A Detailed Introduction to Parsing and Processing XML Documents with Java™ Technology – Part 1

Michael Paolini

Linux Integration Center, Austin Texas, USA

IBM Software Group

paolini@us.ibm.com

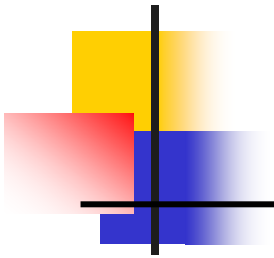
Presenter:

Neil Graham

XML Parser Development

IBM Software Group

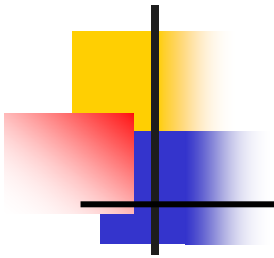
neilg@ca.ibm.com



Session Objectives – Part 1

"Part 1 of 2 provides a detailed overview from a programmer's perspective to the parsing and processing of XML documents on the Java™ Platform and to the key concepts involved. The focus will be on implementation rather than on the history of XML as a language. This is an introductory level session."

- Basic knowledge of Java™ technology is assumed.
- Familiarity with the basic XML concepts will be helpful but is not essential.

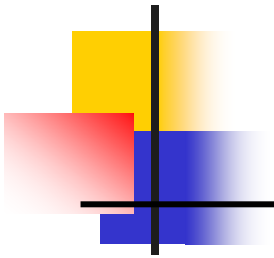


Contacting Me

- Feel free to contact me for additional information or ask follow up questions.

- **e-mail:**
  neilg@ca.ibm.com

- **Phone : +1 (905) 413-3519**

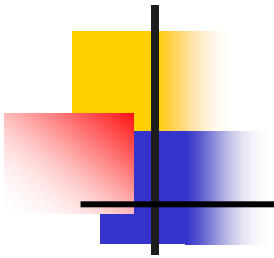


Coverage

- Clearly this is a (very) large topic
  - We'll cover as much as we can in the time.
    - In this session and the part 2 session.
  - If I don't cover an area you are interested in please feel free to talk to me after the session.
  - The samples from this talk will be available on the CSS CDROM

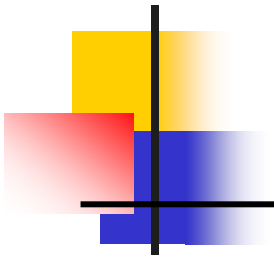
Sidenote - Java™ Technology & XML

- A good combination:
  - Java Technology
    - Portable code
  - XML
    - Portable data
    - The Java programming language is a good way to develop XML solutions but you do not *have* to use it to exploit XML. You can write code that uses XML in many other languages: C, C++, Perl even RPG and COBOL



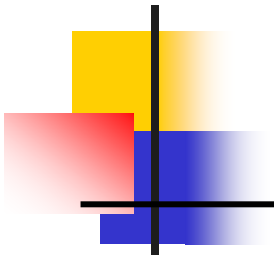
XML is a W3C Recommendation

- As we all know XML 1.0 is a key standard.
  - ƒ Recommendation from W3C
    - **http://www.w3.org/TR/1998/REC-xml-19980210**
    - **http://www.w3.org/TR/2000/REC-xml-20001006**
    - Can be downloaded as PDF or HTML
    - Very small documents (36 pages and 59 pages)
  - ƒ Rules for XML content and DTD's
  - ƒ Rules for parser writers
    - Not an API specification.
    - Doesn't define a parser invocation API.



XML 1.1 from the W3C

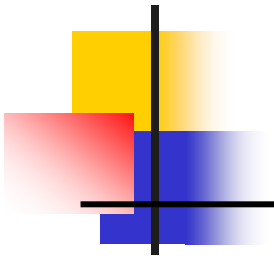
- XML 1.1 became a canidate recommendation Oct 15 2002.
  - **http://www.w3.org/TR/2002/CR-xml11-20021015/**
- It comes from the Blueberry requirements
  - **http://www.w3.org/TR/xml-blueberry-req**
    - Discriminating against languages simply because their scripts were not encoded in Unicode 2.0 is inherently unjust.
    - Unicode versions have marched forward.
    - XML 1.0 discriminates against older mainframe operating systems end-of-line methodologies.



Getting Started

- What tools do I need to get started ?

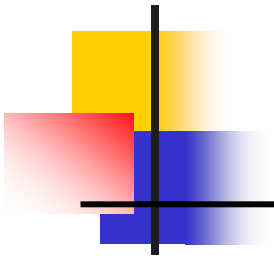
  - An XML Parser
    - Including a SAX and/or DOM implementation.
  - Programming language/environment.
    - Obviously!
  - API documentation.
    - DOM, SAX, Parser control.
  - A few good but simple examples.



Understanding the Basics of XML

- XML is a class of data objects (information) packaged in a specified format.

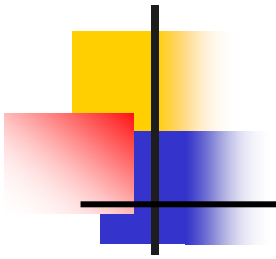
- XML Document
  - Data Objects that are well-formed, as described in the XML specification, form an XML Document
  - Being a valid XML Document is different than being a well-formed document.
    - Valid documents have additional constraints in the Document Type Definitions (DTD) or Schema associated with them



Understanding the Basics of XML (Continued)

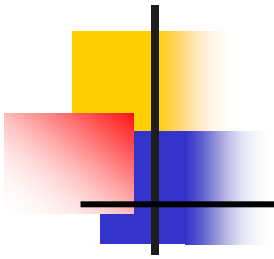
- An XML Document is made up of storage units called XML Entities.

- XML Entities
  - An Entity has content and a name
  - An XML Document has one Document Entity
    - This is also called the Root Entity or Root Element
    - The root entity serves as a starting place for the XML Processor



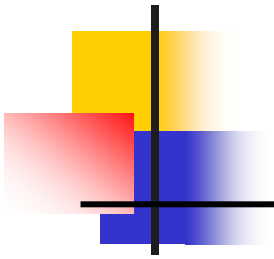
Parser *vs.* Processor

- Parser
  - Code you don't write (in most cases)
  - Understands the rules of XML syntax.
  - Does not understand the semantics of your specific grammar.
- Processor
  - The code you do write
  - Understands the semantics of your grammar.
  - May be as little as 20% of the work.
  - May be as much as 80% of the work.



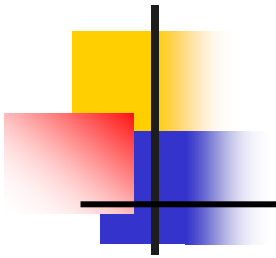
The Processing Steps

- Typical processing steps:
  - Parse the document (using XML parser)
    - Parser builds a DOM tree (DOM case)
    - Parser sends events (SAX case).
  - Process the document :
    - Using the DOM API, or
    - By handling SAX events.
  - Do something with the data.
    - Display it, interpret it and make decisions.
    - Produce a report, transcode it, *etc*...



Parsing the XML Document

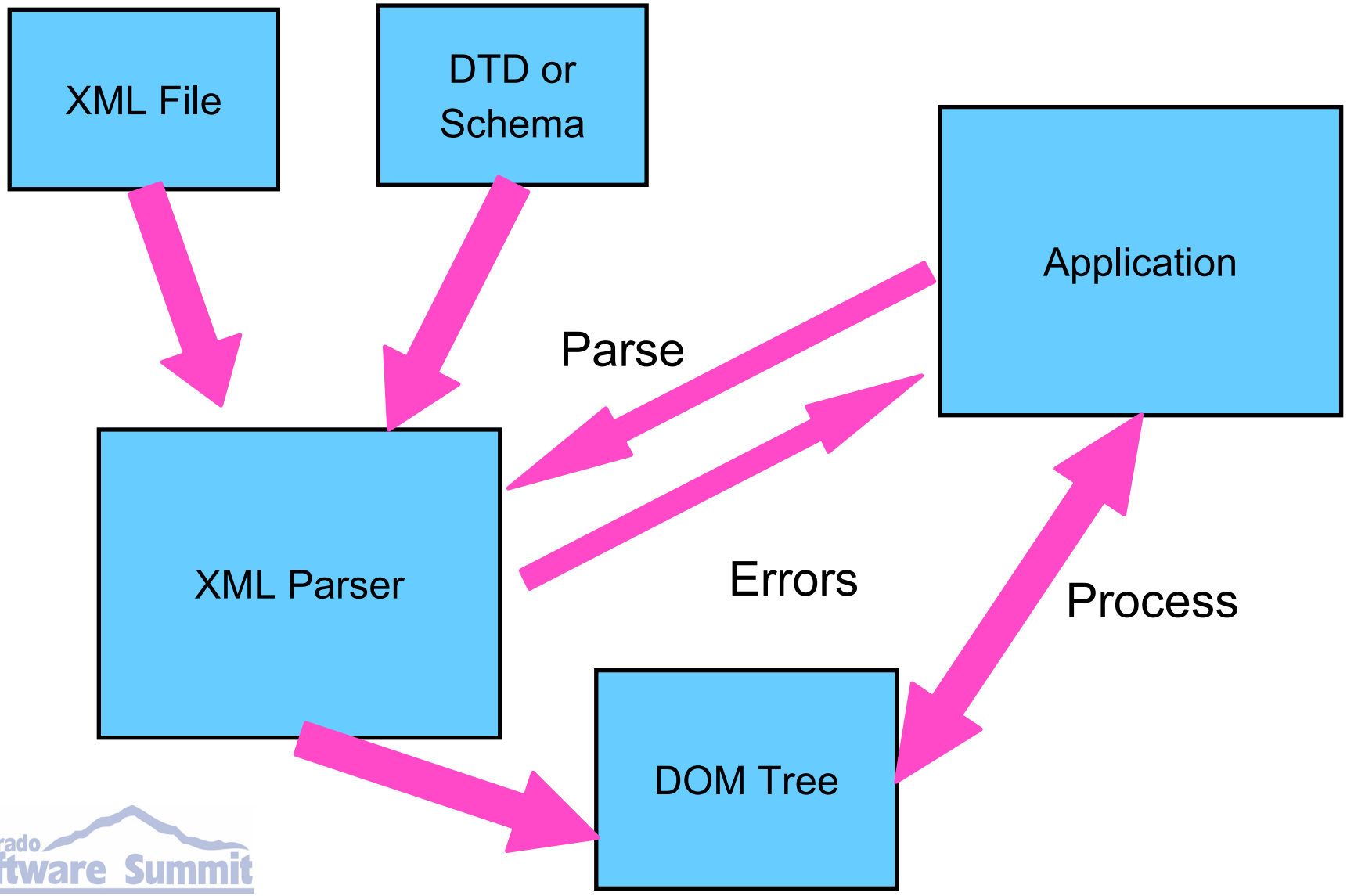
- Invoke the parser
  - Parser invocation/control API may vary.
  - Parser helps with
    - Validation
    - Well formedness checking
    - Building a document tree (DOM)
    - Notifying the application of errors.
  - Most parsers can handle files and streams.
  - Many parsers allow some/all of the error checking features to be disabled.



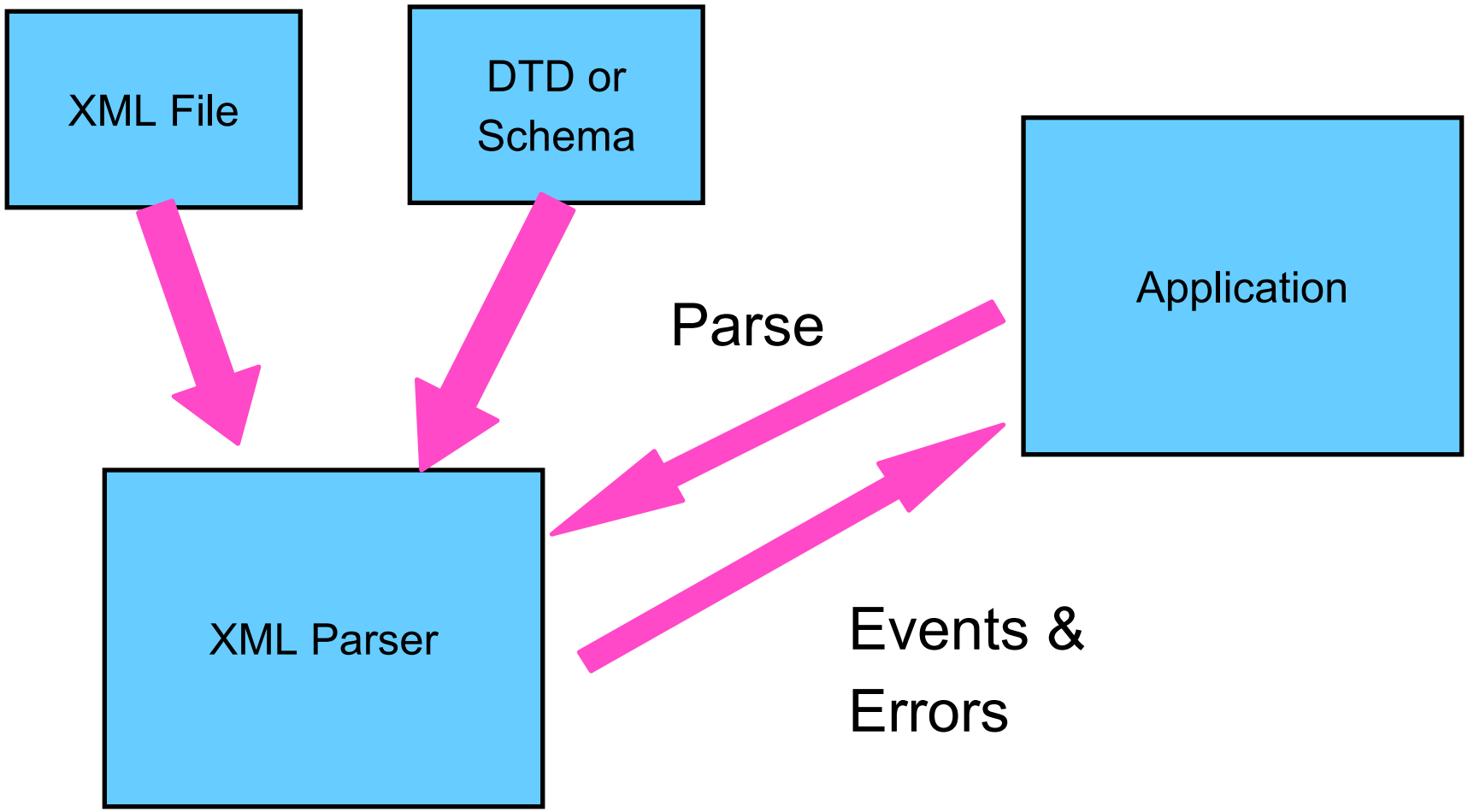
DOM *vs.* SAX

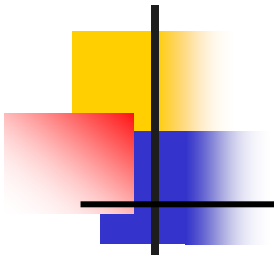
- DOM
  - Document Object Model (tree structure)
  - Highly functional (rich) API
  - Good if you intend to edit the document
  - Good if you plan to process multiple times.
    - Avoids re-parsing each time.
    - Avoids building your own internal tree (model).
- SAX
  - Event based model.
  - Good if you don't need to edit in memory.
  - Good if memory is limited.
  - Good if only parts of document are required.

XML Processing Model (DOM)



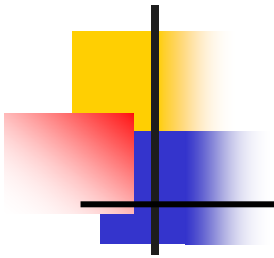
XML Processing Model (SAX)





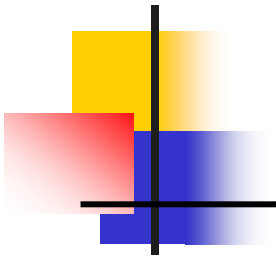
What Role Does the DTD Play?

- Allows for:
  - Sharing your grammar/data with others.
  - Validation by the parser.
  - Defaulting of values.
    - Can query if a value was defaulted or not.
    - Care is needed as DOM can become bloated by badly designed DTD's
- **NOTE:** If specified, the DTD is still read even if validation is off.



What Role Does the Schema Play?

- Allows for:
  - ƒ Sharing your grammar/data with others.
  - ƒ Validation by the parser.
    - More so that DTD, we'll see this coming up
  - ƒ Defaulting of values.
    - Can query if a value was defaulted or not.
    - Care is needed as DOM can become bloated by badly designed Schema's
- **NOTE:** If specified, the Schema is still read even if validation is off.

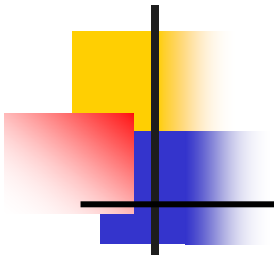


So What Does XML Look Like?

- A snippet of XML looks like this:

```
...
<speaker name="Mike" company="IBM">
  <speeches>
    <presentation>Intro to XML part 1</presentation>
    <presentation>Intro to XML part 2</presentation>
  </speeches>
</speaker>
...
```

Note: XML is case sensitive



What Does a DTD Look Like?

- A snippet of a DTD looks like this:

```
...
<!ELEMENT speaker (speeches)+>
<!ATTLIST speaker name CDATA #REQUIRED
                  company CDATA #IMPLIED >

<!ELEMENT speeches (presentation)+>

<!ELEMENT presentation (#PCDATA) >
...
```

Note: XML DTDs are case sensitive

What Does a Schema Look Like?

- A snippet of a Schema looks like this:

```
...
  <xsd:element name="speaker">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="speeches" minOccurs='1' maxOccurs='unbounded'/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" />
      <xsd:attribute name="company" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="speeches">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="presentation" minOccurs='1' maxOccurs='unbounded'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="presentation" type='xsd:string'/>
```

Note: XML DTD's are case sensitive

Important Parts of XML - *Elements*

- XML has Elements...
  - There are 4 elements on this page
    - 1 speaker, 1 speeches, 2 presentation

```
<speaker name="Mike" company="IBM">
 <speeches>
   <presentation>Intro to XML part 1</presentation>
   <presentation>Intro to XML part 2 </presentation>
 </speeches>
</speaker>
```

Important Parts of XML - *Attributes*

- Elements can have Attributes
  - ƒ There are 2 attributes on this page
    - 3 name, 1 company

```
<speaker name="Mike" company="IBM">
 <speeches>
        <presentation>Intro to XML part 1</presentation>
        <presentation>Intro to XML part 2 </presentation>
 </speeches>
</speaker>
```

Important Parts of XML - *Start Tags*

- XML has start-tags
  - ƒ There are 4 start-tags on this page

```
<speaker name="Mike" company="IBM">
    <speeches>
        <presentation>Intro to XML part 1</presentation>
        <presentation>Intro to XML part 2 </presentation>
    </speeches>
</speaker>
```

Important Parts of XML - *End Tags*

- XML has end-tags
  - ƒ There are 4 end-tags on this page

```
<speaker name="Mike" company="IBM">
    <speeches>
      <presentation>Intro to XML part 1</presentation>
        <presentation>Intro to XML part 2 </presentation>
    </speeches>
</speaker>
```

Important Parts of XML - *Children*

- Elements may have Children
  - The element speaker has 1 child element
    - 1 "speeches" element

```
<speaker name="Mike" company="IBM">
  <speeches>
      <presentation>Intro to XML part 1</presentation>
      <presentation>Intro to XML part 2 </presentation>
 </speeches>
</speaker>
```

Important Parts of XML - *Children* (Continued)

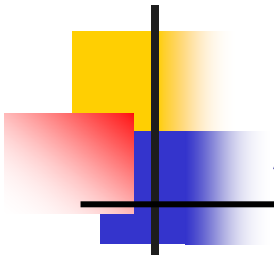
- Elements may have Children
  - The element speeches has 2 child elements
    - 2 "presentation" elements

```
<speaker name="Mike" company="IBM">
 <speeches>
        <presentation>Intro to XML part 1</presentation>
        <presentation>Intro to XML part 2 </presentation>
 </speeches>
</speaker>
```

Important Parts of XML - The XML Declaration

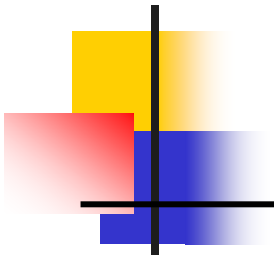
- XML documents should begin with an **XML declaration** which specifies the version of XML being used.
  - <?xml version="1.0"?>
  - This is optional, but recommended

```
<?xml version="1.0"?>
<speaker name="Mike" company="IBM">
    <speeches>
        <presentation>Intro to XML part 1</presentation>
        <presentation>Intro to XML part 2 </presentation>
    </speeches>
</speaker>
```



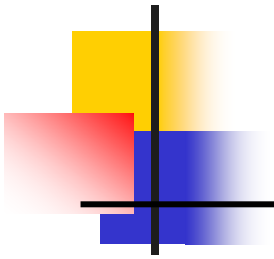
A Word about XML and File Size

- Many people have expressed concern over XML being highly verbose in comparison to a more customized, proprietary format.
  - These people overlook the repetitive nature which yields excellent compression
  - Lets take this presentation as an example:
    - In native PowerPoint format is 378KB. (Compressing that with Zip 193 KB)
    - In OpenOffice's native XML format (which is always compressed) is 203 KB (unzipped is 921 KB)



Introducing XMLView

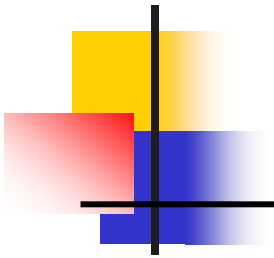
- XML Viewer/Editor
  - Written Java™ technology (uses Swing).
  - Uses the Xerces-J XML parser.
- Quick demo, let's look at
  - Parsing an XML file
  - DTD validation and defaulting
  - Well formedness checking
  - Example: XMLView & speeches.xml



The XML parser.

- You *could* write your own.
  - You could also re-invent the wheel !
- You'll probably use an existing one.
- Several are available for download.
  - Today we'll be using Xerces-J
    - Available from Apache (source and runtime).

    http://xml.apache.org

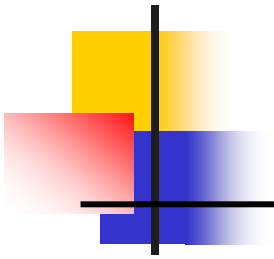


Introducing Xerces-J

- XML parser written using Java™ technology.
- Developed by Apache community.
  - Version 1.4.4 released (November 2001)
    - Use xerces.jar in the classpath
  - Version 2.5 currently available
    - Use xercesImpl.jar and xmlParserAPIs.jar in the classpath
- Supports:
  - DOM Level 1 and 2 (3 experimental support for 3)
  - SAX 1.0 and SAX 2.0
  - XML Schema

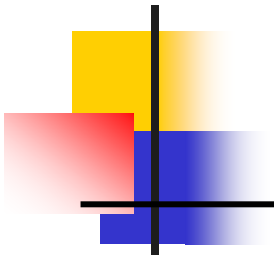
Colorado Software Summit

http://xml.apache.org



Invoking the Parser

- Parser can be invoked in several modes
  - DOM
  - SAX
  - Validating
  - Non-validating (well formedness only)
- W3C does not currently define a parser invocation API.
  - Xerces API is modeled on SAX API.
    - Uses XMLReader interface from SAX2.
    - Also supports JAXP 1.1



Processing the Document

- DOM
  - After parsing
    - Parser returns a "Document" to the application.
    - ***getDocumentElement()*** returns root node.
  - Application processes the DOM tree as needed using the DOM API to traverse tree.
- SAX
  - During parsing
    - SAX events are passed to the application.

Invoking the Parser (DOM Mode)

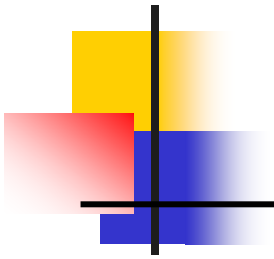
```
DOMParser parser = new DOMParser() ;
parser.setErrorHandler( new MyErrorHandler() ) ;

try
{
  parser.parse( fn ) ;
}
catch( Exception e )
{
  System.out.println( "Exception invoking parser") ;
}
Document doc = parser.getDocument() ;

if ( doc != null )
{
  processTheDocument( doc ) ;
}
```

Invoking the Parser (SAX 1 Mode)

```
SAXParser parser = new SAXParser() ;

parser.setDocumentHandler( this ); //deprecated (replaced by SAX2)

parser.setErrorHandler( new MyErrorHandler() );

try
{
  parser.parse( fn ) ;
}
catch( Exception e )
{
  System.out.println( "Exception invoking parser" ) ;
}
```

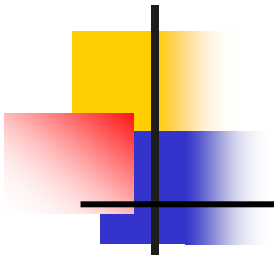
Invoking the Parser (SAX 2 Mode)

```
SAXParser parser = new SAXParser() ;

parser.setContentHandler( this );

parser.setErrorHandler( new MyErrorHandler() );

try
{
  parser.parse( fn ) ;
}
catch( Exception e )
{
  System.out.println( "Exception invoking parser" ) ;
}
```



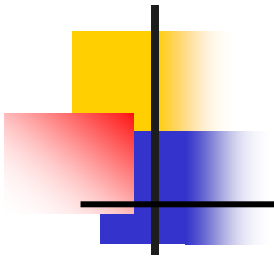
Invoking The Parser - Notes (1)

- Things to be aware of:
  - ƒ You can **not** use DOS file names when calling the ***parse()*** method.
    - You can use URL's or streams
    - It used to work (earlier versions of Xerces)
  - ƒ Valid file name (URL)
    - file:///d:\mypath\mydata.xml
  - ƒ Invalid file name
    - d:\mypath\mydata.xml



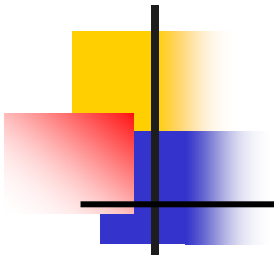
Invoking the Parser - Notes (2)

- You can also use Java streams rather than a URL to tell the parser what file to parse.
  - Create an InputStream object.
  - Note: If you use streams, you ***MUST*** also tell the parser where to look for the DTD by calling the ***setSystemID()*** method.



Streams and setSystemID

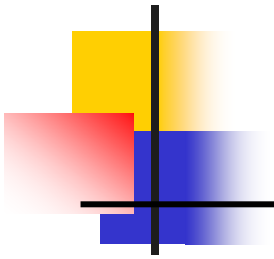
```
String fn = "file:///c:\\mypath\\myfile.xml" ;  //Note the URL

FileInputStream fis = null ;
File file = new File( fn ) ;

try
{
  fis = new FileInputStream( file ) ;
}
catch( FileNotFoundException fnfe )
{ ... }

InputSource is = new InputSource( fis ) ;

is.setSystemId( fn ) ;
```



Controlling Validation

- Turning validation on (Xerces)
  - Use the *setFeature* method

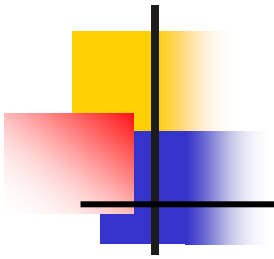
```
try
{
  String f = "http://xml.org/sax/features/validation" ;

  parser.setFeature( f , true);
}
catch (SAXException e)
{
  System.out.println("Error while enabling validation");
}
```



Key Packages

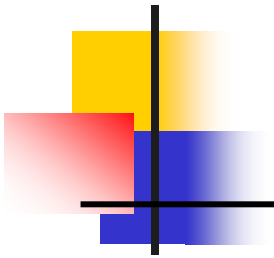
- Key packages included in Xerces-J

  - org.w3c.dom
    - DOM interfaces
  - org.xml.sax
    - SAX interfaces
  - org.apache.xerces.parsers
    - DOMParser
    - SAXParser



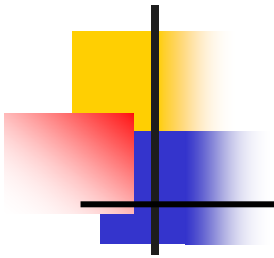
Key Classes/Interfaces

- org.apache.xerces.parsers.DOMParser
  - Class that produces a DOM tree from an XML input document.
- org.apache.xerces.parsers.SAXParser
  - Class that generates SAX 1 and SAX 2 events while parsing an XML document.
- org.w3c.dom.Node
  - Contains constants needed for processing a DOM tree.



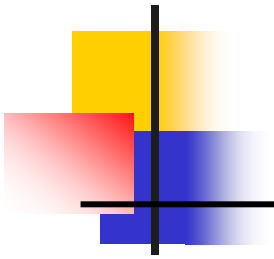
The DOM API

- DOM Level 1 (and 2) defines interfaces for the Java platform.
  - These are provided as bindings by Xerces (and other) parsers.
  - Can be called by the application.
    - To query the DOM.
    - To edit the DOM.
  - The DOM spec from W3C is at:
    - http://www.w3.org/TR/REC-DOM-Level-1/
    - DOM Level1 and 2 are W3C recommendations.
  - We will use the DOM API in our examples today.



Processing the DOM Tree

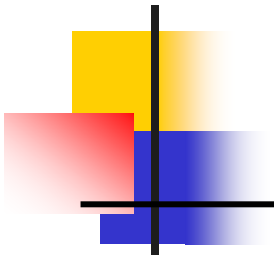
- Think of the DOM tree as a binary representation of the parsed XML file in tree form.
  - Usual tree processing methodologies apply.
- The DOM API is large and powerful (and therefore quite complex).
- DOM tree examples
  - soccer1.xml, soccer.dtd
  - soccer1.xml viewed by XMLView



Node Types

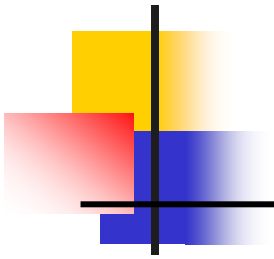
- Key part of processing is to recognize different DOM node types and their attributes (if any).
- Defined by W3C language bindings
  - Set of constant values.
    - Node.ELEMENT_NODE
    - Node.TEXT_NODE
    - Node.COMMENT_NODE,etc...
  - Documented in W3C DOM API

    http://w3.org/TR/REC-DOM-Level-1/



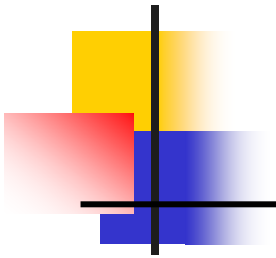
Walking the DOM Tree

- After parsing, you then "process" the DOM.
  - Recurse through the tree
  - Process nodes and their children
    - Handle any attributes present.
    - Process "text" nodes.



Processing Node Types

```
NodeList nl = parent.getChildNodes() ; //parent is of type Node
int count = nl.getLength() ;
Node thisNode ;

for ( int i = 0; i < count; i++ )
{
    thisNode = nl.item(i) ;

    int nodeType = thisNode.getNodeType() ;

    switch( nodeType )
    {
      case Node.ELEMENT_NODE:
      { ; }
      case Node.TEXT_NODE:
      { ; }
```



The SAX API

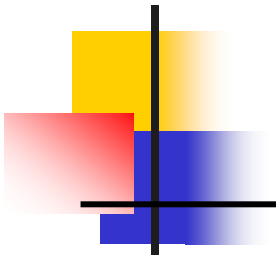
- Simple interface for event based parsing.
  - Xerces supports both SAX 1 and SAX 2.
- SAX events are similar to GUI events.
  - You handle the events.
  - If you don't store the data it is lost.
- For more details on SAX:
  - http://www.megginson.com/SAX/
  - See also xml-dev e-mail list

Handling SAX Events

(DocumentHandler interface examples from SAX 1)

```
public void startDocument()

public void endDocument()

public void startElement( String name
                        , AttributeList attrs )

public void endElement( String name )

public void characters(char[] ch, int start, int length)
```

NOTE: These interfaces are now deprecated (replaced by SAX2)



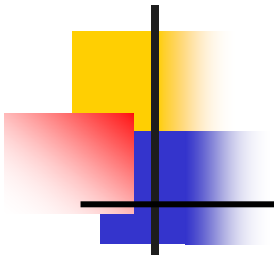
SAX 2.0

- Summary
  - SAX 1 was released in May 1998
  - SAX 2 was released on May 5th 2000
  - Fully supported in Xerces.
  - Now has additional support
    - Namespaces
    - Querying and setting parser features
    - Filter chains
    - Several APIs are deprecated
      - ✓See PxmlSAX2.java in samples package
  - Full details: http://www.saxproject.org/

Handling SAX Events

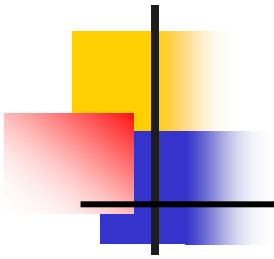
(ContentHandler interface examples from SAX 2)

```
public void startDocument()

public void endDocument()

public void startElement(String namespaceURI,
                         String localName,
                         String qName,
                         Attributes atts)

public void endElement(String namespaceURI,
                       String localName,
                       String qName )

public void characters(char[] ch,int start,int length)
```



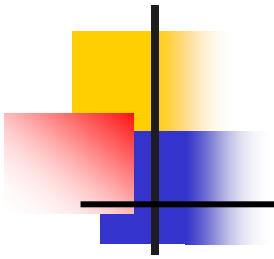
Handling XML Errors

- Examples of parse time errors
  - DTD not found
    - Reported *via* an exception.
  - Document fails validation
    - Does not match content model
    - Reported to ErrorHandler
  - Document is not well formed
    - Breaks the "golden rule" of XML.
    - Reported to error handler



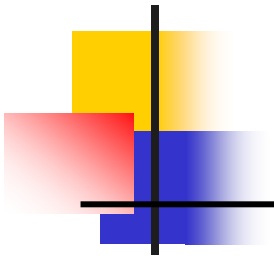
Examples of Errors – Notes 1

- Document is not well formed
  - Breaks the "golden rule" of XML
    - Invalid Sequence
      - <a>
    - Valid Sequence
      - <a>
      - Unlike HTML, sequence matters



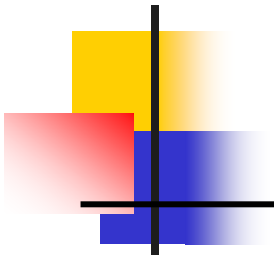
Examples of Errors – Notes 2

- Document is not well formed
  - Breaks the "golden rule" of XML
    - ƒ Invalid Element Type Match
      - <a></**B**>
    - ƒ Valid Sequence
      - <a>
      - XML is case sensitive



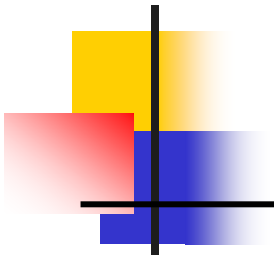
Examples of Errors – Notes 3

- Document is not well formed
  - Breaks the "golden rules" of XML
    - Invalid, not well formed
      - <a attribute= “hello />
    - Valid
      - <a attribute=“hello” />
      - Quotes must match up



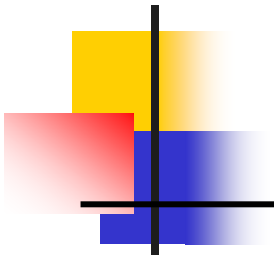
Examples of Errors – Notes 4

- Document is not well formed
  - Breaks the "golden rules" of XML
    - ƒ Invalid Unique Attribute
      - <a attribute= “hello” attribute= “hello” />
    - ƒ Valid
      - <a attribute=“hello” />
      - No attribute name may appear more than once in the same start-tag or empty-element tag



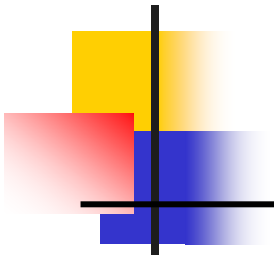
Examples of Errors – Notes 5

- Documents may also be invalid
  - The Document doesn't comply with "content model"
    - Fails validation (against DTD or Schema)



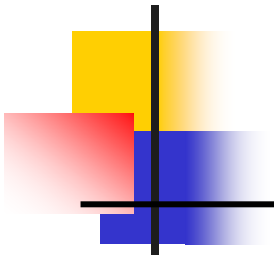
Error Categories

- Fatal
  - XML may not be well formed.
  - Application should assume the XML document is unusable.
  - Parser may abort (settable).
- Error
  - Validation errors (content model).
  - Parsing continues.
- Warning
  - Element declared but not used.
  - Parser continues.



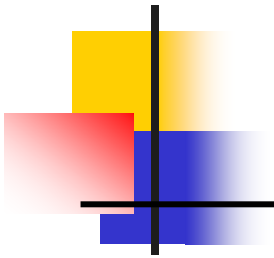
Error Categories (2)

- Where are they defined ?
  - XML Specification section 1.2
  - Check the XML spec for details of all possible causes of fatal errors, errors and warnings.



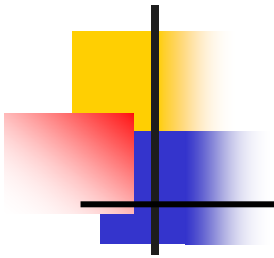
Writing an Error Handler

```
import org.xml.sax.ErrorHandler ;
import org.xml.sax.SAXParseException ;

public class MyErrorHandler implements ErrorHandler
{
  public void error( SAXParseException saxe )
  { ... }

  public void fatalError( SAXParseException saxe )
  { ... }

  public void warning( SAXParseException saxe )
  { ... }
}
```



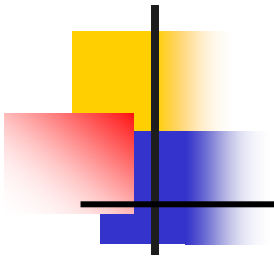
Registering the ErrorHandler

- You need to tell the parser where to send error information.
  - Do this by registering a handler.
    - ***setErrorHandler()***
  - If you don't you won't get notified
    - Even if validation is turned on.
  - Most commonly asked question on the Xerces-J e-mail list
    - "Why am I not getting notified of errors even though I know the XML file I am parsing has problems ?".



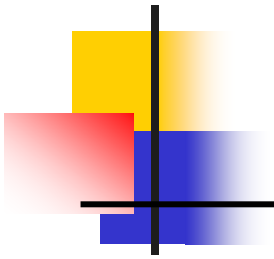
Parser Objects

- Can DOMParser and SAXParser objects be re-used ?
  - Yes
  - You can call the ***reset()*** method.
  - This allows the same object to parse different files and/or streams.



Re-using Parser Objects

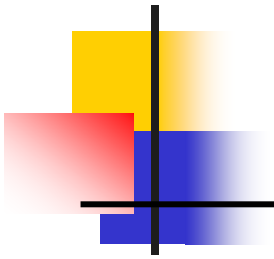
```
DOMParser parser ;
    ...
if ( parser == null )
{
  parser = new DOMParser() ;
}
else
{
  try
  {
    parser.reset() ;
  }
  catch( Exception e )
  { ... }
}
```



Whitespace

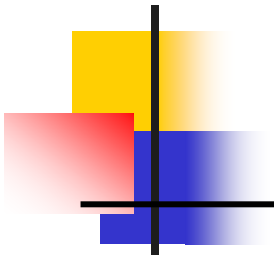
- From the XML spec (Second edition, page 16):

"An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content".



Some Notes about Whitespace

- End of line.
  - Parser does required conversions.
    - All <CR> <LF> get mapped to <LF> (0x0A)
    - Processing applications only sees <LF>
      - ✓Simplifies processing
    - Required by XML specification.



Whitespace Rules

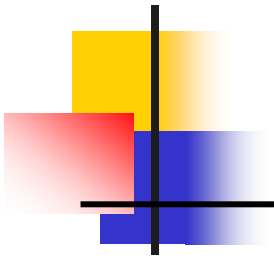
- General whitespace rules
  - ƒ Application needs to handle whitespace
    - Recognize xml:space attribute.
      - ✓This is for you, ***not*** the parser.
    - All whitespace is potentially preserved
    - This may yield "fake" DOM text elements.

Important Parts of XML - *Children Revisted*

- Elements may have Children
  - The element "speaker" has 1 ~~2~~ child elements
    - 1 "speeches" element
    - 1 White Space Text Node

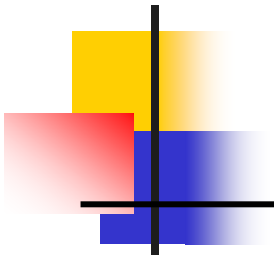
White space Child here

```
<speaker name="Mike" company="IBM">
 <speeches>
        <presentation name="Intro to XML part 1" />
        <presentation name="Intro to XML part 2" />
 </speeches>
</speaker>
```



Whitespace Example

- How should I handle whitespace in my code ?
  - *ƒ isIgnorableWhitespace()*
    - Special method provided by Xerces-J
    - Used to spot non important text nodes.
    - Only works if parser can deduce the content model (needs a DTD or Schema to be present).
    - NOTE: This is not a DOM API, it's a Xerces extension as part of the implementation of Text support (TextImpl class).
- Example: XMLView and whitespace handling.

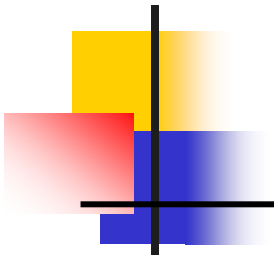


Xerces-J XML Parser

- Available from http://xml.apache.org

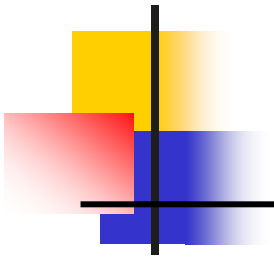
- Provides support for
  - XML 1.0
  - SAX 1.0 and SAX 2.0
  - DOM L1 and DOM L2 (experimental L3 support)
  - XML Schema

- Includes very good HTML material
  - Example: The Xerces documentation



XML Documentation Review

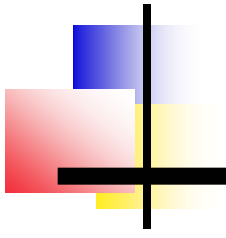
- The following useful HTML documentation is included with the Xerces package.
  - The Xerces invocation/control API.
  - The DOM API specification.
  - The SAX API specification.
  - http://xml.apache.org/xerces-j/api.html



Topics to be Covered in Part 2

- Building on the concepts from Part I.
  - Extra detail and examples
- Advanced topics and additional detail.
- Lots of code examples (as time permits).
  - Walkthrough of DOM processing code.
  - Walkthrough of SAX processing code.

A Detailed Introduction to Parsing and Processing XML Documents with Java™ Technology - Part 1



That's the end of the session!

Please feel free to ask more questions as time allows.

paolini@us.ibm.com

Colorado Software Summit