XML SURVIVAL SKILLS FOR DBAS

John Jay King, King Training Resources

eXtensible Markup Language (XML)

XML is a set of rules for defining tags describing a document's structure and parts. To be precise, XML is a "meta-markup" language, providing the syntax used to define markup languages. A meta-markup language describes the syntax and structure of a document, not the presentation or the format. The XML 1.0 specification is authored by the W3C (World Wide Web Consortium). The World Wide Web Consortium (W3C) posts "7 points" explaining the group's goals and operating principles, go to the site and check them out at www.w3.org.

The World Wide Web Consortium (W3C) develops Web standards and technologies. W3C was founded in 1994 and has over 510 Members and nearly 60 full-time staff around the world. The web site includes a great deal of information regarding the W3C and the technology standards they promote including software. The term "markup" stems from typesetting days: publishers would "markup" the document provided by an author with instructions telling the typesetter how to format the page. In the computer world markup languages serve much the same purpose electronically; HTML and Troff are good examples. Most markup languages are fixed; markup features are designed into the language. For example, HTML provides a set of predefined "tags" used to format data(<title>,<body>,<p>,<h1>, etc…). XML is "extensible" because no tags are predefined. XML is a method to define your own tags or use tags created by others. All XML-based languages use the same syntax.

History of XML

XML is based upon markup language technology first created by IBM in the 1960's. Generalized Markup Language (GML) 1969 (IBM Text Description Language (TDL), renamed GML in 1971, not coincidentally, the team of three developers responsible for "GML" was: Charles F. Goldfarb, Ed Mosher, and Ray Lorie. The GML product was released by IBM in 1973. GML was enhanced in the late 70's IBM product Document Control Facility (DCF), also known as "Script" introducing Document Type Descriptors (DTDs) and Document Types. SGML (Standard Generalized Markup Language) became a reality in the early 1980s. While powerful, SGML was deemed too complex for everyday use, XML was created to provide a much simpler yet still powerful tool. In February 1998 the Worldwide Web Consortium (W3C) proposed a recommendation for XML 1.0. The XML 1.0 second edition was recommended in October 2000. XML and related standards may be found at the W3C website (www.w3.org). The XML 1.0 third edition was recommended in February 2004 as was the recommendation for XML 1.1. It is suggested that documents continue to follow the XML 1.0 standard unless some feature(s) only available in XML 1.1 are needed. Due to its recent availability, few tools support XML 1.1 yet.

XML and HTML

New XML developers often confuse HTML and XML. XML looks like HTML but is NOT HTML

HTML specifies what each tag means and how it will appear in the browser.

```
<html>
 <body>
     <h1>Introduction to XML</h1>
 </body>
</html>
```

XML tags describe chunks of data, it is up to the application that reads it to interpret it.

```
<xmlclasses>
 <class>
     <name>Introduction to XML</name>
     <author>John Jay King</author>
     <email>john@kingtraining.com</email>
 </class>
</xmlclasses>
```

XML rules are strict where HTML rules are often loose. HTML tags describe the presentation of data in a document, tags are predefined and have specific meanings to standard browsers. XML tags describe the content of data in a document, tags are determined by the document's creator and may be any valid name. The new XHTML (Extensible Hypertext Markup Language) standard is a hybrid of XML and HTML that provides precise HTML rules that conform to XML guidelines.

XML "Family" of Software

- XML 1.0 defines what tags and attributes are
- XHTML (eXtensible Hypertext Markup Language) is an XML application to replace HTML, essentially a version of HTML following the stricter XML rules.
- DTD (Document Type Definition) is a non-XML file describing the elements and syntax used by an XML document
- Schema is an XML file describing the elements and syntax used by an XML document
- XLink describes a method of using hyperlinks to reference XML
- XPointer describes hyperlinks that point to specific XML file elements
- JAXP (Java API for XML Processing) is a set of Java classes and interfaces used work with XML inside Java programs
- DOM (Document Object Model) provides access to XML document information
- SAX (Simple API for XML) provides access to XML document information
- XPath (XML Path Language) provides a mechanism for selecting XML document subsets
- XSL (Extensible Style Language) is an advanced style sheet language tailored of XML
- XSLT (XML Stylesheet Language Transformations) is used to transform XML to some other form (e.g. HTML)

Why XML?

XML is non-proprietary providing a standardized mechanism available to all vendors. Data may be standardized and stored in a shareable, standardized form. Elements and tags may be defined as needed allowing specialized languages for different disciplines. Document templates, files, and database data can all be stored using an XML-described format. Standardized formats make it easier to share data across various platforms, cultures, and languages. Industry groups and companies are working to use XML to build common tag sets to allow data exchange via common data structures.

XML is used for creating Markup Languages such as:

- Mathematical Markup Language (MathML)
- Synchronized Multimedia Integration Language (SMIL)
- Voice Extensible Markup Language (VoiceXML)
- Web Ontology Language (OWL)
- Web Services Description Language (WSDL)
- Speech Synthesis Markup Language (SSML)
- A P3P Preference Exchange Language (APPEL)
- Extensible HyperText Markup Language (XHTML)
- XML Path Language (XPath)
- XML Pointer Language (XPointer)
- XML Query Language (XQuery)

XML is frequently being used by software vendors to specify configuration information including: Databases, Web Servers, and Communications. Of course, XML is used to describe data files used for: Word processing, Electronic Data Interchange (EDI), Sequential data storage, and many other reasons.

XML Tags and Elements

XML provides the syntax necessary to define custom tags and with them custom elements. Tags are code delineated by "<" and ">" symbols:

```
Start tag   <name>
End tag </name> (note slash)
```

Empty tags incorporate the start and end together:

```
Tag  <name/>
```

Elements are a tag pair and their contents, as in the "name" element below. The tags and the contents together are the tag element.

```
<name>Geoffrey Holder</name>
```

Element Hierarchy

```
<xmlBooks>
 <book>
     <name>Learning XML</name>
     <author>Eric T Ray</author>
     <publisher>O'Reilly</publisher>
 </book>
 <book>
     <name>XML Bible</name>
     <author>Elliotte Rusty Harold</author>
     <publisher>IDG Books</publisher>
 </book>
 <book>
     <name>XML by Example</name>
     <author>Sean McGrath</author>
     <publisher>Prentice-Hall</publisher>
 </book>
</xmlBooks>
```

The "root" element above is <xmlBooks> (every XML document must have a "root" element). Three <book> elements are part of <xmlBooks>. Each <book> element includes three elements: <name>, <author>, and <publisher>.

Elements may be nested, start and end tags must entirely surround data. Indentation is optional, but, your coworkers will thank you...

Tag/Element Naming

XML has specific rules for naming of Tags/Elements. Element names must begin with a letter or an underscore. Element names may contain letters, underscores (_), numbers, hyphens (-), and colons (:). Start tags must match end tags exactly. Names in XML are case-sensitive and may not contain blanks. Officially there is no limit on the length of names, but, common sense should be used.

```
<name>Jones</lastname>                  incorrect
<lastname>Jones</lastname>              correct
<last name>Jones</last name>            incorrect
<lastname>Jones</lastname>              correct
<lastname>Jones</lastName>              incorrect
<lastName>Jones</lastName>              correct
```

Attributes

XML elements may use descriptive attributes. Attributes are added to an element's start tag using the name of the attribute followed by an equal sign, followed by the value of the attribute (surrounded by quotes or apostrophes).

```
<book isbn="0-13-960162-7" binding="perfect">
 <name>Learning XML</name>
 <author>Eric T Ray</author>
 <publisher>O'Reilly</publisher>
</book>
```

Attribute naming rules are the same as for elements, plus, attribute names must be unique within an element. Usually, attributes are used to provide information about the data in an element. Attribute values must be enclosed by matching quotation marks (") or apostrophes (').

"Well-Formed" XML

XML has a strict set of rules to determine that a document is "well-formed" or the parser will reject the file:

- Each document must declare itself an XML document using an XML declaration (frequently not enforced),
- Each document must have a single "root" element that completely contains all of the other elements in the document (one set of outer tags),
- All elements that include data must have both start <name> and end </name> tags,
- Empty tags are marked using a slash before the close of the start tag and omitting the end tag<name/>,

- Tags may not overlap, but, may be nested,
- Attribute values are surrounded by quotes (") or apostrophes (').

Most XML tools will refuse to process documents that are not well-formed

Character Entities

XML parsers assign specific meanings to certain characters (e.g. "<"). XML defines five "entity references" to allow documents to include the following characters:

- Ampersand &
- Apostrophe '
- Double quote "
- Greater than >
- Less than <

Each entity begins with an ampersand (&) and is ended by a semicolon (;).

```
<company>AT&T</company>     incorrect
<company>AT&amp;T</company>              correct
```

XML Processors

Software that reads and does something with XML is called an "XML Processor." Many web browsers, XML editors, and software products are XML processors. XML Processors typically include all or some of the following features: Parser translates XML markup and data into a stream of tokens, Event Switcher gets tokens from a parser and sorts them by function, Tree representation of XML document structure (maybe allowing manipulation of nodes), and a Tree processor that processes the XML tree for some purpose: transformation, and validity checking. At the least, an XML processor reads an XML document and converts it into a form that may be used by other software; this is called "Parsing" Parsers are the fundamental part of any XML processor.

XML processors provide several useful purposes, they are used to: read XML data, translate the data into recognizable tokens (the stream of characters is separated into instructions or hierarchical information), and assemble data into a hierarchy. By design Parsers are strict! All documents must be "well-formed" and any error aborts the parsing operation.

Oracle XML Support

Oracle's SYS.XMLtype is an Oracle-defined datatype used to store XML data within the database. The XML parser is part of the database. XMLtype may be used to store an entire document (unparsed) in the database, it may also be used to store a parsed document using relational columns for each XML element. Member functions include:

- createXML() Create XMLType instance
- existsNode() Checks if XPath can find any valid nodes
- extract() Uses XPath to return fragment as XMLType
- isFragment() Checks to see if document is really a fragment
- getClobVal() Gets document as a CLOB
- getStringVal() Gets value as a string
- getNumberVal() Gets numeric value as a number

Lots of XML support is added in Oracle9i, check the reference manual for more information:
"Application Developer's Guide – XML"

SQL provides several functions specifically for dealing with XML data including:

- SYS_DBURIGEN(ts) - Generate DBURITYPE URL used to obtain XML data from the database
- SYS_XMLGEN(exp) - Convert specified database row and column into an XML document
- SYS_XMLAGG(exp) - Generate single XML document from aggregate of XML data specified by "exp"
- XMLELEMENT(name,exp) - Generates XML element using name and exp as data

- XMLATTRIBUTES(exp,list) - Generates XML attributes using expression list

SYS_XMLGEN

The new SYS_XMLGEN uses a single input expression representing a particular row/column (scalar value or user-defined type). For scalar values a single XML element representing the value is returned. For user-defined types XML elements representing each of the user-defined type's data items is returned. SYS_XMLGEN returns an instance of XMLType data that is an XML document. The example on the next page displays using getStringVal since XMLType data returns as CLOB and is not displayable by SQL*Plus.

```
select sys_xmlgen(ename).getStringVal() Name
     from emp
     where job = 'ANALYST'
NAME
--------------------------------------------------
<?xml version="1.0"?>
 <ENAME>FORD</ENAME>
<?xml version="1.0"?>
 <ENAME>SCOTT</ENAME>
```

SYS_XMLAGG

SYS_XMLAGG aggregates all XML documents (or fragments of documents) for an expression and produces a single XML document. ROWSET is the default element name used. SYS.XMLGenFormatType may be used to change the output element name. The example below uses the SYS_XMLGEN function to generate an XML document for each dept 20 row of the sample EMP table. The example uses getClobVal since SYS.XMLType data returns as CLOB and is not displayable by SQL*Plus.

```
select sys_xmlagg(SYS_XMLGEN(Ename)).getClobVal() emps
        from emp
        where deptno = 10
EMPS
--------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
<ENAME>KING</ENAME>
<ENAME>CLARK</ENAME>
<ENAME>MILLER</ENAME>
</ROWSET>
```

XMLElement

XMLELEMENT(name,exp) Generates an XML element using name and exp as data:

```
 select xmlelement("employee",
        xmlelement("empid",empno),
        xmlelement("empname",ename)) myxml
 from emp
 <employee> <empid>7369</empid>          <empname>SMITH</empname> </employee>
 <employee> <empid>7499</empid>          <empname>ALLEN</empname> </employee>
 <employee> <empid>7521</empid>          <empname>WARD</empname> </employee>
 <employee> <empid>7566</empid>          <empname>JONES</empname> </employee>
 <employee> <empid>7654</empid>          <empname>MARTIN</empname> </employee>
```

XMLAttributes

XMLATTRIBUTES(exp,list) generates XML attributes using an expression list:

```
select xmlelement("employee",
       xmlelement("emp",
              xmlattributes(empno as "empno",
                            ename as "ename")),
       xmlelement("job",job),
       xmlelement("hiredate",hiredate),
       xmlelement("pay",
                xmlattributes(nvl(sal,0) as "sal",
                              nvl(comm,0) as "comm"))
       ) as myxml
from emp
```

```
<employee>
  <empno>7782</empno>
  <job>MANAGER</job>
  <job>MANAGER</job>
  <hiredate>09-JUN-81</hiredate>
  <pay sal="2450" comm="0"/>
</employee>
<employee>
  <empno>7839</empno>
  <job>PRESIDENT</job>
  <job>PRESIDENT</job>
  <hiredate>17-NOV-81</hiredate>
  <pay sal="5000" comm="0"/>
</employee>
```

Other XML Functions

- XMLColattval - Creates series of XML fragments using an element name of “column” and column names and values as attributes.
- XMLConcat - Concatenates a series of XMLType objects (opposite of XMLElement) XMLForest Creates XML fragments from a list of arguments/parameters.
- XMLSequence - Creates Varray of XMLType instances
- XMLTransform - Uses input XMLType and XSL style sheet (also XMLType) to create a new XMLType.

UpdateXML Uses an XMLType and an XPATH reference and returns an updated XMLType.

Style Sheets

XML describes data contents, not presentation. Style sheets are used to provide presentation specifications for XML documents. XML supports both CSS and XSL stylesheets. Cascading Style Sheets (CSS) have been supported by browsers for some time now and provide a tag-like language, but, is not XML itself. The eXtensible Style Sheet Language (XSL) is an XML application designed specifically for formatting XML documents! XSL uses XML syntax and more-modern browsers interpret XSL.

XML documents use a PI (Processing Instruction) to define the stylesheet type and name:

```
<?xml-stylesheet type="text/css" href="myBooks.css"?>
```

An XML document is a treasure-trove of data waiting to be used. Unfortunately, the format of XML data is easy for computers to understand and not so easy for the average human (too many characters & tags!). CSS gives us some formatting capability, but, it's not XML and can't take advantage of many of XML's features. The eXtensible Style Sheet Language (XSL) was designed specifically to transform XML data from one form to another: XML document to XML document, XML document to text, Text to XML document, XML document to PDF, XML database interactions, and more! eXtensible Stylesheet Language (XSL) stylesheets are well-formed and validated XML document themselves.

With XSL we can use eXtensible Stylesheet Language for Transformations. (XSLT) to display data in a different format than it is stored. XSLT can also trim away unwanted portions of the XML document so that the output has only what is required. XSLT is subset derived from XSL to use style elements for the purpose of transforming data from one format to another to describe the desired output for specific fields. Unlike CSS which generates HTML, XSLT generates a variety of output types including XML output. Java programmers might use the SAX or DOM object model to read and process XML transformations. XSL/XSLT will accomplish the same thing with the help of some XML-aware browsers! XSL and XSLT are the responsibility of the World Wide Web Consortium (W3C), please consult the W3C website for the latest information about XSL and XSLT

XSL/XSLT provide many elements to help in the transforming or presentation of XML data

The xsl namespace prefix is used when processing XSL/XSLT

Some of the Stylesheet elements include:

- xsl:template Creates a template “step”

```
<xsl:template match="/">
<xsl:template match="element">
```

- xsl:apply-templates Matches templates with input data and outputs when matches occur

```
<xsl:apply-templates
   select="element/subelement">
<xsl:apply-templates select="element">
<xsl:apply-templates />
```

Here is an XSL document that uses HTML tags to transform XML into a web page for output:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <xsl:template match="/">
 <html> <head> <title>XML Class List</title> </head>
  <body>
    <h1 align="center">XML Class List</h1>
    <table>
    <xsl:apply-templates select="xmlstudents/class"/>
    </table>
  </body>
 </html>
 </xsl:template>
 <xsl:template match="class">
    <tr> <td> <h2><xsl:value-of select="title"/></h2> </td>  </tr>
    <tr> <td>
      <h2>Number of days = <xsl:value-of select="numberdays"/></h2>
      </td> </tr>
   <xsl:apply-templates select="scheduledClass"/>
 </xsl:template>
 <xsl:template match="scheduledClass">
  <tr>  <td>  Class code = <xsl:value-of select="classcode"/>
            Date = <xsl:value-of select="date"/>  </td>  </tr>
  <tr>  <td> Location = <xsl:value-of select="location"/>
            Instructor = <xsl:value-of select="instructor"/> </td> </tr>
  <xsl:apply-templates select="student"/>
 </xsl:template>
 <xsl:template match="student">
  <tr> <td> <xsl:value-of select="name"/> </td> </tr>
  <xsl:apply-templates />
 </xsl:template>
</xsl:stylesheet>
```

The statement select="student" locates a specific node and apply-templates select="xmlstudents/class" matches up with the node xsl:template match="class. "

To use an XSL Stylesheet from an XML File the syntax is similar to when using a .css file.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
<xmlRootyTooty>
 <!-- xml document goes here -->
</xmlRootyTooty>
```

DTDs and Schemas and "Valid" Documents

To be well-formed: Each XML document must include an XML declaration PI, a root element that appears only once, each start tag has a matching end tag, and elements may not overlap. To be well-formed may not be enough, what if you want to share your XML with others? What guarantees that your elements will match those used by others? XML provides two mechanisms for validating XML files, Document Type Definitions (DTDs) and Schemas.

Document Tag Definition (DTD)

A Document Type Definition (DTD) specifies the elements a document must contain, the element sequence, and the contents of each element. Schemas (discussed in a later section) are also used for validation purposes.

The DTD was the first mechanism used in XML to ensure that document definitions matched. Using a common definition means that team members may collaborate by merging documents. Creating DTDs can be complex and DTD syntax is different from standard XML. Using DTDs ensure that XML documents follow rules. DTDs may be specified internally inside the XML file or externally in a separate file. A DTD is responsible for: naming the document type, defining each element a document might use. defining the data type of each element, defining the number of occurrences allowed for each

element (zero, one, many), and defining attributes used for each element and the allowed attribute values. XML Schema definitions (recommended by W3C in May 2001) are intended to replace DTD use eventually (XML Schemas are written using XML).

XML tags are allowed to be just about anything the document creator feels is reasonable. When passing XML documents it seems like a good idea for the recipient of a document to have a mechanism for validating the XML. DTD's provide a combination of Element and Attribute definitions that describe the contents of a type of XML document. Unfortunately, DTD syntax is unlike standard XML, instead it provides a specification for each element in an associated XML document.

DTD Syntax

DTDs include many features including:

- Document type: name of the document
- Elements: fields in the document
- Data type: PCDATA (parsed character) most common(tags will be interpreted)
- Field occurrences:
- Plus-sign (+) indicates required field, may appear more than once
- Asterisk (*) indicates optional field (0 or more times)
- Question mark (?) indicates the field is optional but may only occur once if present
- Default is for an element to occur exactly once
- Attributes for a field and permissible values may be specified
- DTDs may use standard XML comments <!-- comment -->

DOCTYPE is the high-level (root) element in a DTD, it may either include the DTD text or reference a URI that points to a file containing the DTD. If the DTD is included in the same file as the XML document, the DOCTYPE and associated specifications are included at the top of the file (might use standalone="yes" if no external files are used).

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE xmlstudents [
<!ELEMENT xmlstudents (class+)>
<!ELEMENT class (title, numberdays, scheduledClass+)>
 ...
<!ELEMENT student (name)>
<!ELEMENT name (#PCDATA)>
]>
<xmlstudents>…
</xmlstudents
```

Sample DTD

```
<?xml version="1.0"?>
<!ELEMENT xmlstudents (class+)>
<!ELEMENT class (title, numberdays, scheduledClass+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT numberdays (#PCDATA)>
<!ELEMENT scheduledClass (classcode,date,location,instructor,student*)>
<!ELEMENT classcode (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT instructor (#PCDATA)>
<!ELEMENT student (name)>
<!ELEMENT name (#PCDATA)>
```

Schemas

In an effort to bring everthing into the XML umbrella, the W3C has created a new, improved method for validating XML documents called an XML Schema. Schemas are well-formed XML documents themselves that describe the XML document's format. With Schemas, XML documents and their format descriptions use the same basic formatting rules (XML), perhaps making it easier to work with both. Schemas are used for XML document validation. Schemas are also useful as documentation tools, since they follow the rigid XML standard they are machine-readable! As members of the XML family are updated (e.g. XPath, XSLT, XQuery), Schemas are incorporated into their design.

Problems with DTDs

- DTDs have their own syntax, different from standard XML
- DTD's have no mechanism for specifying the type of data that belongs in a field other than whether data is a character string or a group of elements

XML Schema Syntax

XML Schemas are XML documents themselves, like the one below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="xmlstudents">
     <xs:complexType>
            <xs:sequence>
           <xs:element ref="class"
               maxOccurs="unbounded"/>
         </xs:sequence>
     </xs:complexType>
 </xs:element>
 <xs:element name="class">
     <xs:complexType>
         <xs:sequence>
             <xs:element ref="title"/>
             <xs:element ref="numberdays"/>
             <xs:element ref="scheduledClass"
                  maxOccurs="unbounded"/>
         </xs:sequence>
     </xs:complexType>
 </xs:element>
 <xs:element name="title" type="xs:string"/>
 <xs:element name="numberdays" type="xs:byte"/>                                    ...
 <xs:element name="classcode">
     <xs:simpleType>
         <xs:restriction base="xs:short">
             <xs:enumeration value="1504"/>
             <xs:enumeration value="1508"/>
             <xs:enumeration value="1511"/>
         </xs:restriction>
     </xs:simpleType>
 ...
</xs:schema>
```

To reference an XML Schema from an XML document, modify the root element to include the schema:

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet href="xmlstudents.css" type="text/css" ?>
<xmlstudents xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="xmlstudents.xsd">
 <class>
     <title>Introduction to XML</title>
...
```

The stylesheet reference above uses *XML Namespace* notation (beyond the scope of this paper) to identify a namespace prefix "xsi" referring to a particular version of the XML Schema standard.

XPath and Xlink

XML Path Language (XPath) is designed to provide quick and easy access to any node in our document's hierarchy. Having lots of information stashed away in XML is of little use if we cannot get to data when it is needed, XPath to the rescue! XPath provides a mechanism to address any element or attribute.

XLink (XML Linking Language) provides a hyperlink-type capabilty to XML. The XLink specification is in a state of flux and is not supported by many browsers and tools, it is most useful from within programs. XLink allows links to be constructed using any element (remember, XML does not have any pre-defined element tags). XLinks build on the capability of HTML linking and avoid some of the problems.

XML and JSP (User Tag Libraries)

A JSP developer writes HTML normally and encloses Java code in special JSP tags. JSP tags are coded into HTML and start with "<%" and end with " %>". The separation of Java and HTML makes code more useful to non-technical web designers, JSP Custom Tags allow the elimination of most, if not all, Java code from JSPs.

Custom Tags are probably the most powerful feature of JSPs. Web Page designers may concentrate on using tags to create functional web sites. Developers may concentrate on the nitty-gritty details necessary to make the tags work. A few simple steps are required to create and use Custom Tags:

- Create Java class to provide the low-end code
- Create a Tag Library Descriptor (TLD) connecting the .class file (or .jar file) to a tag name
- Reference and use the TLD
- Use Taglib to connect TLD to a prefix used in the JSP
- Use prefix and tag name in JSP source

XML and Java (DOM, SAX, SOAP)

Java is a relatively new programming environment promising portable code; write-once, run-everwhere (officially released in 1995). XML's text base and standardization make it the ideal complement to Java providing portable data; store-once, use-everywhere. XML and Java seem made for each other!

Programs use XML via Application Programming Interfaces (APIs). Low-level APIs allow the programmer to deal directly with the XML document and its data. DOM, SAX, and JDOM are the most commonly-used low-level APIs today. JAXP (Java API for XML Programming) is relatively new and is becoming popular. High-level APIs provide a simpler interface that calls one of the lower-level APIs "under-the-covers. " High-level APIs tend to be easier to develop with but usually add processing costs (no free-lunch!). XML data binding is an example of a high-level interface.

Application Programmer Interfaces (APIs)

- DOM (Document Object Model) has been around for many years and is frequently used
- SAX (Simple API for XML) is newer than DOM and offers many Java-specific features
- JDOM (Java Document Object Model) is a Java-specific API tailored specifically to the needs of Java programmers
- JAXP (Java API for XML Programming) is really a higher-level API designed to take some of the complexity out of using DOM, SAX, or JDOM

DBA's XML role

The DBA's role in XML development, deployment, and execution is crucial. Database access must be granted to applications and database object using XML must be created. DBAs are the "answer-people" to whom programmers and users bring any manner of programming or SQL problem remotely connected to the database, so, XML is one more tool to be aware of. DBAs must make sure that XML developers access the database as efficiently as possible and help create designs that provide reasonable performance. Finally, most Oracle and other databases are shipping configuration files in XML format and DBAs must be able to understand the syntax in order to make the appropriate modifications.

Conclusion

XML provides portable data to compliment the portable programming that Java provides. DBAs must become aware of the basics of XML and its use. XML is increasingly being used for data interchange and configuration. As developers begin to use XML DBAs must be ready to support and improve their use of XML.

About the Author

John King is a Partner in King Training Resources, a firm providing instructor-led training since 1988 across the United States and Internationally. John specialized in application development software on a variety of platforms including Unix, Linux, IBM mainframe, personal computers, and the web. John has worked with Oracle products and the database since Version 4 and has been providing training to Oracle application developers since Oracle Version 5. John develops and

presents customized courses in a variety of topics including Oracle, DB2, UDB, Java, XML, .NET, and various programming languages. He has presented papers at various industry events including: IOUG-A Live!, UKOUG Conference, EOUG Conference, AUSOUG Conferences, RMOUG Training Days, OOUG, TOUG, MAOP-AOTC, NYOUG, and the ODTUG conference.

John Jay King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627
U.S.A.
Phone: 1.303.798.5727 1.800.252.0652 (within the U.S.)
Fax: 1.303.730.8542
Email: john@kingtraining.com
Website: www.kingtraining.com