) that corresponds to the root element**
- **Element names are case sensitive and must start with a letter or underscore. They can contain letters, digits, hyphens, underscores and periods.**
- **Element names must not start with XML (regardless of the case) and must not contain any white space**
- **Do not use colons as part of an element name. This can create confusion in the parser with namespaces.**
- **Not recommended to use a period "." (used with object properties) or a minus "-" (used for subtraction)**
- **Line feed (vbNewLine in VBScript) is used to move to a new line of XML data**

Character Data

Character data, or text data, is what is generally contained between a sibling's (or leaf's) start and end tags. Character data may be any legal character (8-bit or 16-bit Unicode) with the exception of the right arrow character "<" which is reserved for the start of a tag. Also note in the character references in section below.

Character and Entity References

Character and entity references provide a means to include characters or other information into an XML document by reference rather than typing the characters into the document directly. XML includes five built-in entities for characters to be used in XML data or markup so as to avoid any confusion as to whether character data or markup is being specified. These include:

Character	Entity Reference	Meaning
>	>	Less than
<	<	Greater than
&	&	Ampersand
"	"	Apostrophe or single quote
'	'	Double quote

For example, to write "a<b" you would use "a<b"

In addition, you can use the syntactical construct of an ampersand (&) followed by a value followed and ending a semicolon (;). For example, & #value; is the syntax used for decimal references and &# xvalue; is the syntax used for hexadecimal references. So if you wanted to include the Euro symbol, this could be done by inserting € or € into a document.

XML also provides for document fragments to be incorporated by reference. To use these type of entities, they must be declared in an XML document using a DOCTYPE declaration that always follows the prolog. In the XML document, the document fragment entity is referenced by the following syntax: &entityName;

Example2

```
<?xml version=”1.0”>
<!DOCTYPE AlarmMessages [
        <!ENTITY head1 “Production Shift”>
        <!ENTITY head2 “Total Production”>
        <!ENTITY head3 “Total Defects”>
        ]>
<prodReport>
    <item>
        <heading>&head1;</heading>
        <value>Shift 1<value>
    </item>
        <item>
        <heading>&head2;</heading>
        <value>5000<value>
    </item>
    <item>
        <heading>&head3;</heading>
        <value>4<value>
    </item>
</prodReport>
```

There are other entity references within XML, but they are beyond the scope of these materials.

IMPORTANT NOTES ABOUT XML CHARACTER AND ENTITY REFERENCES

- **Use when characters cannot be entered directly into a document because they would be interpreted as markup, input device limitations or when 8-bit characters are used**
- **Can use when entity references when character strings or document fragments appear repeatedly and can be abbreviated, or when it improves legibility.**
- **In the entity reference, there cannot be any white spaces (between the & and the ;)**

CDATASection

As we have seen so far, generally anything that is inside of a tag is considered markup where anything outside of tags is considered to be character data. CDATA section blocks are a way of telling the parser that there is no markup in the characters contained in the CDATA section. Generally, CDATA sections are used for scripting language content. See the MSDN site for more detail.

IWS XML Recipe File Syntax

Now that we have looked at the XML language syntax, let's look at an IWS recipe file in XML. For this example, there was a IWS class tag ("Cake") generated which has five elements (flour, eggs, milk, sugar, and chocolate) as well as a baking time IWS tag named "Time". These IWS tags were defined in the IWS Recipe Manager and the Recipe Manager was set up to save the recipe data into a file (MyCake.xml) using XML Unicode format. A XLS style sheet was automatically created by IWS so the XML data could be viewed in Microsoft Internet Explorer.

Example 3

```
1   <?xml version="1.0"?>
2   <?xml-stylesheet type="text/xsl" href="recipe.xsl"?>
3   <TAGLIST>
4     <TAG>
5       <NAME>Cake.flour</NAME>
6       <VALUE>40</VALUE>
7     </TAG>
8     <TAG>
9       <NAME>Cake.eggs</NAME>
10      <VALUE>20</VALUE>
11    </TAG>
12    <TAG>
13      <NAME>Cake.milk</NAME>
14      <VALUE>10</VALUE>
15    </TAG>
16    <TAG>
17      <NAME>Cake.sugar</NAME>
18      <VALUE>10</VALUE>
19    </TAG>
20    <TAG>
21      <NAME>Cake.chocolate</NAME>
22      <VALUE>8</VALUE>
23    </TAG>
24     <TAG>
25      <NAME>Time</NAME>
26      <VALUE>35</VALUE>
27     </TAG>
28  </TAGLIST>
29
```

Note: The line numbers were included for reference only and were not generated by IWS. This example illustrates many of XML's syntax rules:

- Line 1 is an XML declaration
- Line 2 is an XML processing instruction (style sheet instruction to Microsoft Internet Explorer)
- Line 3 is the root element and line 28 is the end (closing) tag of the root element
- Each element "TAG" has two child elements, "NAME" and "VALUE". NAME is the name of the IWS tag (uses either the IWS tag name or the IWS class.member name format)

Now that you are an expert on reading XML files, have a look at the corresponding .xsl style sheet file. This file was named Recipe.xsl and was created by the IWS Recipe Manager. It's sole purpose is to define how to display (format) the XML data from the XML Recipe file in a browser such as Internet Explorer.

```
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
- <xsl:template match="/">
- <HTML>
- <BODY>
- <table border="4" width="100%">
- <tr>
- <td width="40%">
  <b>Tagname</b>
    </td>
- <td width="60%">
  <b>Value</b>
    </td>
    </tr>
    </table>
- <table border="1" width="100%">
- <tr>
  <xsl:apply-templates />
    </tr>
    </table>
    </BODY>
    </HTML>
    </xsl:template>
- <xsl:template match="TAGLIST">
- <xsl:for-each select="TAG">
- <tr>
  <xsl:apply-templates />
    </tr>
    </xsl:for-each>
    </xsl:template>
- <xsl:template match="NAME">
- <td width="40%">
  <xsl:apply-templates />
    </td>
    </xsl:template>
- <xsl:template match="VALUE">
- <td width="60%">
  <xsl:apply-templates />
    </td>
    </xsl:template>
- <xsl:template match="text()">
  <xsl:value-of />
    </xsl:template>
    </xsl:stylesheet>
```

Using the XML data from the Recipe file MyCake.xml with the XSL style sheet created by the IWS Recipe Manager, the following would be displayed if the XML Recipe file was accessed by Internet Explorer.

Tagname	Value
Cake.flour	40
Cake.eggs	20
Cake.milk	10
Cake.sugar	10
Cake.chocolate	8
Time	35

As discussed earlier, the XSL style sheet is only used to format the XML data so it is "legible" by the user when viewed in a browser such and Internet Explorer. If a style sheet was not created, the data would appear as shown in the MyCake.xml file when opened by Internet Explorer.

IMPORTANT NOTES

- **So far, we have looked at raw XML files (and one XLS file). We have done this so to become familiar with the various components of an XML file.**
- **While you can create text files for XML, it is much easier to use either the IWS Recipe Manager for XML-based recipe files for IWS, or to use VBScript with the MSXML (XMLDOM COM object) to create custom XML files. Alternatively, you can use VBScript with ADO.NET (ADODB COM object) to communicate with databases, or when combined with FileScriptingObject, to save XML recordsets to a file.**

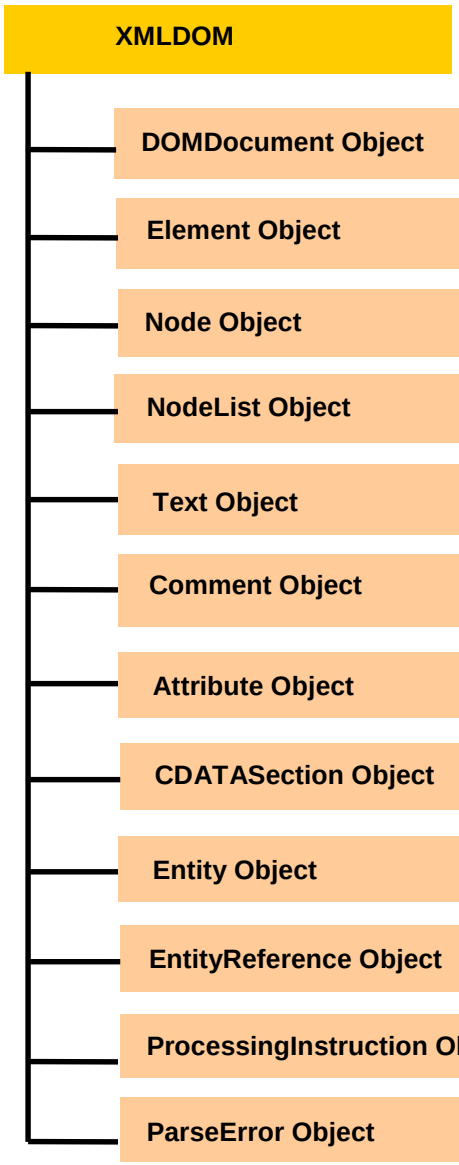
Using XMLDOM in VBScript

If you are just loading and saving XML Recipe files using the built-in IWS Recipe Manager task, you do not need to use VBScript. However, if you need to transfer other data contained in IWS to an external program using the XML format, then using the MSXML XMLDOM in VBScript is an option.

XMLDOM Objects

The following diagram illustrates the XMLDOM object model. In this section, we will itemize the various properties and methods available to the objects. But since many are common among the objects, the properties and methods will be covered later.

XMLDOM Objects



XMLDOM		Microsoft's MSXML DOM Parser that is available to VBScript as a COM object.
DOMDocument Object	- Properties - Methods	Represents the entire XML document.
Element Object	- Properties - Methods	An element in an XML document. If the element contains text, the text is represented in a text-node.
Node Object	- Properties - Methods	A node in the node-tree.
NodeList Object	- Properties - Methods	Represents a node and its child-nodes as a node tree
Text Object	- Properties - Methods	The text of an element or an attribute
Comment Object	- Properties - Methods	A comment element in an XML document
Attribute Object	- Properties - Methods	Represents an attribute in an element in an XML document
CDATASection Object	- Properties - Methods	Represents a CDATA section in an XML document
Entity Object	- Properties - Methods	Represents an Entity in the XML document
EntityReference Object	- Properties - Methods	Represents an Entity Reference node
ProcessingInstruction Object	- Properties - Methods	Represents a Processing Instruction
ParseError Object	- Properties	Used to retrieve error information from the XMLDOM parser

XMLDOM Document Object

Purpose — Represents the top level of the XML source. Includes members for retrieving and creating all other XML objects

Remarks — Only one Document object can be created. All other objects are accessed or created from the document. When the object-creation methods, such as createElement, are used on the document, nodes are created in the context of the document, but the node is not part of the document tree. The node is only part of the document tree when it is explicitly added to the tree by calling insertBefore, replaceChild, or appendChild; or for attributes, using setAttributeNode.

Methods and Properties for the Document object[4]

Method	Properties
appendChild	async
createAttribute	attributes
createCDATASection	baseName
createComment	childNodes
createDocumentFragment	dataType
createElement	documentElement
createEntityReference	firstChild
createNode	lastChild
createProcessingInstruction	nextSibling
createTextNode	nodeName
getElementsByTagName	nodeType
hasChildNodes	nodeTypedValue
insertBefore	nodeTypeString
load	nodeValue
loadXML	parentNode
nodeFromID	parsed
removeChild	parseError
replaceChild	prefix
save	preserveWhiteSpace
selectNodes	previousSibling
selectSingleNode	readyState
	text
	url
	validateOnParse
	xml

To create the Document object, you must instantiate it in VBScript through any of the statements below. Once the object is instantiated, you can access the various XMLDOM objects' properties and methods.

There are both rental-treaded and free-threaded versions of the XMLDOM control. The XML document can be created by either version. The rental version is created for single-threaded access while the free-threaded version is created for multi-threaded access. If you only have one thread to access the XML data through the XMLDOM control, then use the rental version since it will run somewhat faster (the XMLDOM control does not have to keep track of multiple threads). If you have several threads potentially accessing the XML data through the XMLDOM control, then use the free-threaded version.

```
Rental Threaded
Set mydom = CreateObject("Microsoft.XMLDOM")            'Uses the latest version of the XMLDOM Parser
Set mydom = CreateObject("MSXML2.DOMDocument")          'Uses the latest version of the XMLDOM Parser
Set mydom = CreateObject("MSXML2.DOMDocument.4.0")      'Uses the MSXML 4.0 Parser SP2 (XMLDOM)

Free Threaded
Set mydom = CreateObject("MSXML2.FreeThreadedDOMDocument")      'Free-threaded version
```

[4] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

```
Set mydom = CreateObject("MSXML2.FreeThreadedDOMDocument.3.0")   'Use free-threaded Version 3
Set mydom = CreateObject("MSXML2.FreeThreadedDOMDocument.4.0")   'Use free-threaded Version 4
```

XMLDOM Element Object

Purpose — Represents the top level of the XML source. Includes members for retrieving and creating all other XML objects

Remarks — Element nodes are among the most common objects in the XML document tree. Element nodes can have attributes associated with them. By definition, attributes are not defined as child nodes of an element and are not considered to be part of the document tree. Accordingly, the Element object provides methods to make it easier to manage attributes, including methods to associate an attribute with an element and to retrieve an attribute object and the attribute value by name.

To retrieve the set of all attributes associated with an element, you can also access the attributes property, which returns a NamedNodeMap collection object that contains all the element's attributes.[5]

Methods and Properties for the Element object[6]

Method	Properties
appendChild	attributes
getAttribute	baseName
getAttributeNode	childNodes
getElementsByTagName	dataType
hasChildNodes	firstChild
insertBefore	lastChild
removeAttribute	nextSibling
removeAttributeNode	nodeName
removeChild	nodeType
replaceChild	nodeTypeString
selectNodes	nodeValue
selectSingleNode	parentNode
setAttribute	parsed
setAttributeNode	prefix
	previousSibling
	tagName
	text
	xml

XMLDOM Node Object

Purpose — Extends the core node with support for data types, namespaces, document type definitions (DTDs) and schemas.

Remarks

Methods and Properties for the Node object[7]

[5] Reference the MSDN website for further information.
[6] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.
[7] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

Method	Properties
appendChild	attributes
hasChildNodes	baseName
insertBefore	childNodes
removeChild	dataType
replaceChild	firstChild
selectNodes	lastChild
selectSingleNode	nextSibling
	nodeName
	nodeType
	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	text
	xml

XMLDOM NodeList Object

Purpose Supports iteration through a collection of nodes, in addition to indexed access.

Remarks A NodeList collection is live; that is, the addition and removal of nodes, and changes within nodes, are immediately reflected in the collection. This means that two successive requests for items using the same index can return two different items, depending on changes to the collection. This also means that changes to the node objects are immediately available in the nodes obtained from the list. The collection can also be accessed using the "for...next" construct.

Methods and Properties for the NodeList object[8]

Method	Properties
item	length
nextNode	
reset	

XMLDOM Text Object

Purpose Represents the text content of an element or attribute

Remarks XML refers to text content as character data and distinguishes it from markup, the tags that modify the character data. If there is no markup inside an element, that element's text is contained in a single Text object that is the child of the element. If there is markup inside an element, it is parsed into child

[8] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

elements that are siblings of the Text object(s). The content of the markup elements also appears within text nodes, which are the children of the specific markup element.

When a document is first made available to the XML Document Object Model (DOM), all text is normalized: there is only one text node for each block of text. Users can create adjacent text nodes that represent the contents of a given element without any intervening markup but should be aware that there is no way to represent the separations between these nodes, so they will not persist between XML DOM sessions.

Methods and Properties for the Text object[9]

Method	Properties
appendChild	attributes
appendData	baseName
deleteData	childNodes
hasChildNodes	data
insertBefore	dataType
insertData	firstChild
removeChild	lastChild
replaceChild	length
selectNodes	nextSibling
selectSingleNode	nodeName
specified	nodeType
splitText	nodeTypedValue
substringData	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	text
	xml

XMLDOM Comment Object

Purpose Represents the content of an XML comment

Remarks The content refers to all characters between the start <!-- and end --> tags..

[9] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

Methods and Properties for the Comment object[10]

Method	Properties
appendChild	attributes
appendData	baseName
deleteData	childNodes
hasChildNodes	data
insertBefore	dataType
insertData	firstChild
removeChild	lastChild
replaceChild	length
replaceData	nextSibling
selectNodes	nodeName
selectSingleNode	nodeType
substringData	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	text
	xml

[10] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM Attribute Object

Purpose — Represents an attribute of an Element. Valid and default values for the attribute are defined in the document type definition (DTD) or schema.

Remarks — Attribute nodes cannot be the immediate children of DocumentFragment. However, they can be associated with element nodes that are children of DocumentFragment node.

The relationship between the value and specified members can be summarized as follows: If the attribute has an assigned value in the document and the specified member is True, the value is the assigned value. If the attribute has no assigned value in the document and has a default value in the DTD, the specified member is False and the value is the default value in the DTD. If the attribute has no assigned value in the document and has a value of #IMPLIED in the DTD, the attribute does not appear in the structure model of the document.

In XML, the value of an attribute is represented by the child nodes of the attribute node because the value can contain entity references. Thus attributes that contain entity references will have a child list containing both text nodes and entity reference nodes. In addition, because the attribute type might be unknown, there are no tokenized attribute values.

Methods and Properties for the Attribute object[11]

Method	Properties
appendChild	attributes
hasChildNodes	baseName
insertBefore	childNodes
removeChild	dataType
replaceChild	firstChild
selectNodes	lastChild
selectSingleNode	name
	nextSibling
	nodeName
	nodeType
	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	text
	value
	xml

[11] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM CDATASection Object

Purpose Used to quote or escape blocks of text to keep that text from being interpreted as markup language.

Remarks The CDATA section lets you include material such as XML fragments within XML documents without needing to escape all the delimiters. The only delimiter recognized in a CDATA section is the "]]>" string that ends the CDATASection.

CDATA sections cannot be nested.

The text contained by the CDATA section is stored in a text node. This text can contain characters that would need to be escaped outside of CDATA sections.

The CDATASection has no unique members of its own, but exposes the same members as the Text object.

Although the CDATASection inherits Text, unlike text nodes, the normalize method of Element does not merge CDATASection nodes.

Methods and Properties for the CDATASection object[12]

Method	Properties
appendChild	attributes
appendData	baseName
deleteData	childNodes
hasChildNodes	data
insertBefore	dataType
insertData	firstChild
removeChild	lastChild
replaceChild	length
replaceData	nextSibling
selectNodes	nodeName
selectSingleNode	nodeType
splitText	nodeTypedValue
substringData	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	text
	xml

[12] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM Entity Object

Purpose Represents a parsed or unparsed Entity in the XML document.

Remarks Entity represents the entity itself rather than the entity declaration. The World Wide Web Consortium (W3C) Document Object Model (DOM) does not currently define an object that models the entity declaration.

According to the W3C DOM specification, an XML parser can completely expand entity references into entities before the structure model is passed to the DOM. When these entity references are expanded, the document tree does not contain any entity references.

When this parser validates the XML document, it expands external entities, except for binary entities. The nodes representing the expanded entity are available as read-only children of the entity reference. This implementation does not expand these entities when it is not validating.

The nodeName property contains the name of the entity.

The structure of the entity child list is exactly the same as the structure of the child list for the EntityReference object with the same nodeName value.

Level 1 of the W3C DOM application programming interface (API) does not define a way to change entity nodes. All Entity object properties are read-only. This interface inherits all of the methods, properties, and events of Node.

Methods and Properties for the Entity object[13]

Method	Properties
appendChild	attributes
hasChildNodes	baseName
insertBefore	childNodes
removeChild	dataType
replaceChild	firstChild
selectNodes	lastChild
selectSingleNode	nextSibling
	nodeName
	nodeType
	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	text
	xml

[13] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM EntityReference Object

Purpose Represents an Entity Reference node

Remarks If the XML parser expands entity references while building the structure model, no EntityReference objects appear in the tree.

XML does not require non-validating processors to handle external entity declarations, such as those made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded, and therefore the replacement value of the entity might not be available. If available, the replacement value of the referenced entity appears in the child list of EntityReference.

The resolution of the children of the EntityReference, the replacement value of the referenced entity, can be evaluated. Actions such as calling the childNodes property are assumed to trigger the evaluation. Character entities are expanded by the XML parser and do not appear as entity references, but instead appear within the character text in Unicode.

Methods and Properties for the EntityReference object[14]

Method	Properties
appendChild	attributes
hasChildNodes	baseName
insertBefore	childNodes
removeChild	dataType
replaceChild	firstChild
selectNodes	lastChild
selectSingleNode	nextSibling
	nodeName
	nodeType
	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	text
	xml

[14] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM ProcessingInstruction Object

Purpose — Represents a processing instruction, defined by XML to keep processor-specific information in the text of the document

Remarks — The content of the ProcessingInstruction node is the entire content between the delimiters of the processing instruction, <? and ?>.

The content of this node is usually subdivided into the target, which is the application that this processing instruction is directed toward, and the content of the processing instruction. The target consists of the first token following the start of the tag, signified by <?, while the content of the processing instruction refers to the text that extends from the first non-white space character after the target through the character immediately preceding the ?>, which signifies the end of the tag.

This interface inherits all of the methods, properties, and events of Node.

Methods and Properties for the ProcessingInstruction object[15]

Method	Properties
appendChild	attributes
hasChildNodes	baseName
insertBefore	childNodes
removeChild	data
replaceChild	dataType
selectNodes	firstChild
selectSingleNode	lastChild
	nextSibling
	nodeName
	nodeType
	nodeTypedValue
	nodeTypeString
	nodeValue
	parentNode
	parsed
	prefix
	previousSibling
	specified
	target
	text
	xml

[15] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

XMLDOM ParseError Object

Purpose Returns detailed information about the last parse error, including the error number, line number, character position, and a text description.

Methods and Properties for the ProcessingInstruction object[16]

Method	Properties
	errorCode
	filepos
	line
	linepos
	reason
	srcText
	url

[16] Note: this is not a complete list of methods and properties. See MDSN website for a complete listing.

Key XMLDOM Methods & Properties

Methods	XP	CE	Description
appendChild	✓	✓	Appends a new child node as the last child of the node
appendData	✓		Appends a supplied string to the existing string data
createAttribute	✓	✓	Creates a new attribute with the specified name
createCDATASection			
createComment	✓	✓	Creates a comment node that contains the supplied data
createDocumentFragment	✓	✓	Creates and empty document fragment
createElement	✓	✓	Creates an element node using the specified name
createEntityReference	✓	✓	Creates a new Entity Reference object
createNode*	✓	✓	Creates a node using the supplied type, name, and namespace.
createProcessingInstruction	✓	✓	Creates a processing instruction node (declaration & processing)
createTextNode	✓	✓	Creates a text node containing the supplied data
deleteData	✓	✓	Deletes the specified substring within the string of data
getAttribute	✓		Gets the value of the attribute
getAttributeNode	✓		Gets the attribute node
getElementsByTagName	✓	✓	Returns a collection of elements that have the specified name
getNamedItem	✓		Retrieves the attribute with the specified name
hasChildNodes	✓	✓	Verifies whether a node has children
insertBefore	✓		Inserts a child node to the left of the specified node or at the end of the list.
insertData	✓	✓	Inserts a string at the specified offset
item	✓	✓	Allows random access to individual nodes in a nodeList collection
load*	✓	✓	Loads an XML document from the specified location (file or URL)
loadXML*	✓	✓	Loads an XML document using the supplied string
nextNode*	✓	✓	Returns the next node from a nodeList collection.
nodeFromID*	✓		Returns a node whose identifier attribute matches the supplied value
removeAttribute	✓		Removes or replaces the named attribute
removeAttributeNode	✓		Removes the specified attribute from this element
removeChild	✓		Removes the specified child node from the list of children and returns it.
replaceChild	✓		Replaces the specified old child node with the supplied new child node in the set of children in this node, and returns the old child node
replaceData	✓	✓	Replaces the specified number of characters with the supplied string
reset*	✓	✓	Resets the iterator in a nodeList collection
save*	✓	✓	Saves an XML document to the specified location
selectNodes*	✓	✓	Applies pattern matching and returns a list of matching nodes
selectSingleNode*	✓	✓	Applies pattern matching and returns the first matching node
setAttribute	✓		Sets the value of a named attribute
setAttributeNode	✓		Sets or updates the supplied attribute node on this element
setNamedItem	✓		Adds the supplied node to the collection
setOption	✓		Sets the specified option
splitText	✓	✓	Splits the text node into two text nodes at the specified offset and inserts a new text node into the tree as a sibling following the node
substringData	✓	✓	Retrieves a substring of the full string from the specified range
validate	✓		Performs run-time validation
validateNode	✓		Validates a specified DOM fragment

* denotes an extension to the W3C DOM recommendation

While it is beyond the scope of this document to cover all of the key XMLDOM methods in detail, we will cover a few specific methods commonly used.

appendChild

Purpose	Appends newChild as the last child of a node
Use	objNode = objXMLDOM.appendChild(*newChild*)
Parameters	*newChild* An object that is contains the new child to be appended a the end of the list of children belonging to this node.
Return	Returns an object that is the new child node successfully appended to the list
Example	Set doc = CreateObject("Microsoft.XMLDOM") Set root = doc.createElement("TAGLIST") doc.appendChild root

createComment

Purpose	Creates a comment node that contains the supplied data
Use	objElement = objXMLDOM.createComment(*tagName*)
Parameters	*data* A string specifying the value to be supplied in the new Comment object's *nodeValue* property.
Return	Returns an object that is the new comment.
Example	Set doc = CreateObject("Microsoft.XMLDOM") Set node = doc.createComment("This is a sample IWS XML Recipe file created by VBScript")

createElement

Purpose	Creates an element node using the specified name
Use	objElement = objXMLDOM.createElement(*tagName*)
Parameters	*tagName* A string specifying the name for the new element node. The name is case-sensitive. The name is subsequently available as the element node's nodeName property.
Return	Returns an object that is the object for the new element.
Example	Set doc = CreateObject("Microsoft.XMLDOM") Set root = doc.createElement("TAGLIST") doc.appendChild root Set node = dom.createElement("TAG")

createTextNode

Purpose	Creates an text node that contains the supplied data
Use	objElement = objXMLDOM.createTextNode(*data*)
Parameters	*data* A string specifying the value to be supplied to the new text object's *nodeValue* property.
Return	Returns an object that is the text object.
Example	Set doc = CreateObject("Microsoft.XMLDOM") Set root = doc.createElement("TAGLIST") root.appendChild doc.createTextNode(vbNewLine & vbTab)

createProcessingInstruction

Purpose	Creates an XML declaration or processing instruction object that can be inserted into an XML document
Use	objProcInstr = objXMLDOM.createProcessingInstruction(*target, data*)
Parameters	*target* A string specifying the target part of the processing instruction. This supplies the *nodename* property of the new object. *Data* A string specifying the rest of the processing instruction proceeding the closing ?> characters. This supplies the nodevalue property of the new object.
Return	Returns an object
Notes	This instruction is used for XML declaration and Processing Instructions
Example	Set doc = CreateObject("Microsoft.XMLDOM") Set node = doc.createProcessingInstruction("xml", "version=1.0") doc.appendChild node

load

Purpose	Load an XML document from the specified location
Use	boolValue = objXMLDOM.load(*xmlSource*)
Parameters	*xmlSource* A string containing a file name or URL that specifies the location of the XML file
Return	Boolean TRUE if the load was successful, FALSE if the load failed
Example	Set doc = CreateObject("Microsoft.XMLDOM") doc.load ("MyCake.xml")

loadXML

Purpose	Loads an XML document using the supplied string
Use	boolValue = objXMLDOM.loadXML(*strXML*)
Parameters	*strXML* A string containing the XML string to load into this XML document object. The string can contain an entire XML document or a well-formed fragment
Return	Boolean TRUE if the load was successful, FALSE if the load failed. If the load failed, the *documentElement* property of the *DOMDocument* is set to Null.
Example	Dim doc, text text = "<note>" text = text & "<to>Jack</to>" text = text & "<from>Jill</from>" text = text & "<heading>Reminder</heading>" text = text & "<body>Be careful when carrying the pail of water down the hill. Love, Jill.</body>" text = text & "</note>" Set Mydoc = CreateObject("Microsoft.XMLDOM") doc.loadXML (text)

save

Purpose	Saves an XML document to the specified location
Use	objXMLDOM.save(*destination*)
Parameters	*destination* An object. The object can be a file name, a DOMDocument object or a customer object that supports persistence.
Return	None
Example	Dim doc, path Set doc = CreateObject("Microsoft.XMLDOM") ' doc.load ("MyCake.xml") //……. Commands to process the document go here path = $GetAppPath() & "MyCake.xml" doc.save path

Properties	XP	CE	Description
async*	✓	✓	Specifies whether asynchronous download (file access) is permitted (read/write)
attributes	✓	✓	Returns a list of attributes for this node (read only)
baseName*	✓	✓	Returns the base name for the name qualified with the namespace (read only)
childNodes	✓	✓	Contains a node list containing the child nodes (read only)
data	✓	✓	Contains this node's data, dependant on node type (read/write)
dataType*	✓	✓	Specifies the data type for this node (read/write)
documentElement	✓	✓	Contains the root element of the document (read/write)
errorCode	✓	✓	Contains the error code of the last parse error (read only)
filepos	✓	✓	Contains the absolute file position where the error occurred (read only)
firstChild	✓	✓	Contains the first child of a node (read only)
item	✓		Returns the item from the collection with the specified index (read only)
lastChild	✓	✓	Returns the last child node (read only)
length	✓	✓	Indicates the number of items in a collection (read only)
line	✓	✓	Specifies the line number that contains the error (read only)
linepos	✓	✓	Specifies the character position within the line where the error occurred (read only)
name	✓	✓	The attribute name (read only)
next			Returns the next item in the error collection object (read only)
nextSibling	✓	✓	Contains the next sibling of the node in the parent's child list (read only)
nodeName	✓	✓	Contains the qualified name of the element, attribute or entity reference, or a fixed string for other node types (read only)
nodeType	✓	✓	Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes (read only)
nodeTypedValue*	✓	✓	Returns the node's value expressed in its defined data type. (read/write)
nodeTypeString*	✓	✓	Returns the node type in string form (read only)
nodeValue	✓	✓	Contains the text associated with the node (read/write)
parentNode	✓	✓	Contains the parent node (read only)
parsed*	✓	✓	Contains TRUE is this node and all descendants have been parse and instantiated. Contains FALSE if any nodes remain to be parsed. (read only)
parseError*	✓	✓	Returns the ParseError object that contains information about the last parsing error. (read only)
prefix*	✓	✓	Returns the namespace prefix (read only)
preserveWhiteSpace*	✓	✓	Specifies the default white space handling (read/write)
previousSibling	✓	✓	Contains the left sibling of this node (read only)
readyState*	✓	✓	Indicates the current state of the XML document (read only)
reason	✓	✓	Explains the reason for the error
specified*	✓	✓	Indicates whether the node is explicitly defined or derived
srcText	✓	✓	Returns the full length text of the line containing the error (read only)
tagName	✓	✓	Contains the element name (read only)
target	✓	✓	Specifies the target, the application to which a processing instruction is directed. (read only)
text*	✓	✓	Represents the text content of the node or concatenated text representing the node and its descendants (read only)
url*	✓	✓	Returns the canonicalized URL for the last loaded XML document (read only)
validateOnParse*	✓	✓	Indicates whether the parser should validate this document (read/write)
xml*	✓	✓	Contains XML representation of the node and all its descendants (read only)
Value	✓	✓	Contains the attribute value (read/write)

* denotes an extension to the W3C DOM recommendation

While it is beyond the scope of this document to cover all of the key XMLDOM properties in detail, we will cover a few specific properties commonly used.

async

Purpose	Specifies if asynchronous download is permitted.
Use	*boolValue* = objXMLDOM.async(*destination*) objXMLDOM.async(*destination*) = *boolValue*
Parameters	*boolValue* [out][in] TRUE if asynchronous download (or access) is permitted. FALSE if not. Default is TRUE.
Return	
Note	When set to TRUE, the load method returns control to the caller before the download (access) is finished. The *readyState* property can be used to check the status of the download.
Example	Dim doc Set doc = CreateObject("Microsoft.XMLDOM") ‘ doc.async = FALSE

childNodes

Purpose	Contains a node list containing the children nodes.
Use	objNode.childNodes
Parameters	None
Return	A read-only collection (list) of children in the current node.
Note	Even if there are no children, an empty list will be returned. However, its length will be 0.
Example	Dim doc, root, objNodeList, i, item Set doc = CreateObject("Microsoft.XMLDOM") ‘ doc.load ("myfile.xml") root = doc.documentElement objNodeList = root.childNodes For i =0 to objNodeList.length -1 Item = objNodeList.item(i) Next

item

Purpose	Returns the item from the collection with the specified index. The index starts at 0.
Use	obj = objCollection.item(*index*)
Parameters	*index* A zero-based index (integer) to the collection.
Return	An object, based on the collection type. For example, if the collection is a node list (child nodes), the returned item will be a child node object.
Note	Not included in XMLDOM for Windows CE at this time.
Example	Dim doc, root, objNodeList, i, item Set doc = CreateObject("Microsoft.XMLDOM") ‘ doc.load ("myfile.xml") root = doc.documentElement objNodeList = root.childNodes For i =0 to objNodeList.length -1 Item = objNodeList.item(i) Next

length

Purpose	Indicates the number of items in a collection
Use	intValue = objNodeList.length
Parameters	None
Return	An integer value indicating the number of items in a collection
Note	Can be used with a NodeList collection (eg. childNodes) or ParseErrorCollection
Example	Dim doc, root, objNodeList, i, item Set doc = CreateObject("Microsoft.XMLDOM") ‘ doc.load ("myfile.xml") root = doc.documentElement objNodeList = root.childNodes For i =0 to objNodeList.length -1 Item = objNodeList.item(i) Next

text

Purpose: Represents the text content of the node or concatenated text representing the node and its descendants

Use:
strValue = objNode.text
objNode.text = strValue

Parameters: None

Return: Read/write string variable representing the text content

Note: Retrieves and sets the string representing the text contents of this node or the concatenated text representing this node and its descendants. When concatenated, the text represents the contents of text or CDATA nodes. This value depends on the value of the nodeType property.

Value	Description
NODE_ATTRIBUTE, NODE_DOCUMENT, NODE_ENTITY	Returns a string representing the value of the node. This is the concatenated text of all subnodes with entities expanded.
NODE_CDATA_SECTION, NODE_COMMENT, NODE_PROCESSING_INSTRUCTION, NODE_TEXT	Returns the text contained in the node, which is the same as the nodeValue property.
NODE_DOCUMENT_TYPE, NODE_NOTATION	Returns the empty string (""). These node types do not have associated text.
NODE_DOCUMENT_FRAGMENT	Returns the text comprised of the concatenation of all descendant nodes.
NODE_ELEMENT	Contains a string that represents the element content. Note that this will also include the text content from all child elements, concatenated in document order. For example, consider the following XML: <count> <item>one</item> <item>two</item> <item>three</item> <item>four</item> </count> The **text** property for the <count> element contains the value "one two three four".
NODE_ENTITY_REFERENCE	Returns the string representation of the entity reference.

Example:

```
Dim doc
Set doc = CreateObject(“Microsoft.XMLDOM”)    ‘
doc.load (“Myfile.xml”)
set curNode = doc.documentElement.childNodes.item(0)
MsgBox curNode.text
```

xml

Purpose: Contains the XML representation of the node and all its descendants

Use:
strValue = objNode.xml
objNode.xml = strValue

Parameters: None

Return: Read/write string variable representing the xml content

Note: Retrieves and sets the string representing the XML text contents of this node or the concatenated text representing this node and its descendants. When concatenated, the text represents the contents of text or CDATA nodes. This value depends on the value of the nodeType property.

Value	Description
NODE_ATTRIBUTE, NODE_DOCUMENT, NODE_ENTITY	Returns a string representing the value of the node. This is the concatenated text of all subnodes with entities expanded.
NODE_CDATA_SECTION, NODE_COMMENT, NODE_PROCESSING_INSTRUCTION, NODE_TEXT	Returns the text contained in the node, which is the same as the nodeValue property.
NODE_DOCUMENT_TYPE, NODE_NOTATION	Returns the empty string (""). These node types do not have associated text.
NODE_DOCUMENT_FRAGMENT	Returns the text comprised of the concatenation of all descendant nodes.
NODE_ELEMENT	Contains a string that represents the element content. Note that this will also include the text content from all child elements, concatenated in document order. For example, consider the following XML: <count> <item>one</item> <item>two</item> <item>three</item> <item>four</item> </count> The **text** property for the <count> element contains the value "one two three four".
NODE_ENTITY_REFERENCE	Returns the string representation of the entity reference.

Example:

```
Dim doc
Set doc = CreateObject(“Microsoft.XMLDOM”)    ‘
doc.load (“Myfile.xml”)
set curNode = doc.documentElement.childNodes.item(0)
MsgBox curNode.xml
```

Key XMLDOM enumerated Constants

Enumerated Constant	Value	Description
Node_Element	1	The node represents an element. An Element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection, and EntityReference. The Element node can be the child of the Document, DocumentFragment, EntityReference, and Element nodes.
Node_Attribute	2	The node represents an attribute of an element. An Attribute node can have the following child node types: Text and EntityReference. The Attribute node does not appear as the child node of any other node type; it is not considered a child node of an Element.
Node_Text	3	The node represents the text content of a tag. A Text node cannot have any child nodes. The Text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.
Node_CDATA_Section	4	The node represents a CDATA section in the XML source. CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATASection node cannot have any child nodes. The CDATASection node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.
Node_Entity_Reference	5	The node represents a reference to an entity in the XML document. This applies to all entities, including character entity references. An EntityReference node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The EntityReference node can appear as the child of the Attribute, DocumentFragment, Element, and EntityReference nodes.
Node_Entity	6	The node represents an expanded entity. An Entity node can have child nodes that represent the expanded entity (for example, Text and EntityReference nodes). The Entity node can appear as the child of the DocumentType node.
Node_Processing_Instruction	7	The node represents a processing instruction from the XML document. A ProcessingInstruction node cannot have any child nodes. The ProcessingInstruction node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
Node_Comment	8	The node represents a comment in the XML document. A Comment node cannot have any child nodes. The Comment node can appear as the child of the Document, DocumentFragment, Element, and EntityReference nodes.
Node_Document	9	The node represents a document object, that as the root of the document tree, provides access to the entire XML document. It is created using the progID "Microsoft.XMLDOM". A Document node can have the following child node types: Element (maximum of one), ProcessingInstruction, Comment, and DocumentType. The Document node cannot appear as the child of any node types.
Node_Document_Type	10	The node represents the document type declaration, indicated by the <!DOCTYPE> tag. A DocumentType node can have the following child node types: Notation and Entity. The DocumentType node can appear as the child of the Document node.
Node_Document_Fragment	11	The node represents a document fragment. The DocumentFragment node associates a node or subtree with a document without actually being contained within the document. A DocumentFragment node can have the following child node types: Element, ProcessingInstruction, Comment, Text, CDATASection, and EntityReference. The DocumentFragment node cannot appear as the child of any node types.
Node_Notation	12	The node represents a notation in the document type declaration. A Notation node cannot have any child nodes. The Notation node can appear as the child of the DocumentType node.

Applying XMLDOM in VBScript

Now that we have covered the XML language and the XMLDOM COM object in some detail, let's look at how to use this information to build IWS applications.

Example 4 - Generating an IWS XML Recipe File with VBScript

This example shows how to use VBScript with XMLDOM to create an IWS XML Recipe File, albeit the hard way since IWS already contains a Recipe Manager which does the same thing. But with a simple modification to this program, you can extract any information contained in a PC running IWS and send it to an XML file to be used by another application.

This application uses 3 variables (var1, var2, var3) which are IWS integer tags, which were set by the application to be var1 = 100, var2 = 200, var3 = 300. The VBScript code was placed in a screen script. When the code in the screen script is executed, an XML Recipe File is generated. Let's look at the resulting XML recipe file created (recipe1.xml), and then at the VBScript code used to generate the XML recipe file.

XML recipe file generated

```
<?xml version="1.0" ?>
<?xml-stylesheet type='text/xml' href='test.xsl' ?>
<!-- This is a sample xml file created using VBScript. It is identical to a recipe file  -->
- <TAGLIST>
  - <TAG>
        <NAME>var1</NAME>
        <VALUE>100</VALUE>
    </TAG>
  - <TAG>
        <NAME>var2</NAME>
        <VALUE>200</VALUE>
    </TAG>
  - <TAG>
        <NAME>var3</NAME>
        <VALUE>300</VALUE>
    </TAG>
</TAGLIST>
```

VBScript code

```
Dim dom, root, node, subnode
Dim myFile

Sub xml_init
    Set dom  = CreateObject("Microsoft.XMLDOM")       ' Instantiate the XML Parser
    'Create a processing instruction (declaration) for XML
    Set node = dom.CreateProcessingInstruction("xml", "version='1.0'")
    dom.appendChild node
    Set node = Nothing
    'Create a processing instruction for xml-stylesheet (simple test.xsl). Note: this instruction not required
    Set node = dom.CreateProcessingInstruction("xml-stylesheet", "type='text/xml'  href='test.xsl' ")
    dom.appendChild node
    Set node = Nothing
    'Create a comment line
    Set node = dom.createComment("This is a sample xml file created using VBScript. It is identical to a recipe file")
```

```
    dom.appendChild node
    Set node = Nothing
    'Create the root element
    Set root = dom.createElement("TAGLIST")
    'Add the root element (TagList) to the DOM instance
    dom.appendChild root
End Sub

Sub writetag (strMytag, myval)
    ' Insert a newline + tab.
    root.appendChild dom.createTextNode(vbNewLine + vbTab)
    ' Create an node element Tag.
    Set node = dom.createElement("TAG")
    node.appendChild dom.createTextNode(vbNewLine + vbTab + vbTab)
    'Write a tag name and tag value to the XML document.
    Set subnode = dom.createElement("NAME")
    subnode.Text = strMytag
    node.appendChild subnode
    node.appendChild dom.createTextNode(vbNewLine + vbTab + vbTab)
    Set subnode = Nothing
    Set subnode = dom.createElement("VALUE")
    subnode.Text = CStr(myval)
    node.appendChild subnode
    node.appendChild dom.createTextNode(vbNewLine + vbTab)
    Set subnode = Nothing
    root.appendChild node
    Set node = Nothing
End Sub

Sub xml_close
    ' Adds a newline (blank line) at the end of the file.
    root.appendChild dom.createTextNode(vbNewLine)
    Set root = Nothing
End Sub

'Main Program
xml_init                    'initialize XMLDOM
writetag "var1", CInt($var1)        'Read IWS tags, send to XML file
writetag "var2", CInt($var2)
writetag "var3", CInt($var3)
xml_close                   'close out the XMLDOM
' Save the XML document to a file.
myFile = $GetAppPath() & "recipe1.xml"
dom.save myFile
Set dom = Nothing
```

We could subsequently load this XML recipe file into IWS using the IWS Recipe Manager. The procedure would be to create a Recipe Worksheet which defines the IWS tags (var1, var2 and var3), selects XML and UniCode, and points to the XML recipe file. Then, a button on the screen has its command properties set to execute the following VBScript code when pressed:

```
    $Recipe("Load:Recipe1.rcp")        'where Recipe1.rcp is the name of the saved Recipe Worksheet
```

Also note the use of the XML style sheet using test.xsl. This is a dummy transformation that can be used to allow the XML file to be read with Microsoft Internet Explorer.

Example 5 – Displaying contents of a XML Recipe File

This application displays the IWS tag names and values contained in an IWS Recipe (-compatible) XML file. This application assumes the XML file structure is similar to that in Example 4.

```
Dim dom, mylist, x

'Main Program
Set dom= CreateObject("Microsoft.XMLDOM")                'Instantiate the XML Parser
dom.load("recipe1.xml")                                  'Load the Recipe file
mylist = ""                                              'Null out the display string
For Each x In dom.documentElement.childNodes             'Traverse through the list of child nodes
    mylist = mylist & x.nodeName & ": " & x.Text & vbCrLf  'Add node names (IWS tag name) and values
Next
MsgBox mylist                                            'Display in a message box
```

Example 6 – Loading an XML file and saving to IWS tags

This application reads an IWS Recipe (-compatible) XML file and stores the correct values in the correct tag. This application assumes the XML file structure is similar to that in Example 4, and that the IWS tags are already defined in the IWS application.

```
Dim dom, objNodeList, myNode, mynode1, i
Dim curNode, nxtNode

'Main Program
Set dom= CreateObject("Microsoft.XMLDOM")                'Instantiate the XML Parser
dom.load ($GetAppPath() & "recipe1.xml")                 'Load the Recipe file
Set objNodeList = dom.getElementsByTagName ("TAG")
For i = 0 To objNodeList.length -1
    Set mynode = objNodeList.item(i).firstChild          'Get the tag name
    $tagprt = mynode.text                                'Store tag name in IWS tag pointer
    Set mynode1 = mynode.nextSibling                     'Get the tag value
    $@tagprt = mynode1.text                              'Using the IWS Tag pointer, store tag value
    Set mynode = Nothing
    Set mynode1 = Nothing
Next
```

XMLDOM Parsing Errors

So far, we have examined what the XML language is, the various objects, methods and properties, and VBScript applications to manipulate XML data files. But what if we use write an XML application incorrectly. Depending on what the error is, either VBScript will generate an error (compile or runtime error) or the XMLDOM Parser will generate a parsing error.

A well-formed XML document conforms to all of XML's syntax rules. If the document is not well formed, it is not considered to be XML and that parser stops processing it. Some of the XML syntax rules checked for in by the Parser include:

- Elements are properly nested and not overlapped
- One and only one root Element
- A non-empty Element has a start and an end tag
- An empty Element may be marked with an empty element tag

Example 7 – Displaying XML Parsing Errors in VBScript

This example shows how to use VBScript with XMLDOM to display XML parsing errors, such as when a non-existent file is attempted to be loaded, or when the XML is not well-formed.

```
Dim dom, myErr, strErr

'Main Program
Set dom = CreateObject("Microsoft.XMLDOM")
dom.load("wyxz.xml")
If dom.parseError.errorCode <> 0 Then
  Set myErr = dom.parseError
  strErr = "You have an Error" & vbCrLf
  strErr = strErr & "Error code: " & dom.parseError.errorCode & vbCrLf
  strErr = strErr & "Error reason: " & dom.parseError.reason & vbCrLf
  MsgBox strErr
End If
```

Using XML with ADO.NET and VBScript

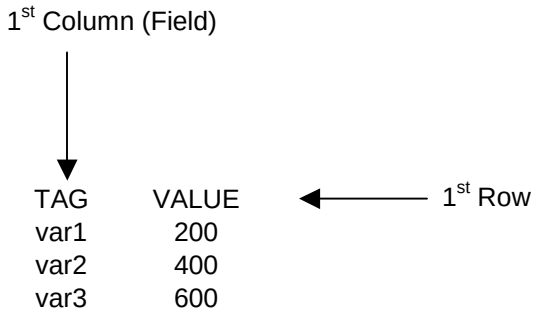
IWS supports ADO.NET, which in turn is built on the foundation of XML and its related technologies. Thus interaction between XML files and an ADO.NET database can be accomplished without much effort. However, an ADO.NET generated XML database includes lots of markup language and stores the XML data in a proprietary format that's optimized for XML representation, so viewing the XML file from a text editor, for example, is not possible.

SQL (structured query language) contains XML commands, depending on the database it is connected to. These commands can be executed against the database, usually resulting in a Recordset.

Example 8 – Displaying XML Parsing Errors in VBScript

Below is an example of how to use ADO.NET to read a Microsoft Excel data file that has a recipe stored in one of its worksheets. The ADO Recordset containing the Excel worksheet is then converted into XML format so that an IWS-compatible XML Recipe file can be created.

The Excel file is stored in a folder called "data" which is a subfolder of the project folder. This file can be located anywhere by changing the file path. The Excel file contains a worksheet named "Redcake". The contents of this worksheet is loaded into the ADO RecordSet. The function ConvertRStoXML[17] converts the ADO Recordset contents into an XML string (XML DOM), and is subsequently saved to an XML file.



Excel Worksheet "RedCake" contents

```
Dim myFile, myPath                                          'Declarations
Dim strCon
Dim objCon, objRS, objXMLDom
Dim xmlString

Function ConvertRStoXML(objRS, strRootNodeName, strRowNodeName)
    Dim objDom, objRoot, objRow, objRSField, objSubNode, objPI
    Dim i
    Set objDom  = CreateObject("Microsoft.XMLDOM")          'Instantiate XMLDOM
    objDom.preserveWhiteSpace = True
    Set objRoot = objDom.createElement(strRootNodeName)     'Define the Root Node
    objDom.appendChild objRoot                              'Add it to the DOM
```

[17] This program was adapted from http://www.4guysfromrolla.com/demos/CustomRStoXML.asp

```
    Do While Not objRS.EOF                                          'Iterate through the RS
        Set objRow = objDom.createElement(strRowNodeName)           'Define Tag Node
        i = 0
        For Each objRSField In objRS.Fields                         'Create Name Tag
            If i = 0 Then
                Set objSubNode = objDom.createElement("NAME")
                objSubNode.Text = objRSField.Value
                objRow.appendChild objSubNode
            End If
            If i=1 Then
                Set objSubNode = objDom.createElement("VALUE")      'Create Value Tag
                objSubNode.Text = objRSField.Value
                objRow.appendChild objSubNode
            End If
            i = i+1
        Next
        objRoot.appendChild objRow                                  'Append top level root
        objRS.MoveNext                                              'Next row in RecordSet
    Loop
    Set objPI = objDom.CreateProcessingInstruction("xml-stylesheet", "type='text/xml'  href='test.xsl' ")
    objDom.insertBefore objPI, objDom.childNodes(0)
    Set objPI = objDom.createProcessingInstruction("xml", "version='1.0'")   'Add Processing Instruction
    objDom.insertBefore objPI, objDom.childNodes(0)                 'Place in beginning of DOM
    ConvertRStoXML = objDom.xml                                     'Return XML contents
    Set objDom = Nothing                                            'Cleanup
    Set objRoot = Nothing
    Set objRow = Nothing
    Set objRSField = Nothing
    Set objSubNode = Nothing
    Set objPI = Nothing
End Function


'Main program
myFile = "recipes.xls"
myPath = $GetAppPath() & "data\" & myFile                           'Define source file
strCon = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & MyPath & ";"
strCon = strCon & "Extended Properties=""Excel 8.0;HDR=Yes"""       'Establish connection string
Set objCon = CreateObject("ADODB.Connection")                       'Instantiate ADO Connection object
Set objRS = CreateObject("ADODB.Recordset")                         'Instantiate ADO Recordset object
Set objXMLDom  = CreateObject("Microsoft.XMLDOM")                   'Instantiate XMLDOM
objCon.connectionString = strCon
objCon.Open                                                         'Open ADO Connection object
objRS.Open "Select * FROM [RedCake$]", objCon                       'Open ADO Record object
xmlString = ConvertRStoXML(objRS, "TAGLIST", "TAG")                 'Convert Recordset to XML
objXMLDom.loadxml (xmlstring)                                       'Load XML string into DOM
myFile = "recipe1.xml"                                              'Define XML file to store into
myPath = $GetAppPath() & "data\" & myFile
objXMLDom.save myPath                                               'Save XML file
Set objCon = Nothing                                                'Cleanup
Set objRS = Nothing
Set objXMLDom = Nothing
```

The following is a result of the execution of this code:

```
<?xml version="1.0" ?>
<?xml-stylesheet type='text/xml' href='test.xsl' ?>
- <TAGLIST>
- <TAG>
  <NAME>var1</NAME>
  <VALUE>200</VALUE>
  </TAG>
- <TAG>
  <NAME>var2</NAME>
  <VALUE>400</VALUE>
  </TAG>
- <TAG>
  <NAME>var3</NAME>
  <VALUE>600</VALUE>
  </TAG>
  </TAGLIST>
```

The above example can easily be modified to read from a variety of databases (e.g. Microsoft Access, SQL Server, MySQL). The main difference is the connection string used.

Summary

As this Application Note shows, there are several ways to read and write XML data from IWS using VBScript with Microsoft's XMLDOM and ADO.NET. Which method you use depends on what type of data you are extracting from IWS or loading from a separate file or PC. In addition to the examples shown, remember that alarm and event history files as well as historical trend data can be read and converted to XML using IWS built-in functions and XML tools discussed herein. XML-based custom reports can be easily generated.

Finally, there are some freeware XML editors such as Microsoft XML Notepad that you can use to check the contents of your XML.