

# ***XML Web Performance***



**Presented by: John Jay King**  
King Training Resources  
john@kingtraining.com

**Download this paper from:**  
<http://www.kingtraining.com>



- Understand XML document processing
- Learn different factors impacting XML performance
- Be able to improve the performance of XML documents



- John King – Partner, King Training Resources
- Providing training to Oracle and IT community for over 20 years
  - Databases: Oracle, DB2, SQL Server, more...
  - Languages: PL/SQL, Java, C#, COBOL, PL/I, more...
  - Operating Systems: Linux, Unix, Windows, z/OS
  - Tools: XML, HTML, JavaScript, more...
- Leader in Service Oriented Architecture (SOA) design and implementation
- Home is Centennial, Colorado – I like to hike and drive in the mountains



- XML is a set of rules for defining tags to describe a document's structure and parts
- XML is a "meta-markup" language, providing the syntax used to define the syntax and structure of a document, not the presentation or the format
- The XML specification is authored by the W3C (World Wide Web Consortium [www.w3.org](http://www.w3.org))
- "Markup" is from typesetting: publishers "markup" a document telling the typesetter how to format the page
- XML is "extensible" because no tags are predefined; organizations/industries define tags for "XML Language" to support business needs



- Is XML Cheaper? Smaller? Faster?
- Is XML more-capable?
- XML is STANDARD
  - Character strings are common to all operating systems and computer languages
  - XML standardizes the character string formats exchanged between programs and systems
- Savings attributed to XML are usually due to reduced complexity (Standards for Technology in Automotive Retail (STAR))



- Savings attributed to XML are usually due to reduced complexity (Standards for Technology in Automotive Retail (STAR))
  - STAR was used to standardize interchange of parts and automobile information between automobile manufacturers, suppliers, dealers, and credit organizations
  - Millions in estimated savings due to standardization



- XML is non-proprietary
- Elements and tags may be defined as needed allowing specialized languages
- Document templates, files, and database data can all be stored using an XML-described format
- Standardized formats make data easier to share
- Industry groups and companies use XML to build common tag sets and common data structures
- XML is frequently used by software vendors to specify configuration
- XML is used to describe data files used for: Electronic Data Interchange (EDI), word processing, and more



- Industry groups and companies use XML to build common tag sets and common data structures
- XML is frequently used by software vendors to specify configuration
- XML is used to describe data files used for: Electronic Data Interchange (EDI), word processing, and more





- XML is designed to be “human readable”
  - Verbose tag names are used to aid readability
  - Design “best practices” lead to use of sub-elements to maximize flexibility
  - Hierarchical nature of XML allows deep structures of elements, sub-elements, sub-sub-elements and so on
  - Tab characters and carriage-returns (end-of-line markers) are added to support readability
- Result: XML documents take up too much space!  
XML documents take too long to parse!

**Nobody should need to read XML documents manually  
(except when problem-solving)**



- XML has specific rules for naming of Tags/Elements
- Element names 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 length limit)
- Names should not begin with “xml”

```
<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
```



- Descriptive 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&apos;Reilly</publisher>  
</book>
```

- Attribute naming rules are the same as for element naming, 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 quotation marks (") or apostrophes (')



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

143 bytes

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

179 bytes

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

154 bytes



- Are Elements better than Attributes or vice-versa?
  - Some feel attributes should be used for metadata or for unchangeable values (primary keys)
  - Some feel that elements are more flexible long-term
  - Some feel that attributes without too many elements reduces file sizes
  - Attributes may not be divided in any way  
(Elements may have sub-Elements and Attributes)
  - Attributes may not be repeated within an element  
(Elements may be repeated as specified by Schema)
- Choose the mechanism that best seems to fit the business rules of your system



- Strict rules determine that XML is "well-formed":
  - Document should declare itself using an XML declaration  
`<?xml version="1.0" encoding="UTF-8"?>`
  - A single "root" element must completely contain all 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/>`  
(usually include attributes `<name first="Al" last="Ono" />`)
  - Tags may not overlap, but, may be nested
  - Attribute values enclosed in quotes (") or apostrophes (')
  - XML tools refuse to process non-"well-formed" documents



- Software processing XML must make sure the document is “well-formed” before processing (a requirement of the XML standard)
  - XML document must be stored to make sure it is “well-formed” (large documents take up space)
  - Computer resources must be used to ensure that rules for “well-formed”-ness are enforced



- SOAP message received
  - SOAP processor checks if message **well-formed**
  - Process SOAP message and retrieve payload
  - If payload is XML it is checked to be sure it is **well-formed** before being passed to program
  - Pass payload to program
  - Program's parser checks if document **well-formed**
  - Program processes file
  - Program generates XML result
  - Parser checks to see if result is **well-formed** before passing to SOAP
  - SOAP parser checks XML to make sure it is **well-formed**
  - Process repeated on other end...





- Being “well-formed” does not mean that the XML document being transmitted/received is correct!
- W3C has a method for validating XML documents called XML Schema (previously used “DTDs”)
- Schemas are well-formed XML documents themselves describing an 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 also useful as documentation tools, since they follow the rigid XML standard they are machine-readable!



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="myStudents">
    <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:schema>
```



```
<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>
```



- XML Schemas are sometimes referenced from an XML document's root element:

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



- XML Path Language (XPath) is designed to provide quick and easy access to any node in our document's hierarchy
- Having lots of information 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
- XPath is a World Wide Web Consortium (W3C) standard: <http://www.w3.org/Style/XSL>



- XPath nodes are very similar to DOM (Document Object Model) nodes; being sometimes-complex webs of nested elements and attributes
- An XPath address points to a specific end-point (position) in the hierarchy, or, to a specific node
  - Location paths may be absolute or relative:
    - Absolute paths start at the root node and move out from there
    - Relative paths start at a preselected spot called the context node (usually both easier and faster); context node is the current location
  - Each step in a location path is divided into three parts:
    - Axis, describes the direction of travel through the nodes
    - Node Test, to select the desired nodes
    - Predicates, optional tests to eliminate some XML from output



- The size of XML documents impacts performance if many documents are being transmitted/stored
- XML Processing for “well-formed” testing cannot be avoided
- XML Processing for schema validation is optional in most XML environments and may be avoided if unnecessary (careful!)



- Techniques exist to minimize the size of XML documents
- Smaller XML documents reduce:
  - Bandwidth requirements
  - Memory space requirements
  - XML Parsing time
  - Storage requirements
- XML optimization techniques and savings apply to just about any XML document





- XML has become the "lingua franca" of Information Technology data transfer used as:
  - EDI (Electronic Data Interchange) messages
  - Web Service messages
  - Transaction messages
  - Configuration data
- XML processing causes performance issues in systems passing many XML documents or very large XML documents



- XML optimization should not corrupt the integrity of the XML document
- XML optimization techniques reduce the size of XML documents while keeping the “human readable” goal of XML (though admittedly less-readable)
- Software and hardware products are available that will convert XML documents to proprietary binary forms to speed processing; eliminating the advantages of standardized XML (not discussed further here...)



- XML Optimization Techniques include:
  - Elimination of whitespace
  - Elimination of comments
  - Careful use of CDATA sections
  - Reducing name lengths for: Element, Attribute, and Namespace
  - “Flattening” Structure
  - Careful use of Schema validation



- While human readability is a stated goal of XML
- In most cases XML is transmitted between computer programs and humans need to read it only when something goes wrong
- Eliminate extra blank spaces
- Eliminate indentation
- Eliminate tab characters
- Eliminate carriage-return line-feed characters



- Document with carriage returns and tabs

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

- Document with whitespace reduced (saves 8 chars at least)

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



- XML comments are useful when describing an XML document to a human
- Many good XML document designers include comments explaining the roles of Elements and Attributes
- Most comments are “whitespace” in production use and may be eliminated



- XML allows documents to include text that might contain XML formatting characters
- CDATA sections are simply passed by XML processing without any attempt to check them for “well-formed”-ness
- CDATA sections should be as short as possible (may be tricky)



- XML documents being used as Web Service messages or transactions sometimes contain information that is redundant at best and potentially incorrect
- For example if a purchase document includes:
  - Item ID, Item Description, Item Size, and Item weight in addition to quantity purchased, price, and date
  - Chances are Item ID may be used by the programs on either end of the transaction to look up Item specifics
- Consider eliminating redundant Elements and Attributes





- Since human readability is a stated object of XML; names tend to be lengthy
- Element names are repeated in Start `<name>` and End `</name>` tags
- Namespace prefixes are repeated each time they are referenced `<myNamespace:myElement>`
- In most systems XML documents are “seen” only by computer programs; why not make names as short as possible?
- Consider reducing names to 1-3 character meaningful mnemonics so that the document may still be read in the event of problems



- Names at original length

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

- Abbreviated names (reduced 34 characters)

```
<bk>
  <isbn>0-13-960162-7</isbn>
  <bdg>perfect</bdg>
  <top>IT XML</top>
  <nam>Learning XML</nam>
  <auth>Eric T Ray</auth>
  <pub>O' Reilly</pub>
</bk>
```



- XML Documents with a structure of Elements, Sub-Elements, and Sub-Sub-Elements are said to be “vertical”

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

- XML Document structures using Attributes rather than Sub-Elements are said to be “horizontal”

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



- XML Documents with a structure of Elements, Sub-Elements, and Sub-Sub-Elements are said to be “vertical”
- XML Documents with a structure using Attributes rather than Sub-Elements are said to be “horizontal”
- If Elements are optional or limited to one occurrence; Attributes may be used rather than Sub-Elements without changing the intent of the XML designer (“flattening” the vertical document)
- Using XML “Empty Elements” the ending Element tag is eliminated:

```
<name first="Al" last="Orr" />
```

- Eliminating unnecessary Sub-Elements (vertical depth) reduces the number of tags required



- Careful use of Schema validation can reduce processing time
  - Scheme validation is optional
  - If an XML document is being used in a closed-loop system; validation may be redundant and wasteful
  - Complex Schema validations take longer than simple Schema validations
- Make sure that the complex Schema testing is really necessary



- The Parser used to process XML can have a dramatic impact on performance
- As usual in IT there is no “one-size-fits-all” solution
- Many types of Parsers are available, the most common are:
  - DOM
  - SAX
  - JDOM
  - StAX



- Software that reads/uses XML is called an “XML Processor”
- Many web browsers, XML editors, and software products are XML processors
- Some features often provided in XML Processors:
  - Parser Translate XML markup & data into tokens
  - Event Switcher Sorts/routes tokens to event handler or Call-back procedure
  - Call-back procedures Responds to events and adds nodes to “tree”
  - Tree representation Persistent hierarchy of XML document, may allow manipulation
  - Tree processor Code that processes the XML tree



- Parsers are the fundamental part of any XML processor
- Parsers 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)
  - Assemble data into a hierarchy





- All documents must be “well-formed”
  - Start tags must have End tags (or be empty)
  - Start and End tag names must match exactly (case too!)
  - Ambiguous names are not allowed
- By standard, XML parsers are not allowed to “fix” things
- Any error aborts the parsing operation
- Strict parsing means that a successfully parsed document is predictable and reliable



- Many parsers are available, including:
  - Xerces (Java, C++, Perl)
  - JAXP (Java)
  - IBM XML Parser for Java
  - Oracle XML Parser (Java)
  - Microsoft MSXML (C++, C#, JavaScript, VB, .NET, Java, Perl, Python)
  - IBM Enterprise COBOL
  - XML::Parser (Perl)
  - IBM Alphaworks XML for C++
  - Xparse (Python)
- Get a more-complete list of available parsers at:  
<http://wdvl.com/Software/XML/parsers.html>



- 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 also 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 (JAXB) is an example of a high-level interface



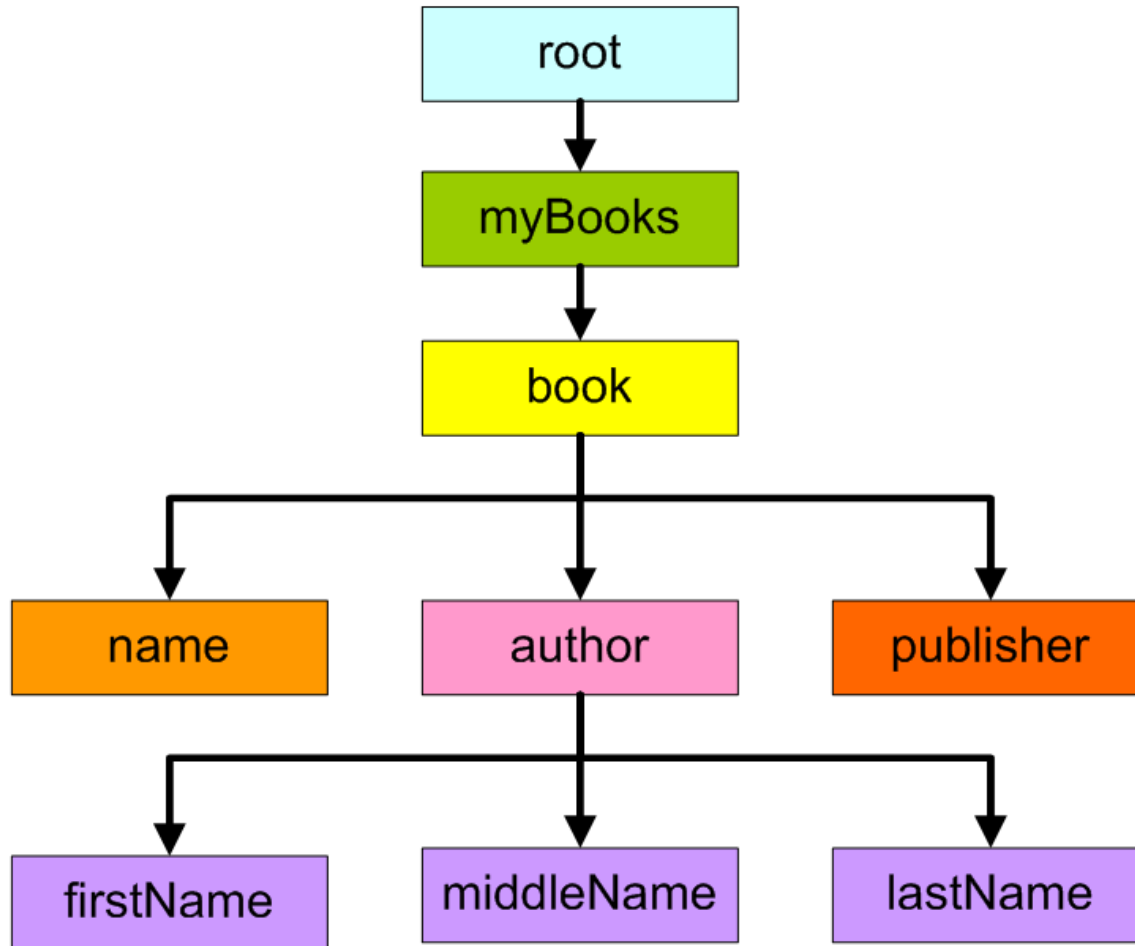
- DOM, SAX, JDOM, and JAXP all offer low-level Application Programming interfaces:
  - DOM (Document Object Model) has been around for many years and is frequently used
  - SAX (Simple API for XML) offers the most basic Java-specific features
  - JDOM (Java Document Object Model) is a Java-specific API 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



- The three most commonly used Java-XML APIs today are SAX, DOM, and JDOM:
  - SAX allows the quickest possible processing of an XML document since data is read and written in a continuous stream; however, it is up to the programmer to make some sense of what is being processed
  - DOM reads an XML document into memory and creates a hierarchical "tree" structure that may be referenced in a reasonably simple fashion, DOM has been available for quite some time and is sometimes criticized for not being Java-like
  - JDOM is the brainchild of a Java programmer who wanted the convenience of DOM but also wanted a pure-Java mechanism
  - Other APIs exist, these are simply the most common



- DOM is a recommendation of the W3C
- DOM creates an object-tree that is very useful for parsing and processing XML's hierarchical data
- DOM is both platform and language agnostic and is heavily used in Java, C++, and JavaScript environments
- DOM parsers read XML documents and organize the data in memory into a “tree” structure of objects
- DOM then uses the “tree” for processing
  - Tree has a root that encompasses the entire document
  - Programs may navigate the branches of the tree
  - Nodes of the tree may be read and/or modified





- SAX is the basic mechanism for Java-XML programming (also IBM Enterprise COBOL)
- SAX reads an XML document and passes the document's elements one at a time to the program
- SAX uses “event-based” parsing where values are read and presented to the program using a method created by the program
- When using SAX at least two Java classes are involved:
  - Controlling class      Uses “Content Handler” to process XML document
  - Content Handler      Reads and processes XML data
- SAX might also involve an Event Handler class and/or an Error Handler class





- Content handlers implement the ContentHandler interface and must include code for “events”:
  - setDocumentLocator Get object for finding SAX events
  - startDocument Begin XML document
  - processingInstruction Examine PI's (except xml PI)
  - startPrefixMapping Map prefix to namespace
  - startElement Start XML element
  - characters Process element characters
  - endElement End of XML element
  - endPrefixMapping Stop mapping prefix to namespace
  - ignorableWhitespace Return contiguous whitespace
  - skippedEntity Return name of skipped entity
  - endDocument End of XML document



- SAX parsers read a document as events:

**start document**

**start element: myBooks**

**start element: book**

**start element: name**

**text: "Learning XML"**

**end element: name**

**start element: author**

**start element: lastName**

**text: "Ray"**

**end element: lastName**

**start element: firstName**

**text: "Eric"**

**end element: firstName**

**start element: middleName**

**text: "T"**

**end element: middleName**

**end element: author**

**start element: publisher**

**text "O'Reilly"**

**end element: publisher**

**end element: book**

... <more "book" elements> ...

**end element: myBooks**

**end document**



- JAXP (Java API for XML Parsing) was released by Sun in an attempt to make Java XML parsing simpler through abstraction
- Many programmers do not realize a new parser is being used since Sun's parser is downloaded when people download JAXP
- JAXP is really an API and is not a parser
- JAXP makes it easier to use the existing SAX, DOM, and JDOM APIs



- JAXP works with SAX using the SAXParser class
- SAXParser objects are created using a SAXParserFactory object
- SAXParser and SAXParserFactory objects use the available SAX parser to accomplish their work
- Generic factory methods are used to initiate validation and namespace recognition
- Generic methods are used to test current settings
- The parse() method allows specification of a DefaultHandler object or HandlerBase object to handle document events



- JAXP with DOM is pretty much the same as using JAXP with SAX only using slightly different class names
- The DocumentBuilderFactory and DocumentBuilder classes seen earlier are used to generically access the available DOM parser



- JAXB (Java Architecture for XML Binding) provides a “bridge” between XML and Java, it uses APIs like SAX and DOM it does not replace them
- JAXB maps XML to Java objects
- JAXB uses a DTD at the same level of the XML
  - Maps XML to Java data objects
  - JAXB acts as a code-generator using a DTD or Schema to generate Java class code mapping XML to internal data objects
  - JAXB uses a DTD or Schema to “bind” to a set of generated classes
  - The binding schema is an XML-based binding language

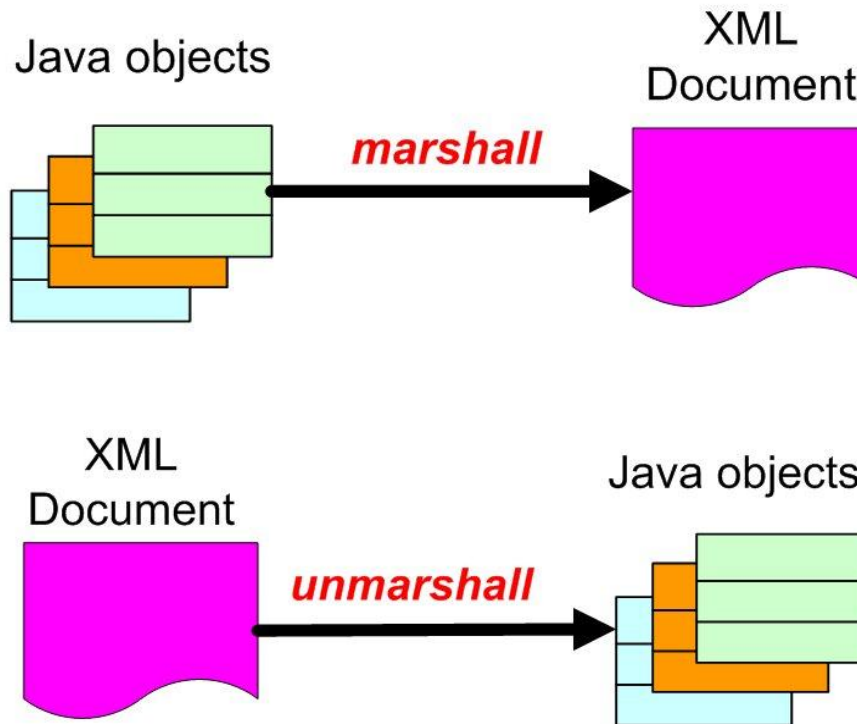


- Code is much simpler “hiding” the complexity of XML parsing
- SAX and DOM are generic XML parsers and parse any well-formed XML
- JAXB creates a SAX or DOM parser that is specific to your DTD or Schema and parses only valid XML
- JAXB produces a “tree” in memory specific to the Elements and Attributes defined by your DTD or Schema
- In early releases JAXB worked with DTD only



- JAXB is based upon data binding which includes:
  - Java source file and class generation  
Using utility program using DTD or Schema as input to produce Java class definitions matching document
  - Unmarshalling  
Taking data from XML document into Java objects
  - Marshalling  
Moving data from Java objects into XML document
  - Binding Schemas  
Rules for generating Java classes







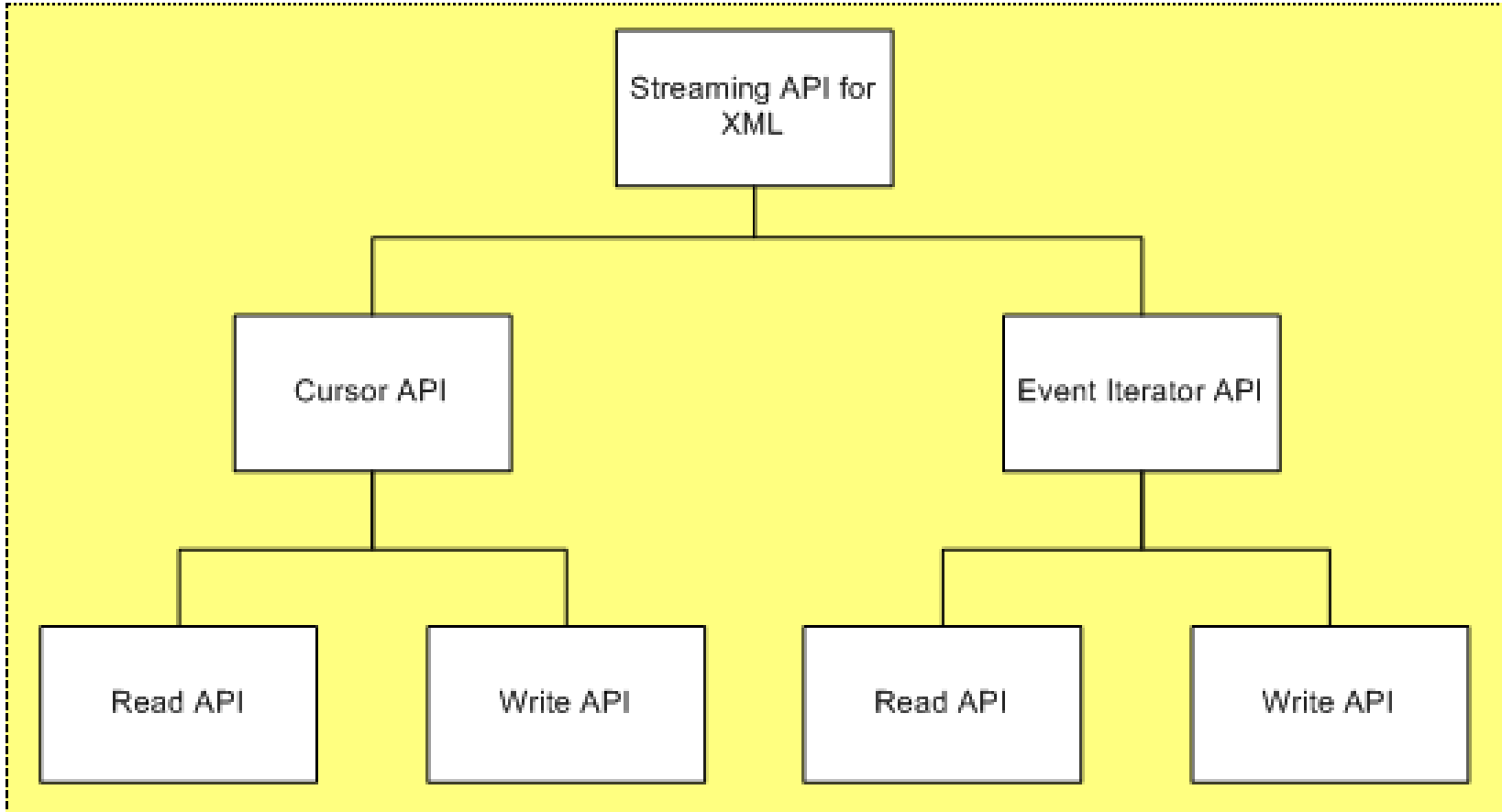
- Before StAX programmers had only two likely choices when processing XML documents:
  - Reading/writing streamed XML content (SAX)
  - Building the entire document in memory before processing (DOM)
- Both SAX and DOM have drawbacks:
  - SAX programs must carefully track progress through the document creating miscellaneous structures as needed
  - DOM stores the entire document in memory using a generic "tree" structure that is easy to use but bulky
- Streaming APIs like SAX are called "push" APIs; they shovel data to the program whether the program is ready or not based upon events



- StAX is even simpler than SAX, it is a "pull" type API that allows the programmer to control parsing via an iterator-based API and stream of events
- StAX provides both a low-level cursor API and a higher-level event-iterator API
  - Cursor-based API (based upon XMLStreamWriter); best for creating a document from application data
  - Event-based API (based upon XMLEventWriter); best when creating a new document based upon multiple existing documents



- StAX is a newer style of API, a pull API
- Pull APIs are streaming APIs and very fast, but are also memory efficient
- The program is in control asking the parser for the next part of the document the program wishes to process  
(the program pulls data from the document)





- The cursor-based API uses a virtual cursor to process the XML document
- When reading, the `XMLStreamReader` is created using an `XMLInputFactory`
- The API has a built-in iterator using `hasNext()` and `next()` methods to walk through the document
- `getText()` and other methods may be used to obtain information from the current element, attribute, or text
- When writing with the Cursor API the `XMLStreamWriter` provides methods used to write elements, attributes, and text to the XML file
- `XMLStreamWriter` is created using the `XMLOutputFactory`



- The event iterator API reads and writes XML data too
- A new factory, `XMLEventFactory` builds events to be processed
- Once again a built-in iterator using `hasNext()` and `nextEvent()` methods is available to process various events
- When writing, the `XMLEventFactory` creates events that may be added to the output with the `XMLEventWriter` object



- Pull Parsing is the wave of the future for many applications
- Stax grew out of the need to read and write XML in an efficient manner in the context of XML Binding and Web Services
- XML Pull Parsing is touted as a high performance alternative to DOM for XML parsing that is easier to use than SAX
- SAX is a push API and used more widely than any other push API currently in use
- So far many pull APIs have been created and only recently has the industry coalesced around a single one (StAX)
- Choosing between tree oriented (DOM), streaming push (SAX), and pull (StAX) parsers it is important to understand the limits and advantages of each





- Clearly there are many issues involved in processing XML data; choose the parser which best suits your situation
  - DOM Small-medium size documents; easy to use, supported widely
  - SAX Longer documents, more difficult to use, supported widely
  - StAX Longer documents, can be difficult to use, not available in all environments
  - JAXB Use SAX or DOM “under the covers”, requires Schema/DTD, Java-only, easier programming
  - JAXP Use SAX or DOM “under the covers”, easier than DOM/SAX/StAX not as easy as JAXB, Java-only



- Oracle's XML support is provided as XML DB:
  - W3C (Worldwide Web Consortium) XML compliance
  - XMLType is an Oracle-defined datatype storing XML data
    - Unstructured (CLOB underneath)
    - Structured (“Shredded” into relational columns and rows)
    - Binary XMLType (new with Oracle 11g)
  - The XML parser is part of the database
  - Oracle provides several XML-oriented SQL functions to support XML, some support the emerging ISO/ANSI SQLX initiative
  - Check the reference manual for complete information:  
"XML DB Developer's Guide"



- XMLType may be used to represent a document or document fragment in SQL
- XMLType has several built-in member functions to operate on XML content
- XMLType may be used in PL/SQL as variables, return values, and parameters
- XMLType APIs are provided for both PL/SQL and Java programming
- XMLType is also supported on the client via FTP, HTTP, and WebDav



- Caution! Your results may be different than mine
- Please confirm any “improvements” you make by testing thoroughly in the actual runtime environment



- Can I Improve XML Web Performance? -- Yes!
  - Eliminate non-essential whitespace including tabs and carriage-return line-feed characters
  - Eliminate non-essential Elements and Attributes
  - Reduce size of Element and Attribute Names
  - Consider “flattening” document by converting Elements into Attributes where practical



**Training Days 2010**

**Save the dates!**

**February 17-18 2010!**



**Save the Date: April 18-22 2010**

**Las Vegas, Nevada!**





- Web Architecture
- Oracle Tools
- Professional Development
- Web and Java
- Essbase
- Application Express
- Web and Not Java
- Hyperion
- Business Intelligence and Data Warehousing
- Database Server
- BEA
- Best Practices
- Third Party Tools

**Monterey, CA**  
**June 21–25, 2009**

**SPOTLIGHT ON  
DEVELOPERS**

[www.odtugkaleidoscope.com](http://www.odtugkaleidoscope.com)





## XML Web Performance

To contact the author:

**John King**

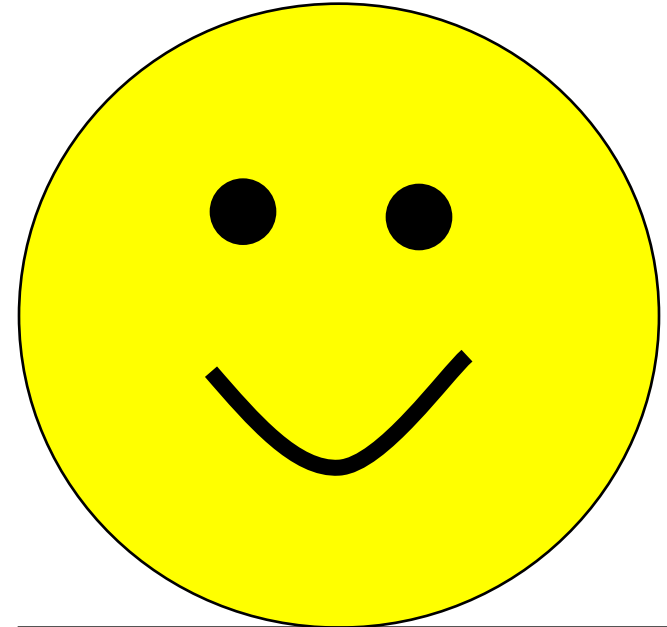
**King Training Resources**

6341 South Williams Street

Littleton, CO 80121-2627 USA

1.800.252.0652 - 1.303.798.5727

Email: [john@kingtraining.com](mailto:john@kingtraining.com)



**Thanks for your attention!**

Today's slides are on the web:

<http://www.kingtraining.com>