More about XML

What is XML Infoset ?

XML Information Set called xml **infoset** is the data or object model for XML document. This data model is an abstract set of information which forms the essence of an XML document. Basically, the XML **Infoset is a tree-based hierarchical representation of an XML document.** The xml infoset contains a document root, element, attributes, cdata,comment,namespace etc.

What is difference between CDATA and PCDATA ?

The CDATA section of an xml document is the text information which should not be parsed by XML parsers. Characters like "<,>,&" are illegal in XML elements. Some text, like JavaScript code, contains a lot of "<" or "&" characters. To avoid errors script code can be defined as CDATA. Basically,Everything inside a CDATA section is ignored by the parser.
A CDATA section starts with "<![CDATA[" and ends with "]]>":

<![CDATA[

]]>

```
<script>
<![CDATA[

     function doIt(a,b)
     {
          if (a < b && a < 0) then
            {
                return 1;
            }
          else
            {
                return 0;
            }
     }

]]>
</script>
```

What is DTD ?

If you want want to define the structure of the xml document by providing certain set of rules and regulations and to validate an xml against that predefined rules and there by to create a valid xml document, you can create a DTD called Document Type Definition. DTD encompasses the building blocks of an XML document. Basically DTD helps in the organisation of XML infoset. In a commonly used terms, I can say DTD specifies the order and position and structure of the elements in an XML document. DTD defines the structure of the XML document making compatible with SGML specification. It was developed and considered as the legacy XML rule modeller and to retain the structure of SGML.
There are two types of DTDs one is internal dtd and another is external dtd.
Internal DTD can be included in the target xml file and sometimes it is called inline DTD.
External DTD is placed outside the target xml file. It can be used by other xml file.

In a XML document, DTD is declared in a DOCTYPE declaration, which is beneath the XML declaration or prologue. External DTD is useful when you use XML protocol to communicate between separate systems since it reduces the overhead of resending the DTD each time as with inline definition. The external DTD could be placed in a place like a web server that could be accessed by both the systems.

What are the advantages of DTD ?

The main advantage of the DTD is that **it provides a validating parsers. It means the XML document itself is valid, because XML first used DTDs to validate documents,** they're the de facto standard protocol for describing XML documents. **All XML parsers do not support DTD validations. If do not bother about the data types, you can go for DTD.**

What is the difference between DTD and XML Schema ?

- ➔ Everything that can be defined by the DTD can also be defined by schema, but not vice versa.
- ➔ **DTD can have only two types of data, the CDATA and the PCDATA.** But in a schema you can use all the primitive data type that we use in the programming language and you have the flexibility of defining your own custom data types.
- ➔ **XML Schema, is namespace aware whereas DTD is not.**
- ➔ The most significant difference is XML schema use and XML based syntax whereas DTD syntax is completely different.
- ➔ **DTDs do not provide for enumerations** in an element's text content where as it is there in schema.
- ➔ **XML schema supports regular expressions** to set constraints on character data, which is not possible if you use a DTD.
- ➔ XSD supports references to external other XML schema where as DTD does not refer to other DTDs.
- ➔ XSD supports default values. You can specify default values of the elements. It is not the case in case of DTD.

What is an XML Schema ?

If you want to create a valid xml document by defining the certain rules and regulations, you can go for XML Schema. XSD elements can be of type **simpleType**, **complexType**, or **anyType**. An element of type simpleType contains only text. It cannot have attributes and elements. An element of type complexType can contain text, elements, and attributes. Elements can be local or global. An elements declared as the direct child of an XML schema is called a global element and can be used throughout the schema. Elements declared within a complex type definition are local elements. An element can have constraints such as minOccurs and maxOccurs. If the value of the maxOccurs constraint of an element is greater than one, then the element is stored as an array.

Sample XML,DTD and XSD

Let us look at the following XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation>
     <Name>DDSOFT INC</Name>
     <Employee Id="01">
          <Name>Sambit</Name>
          <Age>23</Age>
          <Address Type="Current">
               <Location>Bangalore</Location>
          </Address>
          <Address Type="Permanent">
               <Location>Delhi</Location>
          </Address>
     </Employee>
     <Employee id="02">
          <Name>Sambit</Name>
     </Employee>
</Organisation>
```

The dtd for the above xml file is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Organisation ((Name, Employee+))>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT Employee ((Name, (Age, Address*)?))>
<!ATTLIST Employee Id CDATA  #REQUIRED >
<!ELEMENT Age (#PCDATA)>
<!ELEMENT Address ((Location))>
<!ATTLIST Address Type (Permanent | Current) #REQUIRED >
```

The statement "<!ELEMENT Organisation ((Name, Employee+))>" indicates that employee tag should be repeated and there will be at least one
employee tag.

The statement <!ELEMENT Employee ((Name, (Age, Address*)?))> indicates that Age and Address tags are optional and the appearance of the tag Address is 0 or more.

The statement **<!ATTLIST Address Type (Permanent | Current) #REQUIRED > indicates that the attribute** can contain either (Permanent or Current). It can not
have different values other than Permanent or Current.

* means 0 or more
+ means 1 or more

Let us look at the following XML document for which we will create an XML Schema(XSD).

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation>
     <Name>DDSOFT INC</Name>
     <Employee Id="01">
          <Name>Sambit</Name>
          <Age>23</Age>
          <Country city="Bangalore">India</Country>
          <Address Type="Current">
               <Location>Bangalore</Location>
          </Address>
          <Address Type="Permanent">
               <Location>Delhi</Location>
          </Address>
     </Employee>
     <Employee id="02">
          <Name>Sambit</Name>
     </Employee>
</Organisation>
```

The following is the XML Schema .

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

<xs:element name="Organisation">
   <xs:complexType>
          <xs:sequence>
     <xs:element name="Name" type="xs:string"/>
     <xs:element       name="Employee"       type="EmployeeType"       minOccurs="1"
maxOccurs="unbounded"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:complexType name="EmployeeType">
      <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="Age" type="xs:string" minOccurs="1"   maxOccurs="1"/>
        <xs:element name="Country" type="CountryType" minOccurs="0" maxOccurs="1" />
        <xs:element       name="Address"       type="AddressType"       minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Id" use="required"  type="xs:string"/>
</xs:complexType>

<xs:complexType name="CountryType">
     <xs:attribute name="City" use="required" />
</xs:complexType>

<xs:complexType name="AddressType">
     <xs:sequence>
          <xs:element name="Location" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="Type" use="required">
     <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:enumeration value="Current"/>
```

```
            <xs:enumeration value="Permanent"/>
             </xs:restriction>
      </xs:simpleType>
        </xs:attribute>
</xs:complexType>
</xs:schema>
```

XML Parsing

There are basically four ways to parse an XML document.

1. **Parsing using DOM**
2. **Parsing using SAX**
3. **Parsing using Xpath**
4. **Parsing using Stax**

Let us consider the following sample XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation>
      <Name>DDSOFT INC</Name>
      <Employee Id="01" Type="Contract">
            <Name>Leena</Name>
            <Age>23</Age>
            <Country city="Bangalore">India</Country>
            <Address Type="Current">
                  <Location>Bangalore</Location>
            </Address>
            <Address Type="Permanent">
                  <Location>Delhi</Location>
            </Address>
      </Employee>
      <Employee Id="02" Type="Permanent">
            <Name>John</Name>
      </Employee>
</Organisation>
```

1. **Parsing using DOM parser.**

```
package com.ddlab.rnd.xml.parsing.dom;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class MyDOMParser
{
      private static void recursiveParsing( Element element )
      {
            NodeList nodesList = element.getChildNodes();
            for( int i = 0 ; i < nodesList.getLength() ; i++ )
            {
                Node node = nodesList.item(i);
                if( node.getNodeType() == Node.ELEMENT_NODE )
                {
                    element = (Element) node;
                    System.out.println("Element Name :::"+element.getNodeName());
                    Node eleNode = element.getFirstChild();
                    System.out.println(element.getNodeName()+"-----
>"+eleNode.getNodeValue());

                    if( node.hasAttributes() )
                    {
                        NamedNodeMap map = node.getAttributes();
                        int len = map.getLength();
                        System.out.println("------------------- List of attributes
----------------");
                        for( int j = 0 ; j < len ; j++ )
                        {
                            Node theAttribute = map.item(j);
                            System.out.println(theAttribute.getNodeName() +
"==========" + theAttribute.getNodeValue());
                        }
                        System.out.println("------------------- List of attributes
----------------");
                    }
                    recursiveParsing(element);
                }
            }
        }

        public static void main(String[] args)
        {
            File fXmlFile = new File("xmlfiles/org.xml");
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            try
            {
                DocumentBuilder  dBuilder = dbFactory.newDocumentBuilder();
                Document doc = dBuilder.parse(fXmlFile);

                NodeList nodesList = doc.getChildNodes();
                System.out.println(nodesList.getLength());

                for( int i = 0 ; i < nodesList.getLength() ; i++ ) {
                    Node node = nodesList.item(i);
                    if( node.getNodeType() == Node.ELEMENT_NODE ) {
                        Element element = (Element) node;
                        System.out.println("Element Name
:::"+element.getNodeName());
                        recursiveParsing(element);
                    }
                }
            }
            catch (ParserConfigurationException e)
```

```
		{
			e.printStackTrace();
		}
		catch (SAXException e)
		{
			e.printStackTrace();
		}
		catch (IOException e)
		{
			e.printStackTrace();
		}

	}
}
```

2. **Parsing using SAX Parser**

```
package com.ddlab.rnd.xml.parsing.sax;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class MySAXParser
{
	private class MessageInfoHandler extends DefaultHandler
	{
		private String data = null;
		@Override
		public void characters(char[] ch, int start, int length)
				throws SAXException
		{
				data = new String(ch,start,length);
				System.out.println("Data :::"+data);
		}

		@Override
		public void startElement(String uri, String localName, String qName,
				Attributes attributes) throws SAXException
		{
			System.out.println("QName :::"+qName);
			int attribLen = attributes.getLength();
			for( int i = 0 ; i < attribLen ; i++ )
			{
				System.out.println("Attribute Name
:::"+attributes.getQName(i));
				System.out.println("Attribute Value
:::"+attributes.getValue(i));
			}
```

```
            }



            @Override
            public void endElement(String uri, String localName, String qName)
                        throws SAXException
            {
//                  System.out.println("QName in End Element:::"+qName);
            }

      }

      public void doSAXParsing( ) throws Exception
      {
            File fXmlFile = new File("xmlfiles/org.xml");
            try
            {
                  SAXParserFactory saxFactory = SAXParserFactory.newInstance();
                  SAXParser saxParser = saxFactory.newSAXParser();
                  MessageInfoHandler msgInfoHandler = new MessageInfoHandler();
                  saxParser.parse(fXmlFile, msgInfoHandler);
            }
            catch (ParserConfigurationException e)
            {
                  e.printStackTrace();
            }
            catch (SAXException e)
            {
                  e.printStackTrace();
            }
            catch (IOException e)
            {
                  e.printStackTrace();
            }
            catch( Exception e )
            {
                  e.printStackTrace();
            }
      }

      public static void main(String[] args) throws Exception
      {
            new MySAXParser().doSAXParsing();
      }
}
```

3. **Parsing using Xpath API**

```
package com.ddlab.rnd.xml.parsing.xpath;
import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Document;

public class MyXPath
{

     public static void main(String[] args) throws Exception
     {
//          String expression = "Organisation/Employee[1]/Name";//To get the
information for the Name tag
//          String expression = "Organisation/Employee[1]/Age";//To get the
information for the Age tag
//          String expression = "Organisation/Employee[1]/Country";//To get the
information for the Country tag
            String expression = "Organisation/Employee[1]/Country/@city";//To get the
information for the Country city attribute tag
//          String expression = "//Employee";
            File xmlFile = new File("xmlfiles/org.xml");
            Document xmlDocument =
DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(xmlFile);
            XPathFactory xPathFactory = XPathFactory.newInstance();
            XPath xPath = xPathFactory.newXPath();
            XPathExpression xPathExpression = xPath.compile(expression);
            Object result = xPathExpression.evaluate(xmlDocument);//To get a single
value
//          Object result = xPathExpression.evaluate(xmlDocument,
XPathConstants.NODESET);//To get a list of results
            String resultStr = (String)result;
            System.out.println("Result :::"+resultStr);
     }

}
```

4. Parsing using Stax Parser(Cursor API)XMLStreamReader

```
package com.ddlab.rnd.xml.parsing.stax;
import java.io.File;
import java.io.FileReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamReader;

public class MyStaxCursorParser
{
	public static void main(String[] args) throws Exception
	{
		FileReader filereader = new FileReader( new File("xmlfiles/org.xml") );
		XMLInputFactory xmlInputFact = XMLInputFactory.newInstance();
		XMLStreamReader xmlStreamReader = 
xmlInputFact.createXMLStreamReader(filereader);

		while( xmlStreamReader.hasNext() )
		{
			int eventType = xmlStreamReader.next();
//			System.out.println("Event Type :::"+eventType);
			if( eventType == XMLStreamReader.START_ELEMENT )
			{
				System.out.println("Element Name 
:::"+xmlStreamReader.getName().toString());
				int totalAttribs = xmlStreamReader.getAttributeCount();
				if( totalAttribs != 0 )
				{
					System.out.println("----------------------------------------
-");
					for( int i = 0 ; i < totalAttribs ; i++ )
					{
						System.out.println("Attribute Name 
:::"+xmlStreamReader.getAttributeName(i));
						System.out.println("Attribute Value 
:::"+xmlStreamReader.getAttributeValue(i));
					}
				}
			}
			else if( eventType == XMLStreamReader.CHARACTERS )
			{
				System.out.println("Element Text 
:::"+xmlStreamReader.getText());
			}
			else if( eventType == XMLStreamReader.END_ELEMENT )
			{
//				System.out.println("Element Name 
:::"+xmlStreamReader.getName().toString()+"<-----
>"+xmlStreamReader.getElementText());
			}
		}
	}
}
```

5. **Parsing using Stax Parser(Event Iterator API)** **XMLEventReader**

```
package com.ddlab.rnd.xml.parsing.stax;
import java.io.File;
import java.io.FileReader;
import java.util.Iterator;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.XMLEvent;

public class MyStaxEventParser
{
	public static void main(String[] args) throws Exception
	{
		FileReader filereader = new FileReader( new File("xmlfiles/org.xml") );
		XMLInputFactory xmlInputFact = XMLInputFactory.newInstance();
		XMLEventReader xmlEventReader = 
xmlInputFact.createXMLEventReader(filereader);

		while( xmlEventReader.hasNext() )
		{
			XMLEvent xmlEvent = xmlEventReader.nextEvent();
			if( xmlEvent.isStartElement() )
			{
				System.out.println("Element Name 
:::"+xmlEvent.asStartElement().getName().toString());
				Iterator attrItr = xmlEvent.asStartElement().getAttributes();
				while( attrItr.hasNext() )
				{
					Object obj = attrItr.next();
					Attribute attribs = (Attribute)obj;
					System.out.println("Attribute Name 
:::"+attribs.getName());
					System.out.println("Attribute Value 
:::"+attribs.getValue());
				}
			}
			else if( xmlEvent.isCharacters() )
			{
				System.out.println("Element Text 
:::"+xmlEvent.asCharacters().getData());
			}
			else if( xmlEvent.isEndElement() )
			{
				// System.out.println("Element Name 
:::"+xmlStreamReader.getName().toString()+"<-----
>"+xmlStreamReader.getElementText());
			}
		}
	}

}
```

XML validations with DTD and XSD

Inline XML dtd validation

The sample XML file is given below. This xml file contains an inline DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Organisation [
<!ELEMENT Organisation ((Name, Employee+))>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT Employee ((Name, (Age, Address*)?))>
<!ATTLIST Employee Id CDATA  #REQUIRED >
<!ELEMENT Age (#PCDATA)>
<!ELEMENT Address ((Location))>
<!ATTLIST Address Type (Permanent | Current) #REQUIRED >
]>
<Organisation>
      <Name>Test</Name>
      <Employee Id="01">
            <Name/>
      </Employee>
</Organisation>
```

If the dtd is attached to an XML file, whether it is internal or external dtd, the XML document itself is valid.

XML validation with DTD using DOM

```
package com.ddlab.rnd.xml.validation.dom;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

class SimpleErrorHandler implements ErrorHandler {
    public void warning(SAXParseException e) throws SAXException {
//          e.printStackTrace();
        System.out.println("SAXParseException warning...\n"+e.getMessage());
        throw e;
    }

    public void error(SAXParseException e) throws SAXException {
//          e.printStackTrace();
        System.out.println("SAXParseException error...\n"+e.getMessage());
        throw e;
    }

    public void fatalError(SAXParseException e) throws SAXException {
//          e.printStackTrace();
      System.out.println("SAXParseException fatalError...\n"+e.getMessage());
      throw e;
    }
}
```

```
public class MyDOMDtdValidator
{
      public static void main(String[] args)
      {

            try
            {
                  String xmlFilePath = "xmlfiles/org-inlinedtd.xml";//Correct XML file
//                String xmlFilePath = "xmlfiles/bad-org-inlinedtd.xml";//Incorrect XML
file
                  DocumentBuilderFactory factory = 
DocumentBuilderFactory.newInstance();
                  /*
                   * The following line is important to validate a document
                   */
                  factory.setValidating(true);
//                factory.setNamespaceAware(true);
                  DocumentBuilder builder = factory.newDocumentBuilder();
                  builder.setErrorHandler( new SimpleErrorHandler());
                  builder.parse(new InputSource(xmlFilePath));
                  System.out.println("This is a valid document");
            }
            catch( SAXParseException spe )
            {
                  spe.printStackTrace();
            }
            catch (ParserConfigurationException e)
            {
                  e.printStackTrace();
            }
            catch (SAXException e)
            {
                  e.printStackTrace();
            }
            catch (IOException e)
            {
                  e.printStackTrace();
            }
      }
}
```

XML validation with DTD using SAX

```
package com.ddlab.rnd.xml.validation.sax;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

class MyHandler extends DefaultHandler
{
     private String data = null;
     @Override
     public void characters(char[] ch, int start, int length)
                throws SAXException
     {
                data = new String(ch,start,length);
                System.out.println("Data :::"+data);
     }

     @Override
     public void startElement(String uri, String localName, String qName,
                Attributes attributes) throws SAXException
     {
          System.out.println("QName :::"+qName);
     }

     @Override
     public void endElement(String uri, String localName, String qName)
                throws SAXException
     {
          System.out.println("QName in End Element:::"+qName);
     }
}

class MyErrorHandler implements ErrorHandler
{
    public void warning(SAXParseException e) throws SAXException
    {
//        e.printStackTrace();
        System.out.println("SAXParseException warning...\n"+e.getMessage());
        throw e;
    }

    public void error(SAXParseException e) throws SAXException
    {
//        e.printStackTrace();
        System.out.println("SAXParseException error...\n"+e.getMessage());
        throw e;
    }

    public void fatalError(SAXParseException e) throws SAXException
    {
//          e.printStackTrace();
      System.out.println("SAXParseException fatalError...\n"+e.getMessage());
      throw e;
    }
}

public class MySAXDtdValidator
{
      public static void main(String[] args) throws Exception
      {
//          String xmlFilePath = "xmlfiles/org-inlinedtd.xml";//Correct XML file
            String xmlFilePath = "xmlfiles/bad-org-inlinedtd.xml";//Incorrect XML file
            SAXParserFactory saxFactory = SAXParserFactory.newInstance();
            /*
             * The following line is important to
             * validate an XMl file with inline dtd.
             */
            saxFactory.setValidating(true);
            saxFactory.setNamespaceAware(true);
            SAXParser saxParser = saxFactory.newSAXParser();
            MyHandler myHandler = new MyHandler();
            XMLReader reader = saxParser.getXMLReader();
            reader.setErrorHandler(new MyErrorHandler());
            reader.setContentHandler(myHandler);
            reader.parse(new InputSource(xmlFilePath));
      }
}
```

Sample DTD and XML

The following is a sample XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation>
      <Name>DDSOFT INC</Name>
      <Employee Id="01" Type="Contract">
            <Name>Leena</Name>
            <Age>23</Age>
            <Country city="Bangalore">India</Country>
            <Address Type="Current">
                  <Location>Bangalore</Location>
            </Address>
            <Address Type="Permanent">
                  <Location>Delhi</Location>
            </Address>
      </Employee>
      <Employee Id="02" Type="Permanent">
            <Name>John</Name>
      </Employee>
</Organisation>
```

For this above XML document, the DTD is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Organisation ((Name, Employee+))>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT Employee ((Name, (Age, Address*)?))>
<!ATTLIST Employee Id CDATA  #REQUIRED >
<!ELEMENT Age (#PCDATA)>
<!ELEMENT Address ((Location))>
<!ATTLIST Address Type (Permanent | Current) #REQUIRED >
```

XML validation with XML Schema(XSD) using DOM

The XML Schema(XSD) file will always be external to the target XML document. If an XML document has a schema associated with it, it is the Xml processor which can validate the XML document against that XML schema.

Let us look at the sample XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation>
      <Name>DDSOFT INC</Name>
      <Employee Id="01" Type="Contract">
            <Name>Leena</Name>
            <Age>23</Age>
            <Country city="Bangalore">India</Country>
            <Address Type="Current">
                  <Location>Bangalore</Location>
            </Address>
            <Address Type="Permanent">
                  <Location>Delhi</Location>
            </Address>
      </Employee>
      <Employee Id="02" Type="Permanent">
            <Name>John</Name>
      </Employee>
</Organisation>
```

So an XML document with associated XML schema is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Organisation
      xsi:noNamespaceSchemaLocation="file:///F:/javaworkspaces/redworkspace-6-
11/xmlproject/xsd/org.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Name>String</Name>
      <Employee Id="String">
            <Name>String</Name>
            <Age>String</Age>
            <Country City="text"/>
            <Address Type="Current">
                  <Location>String</Location>
            </Address>
      </Employee>
</Organisation>
```

For this above XML document, the XML schema is given below.

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

<xs:element name="Organisation">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Name" type="xs:string"/>
                  <xs:element name="Employee" type="EmployeeType" minOccurs="1"
maxOccurs="unbounded"   />
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:complexType name="EmployeeType">
      <xs:sequence>
        <xs:element name="Name" type="xs:string"     minOccurs="1"     maxOccurs="1"/>
        <xs:element name="Age" type="xs:string" minOccurs="1"    maxOccurs="1"/>
        <xs:element name="Country" type="CountryType" minOccurs="0" maxOccurs="1" />
        <xs:element name="Address" type="AddressType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Id" use="required"  type="xs:string"/>
</xs:complexType>

<xs:complexType name="CountryType">
      <xs:attribute name="City" use="required" />
</xs:complexType>

<xs:complexType name="AddressType">
      <xs:sequence>
            <xs:element name="Location" type="xs:string"/>
      </xs:sequence>
      <xs:attribute     name="Type" use="required">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Current"/>
                        <xs:enumeration value="Permanent"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
</xs:complexType>

</xs:schema>
```

XML validation with XSD using DOM

The java program for the above is given below.

```
package com.ddlab.rnd.xml.validation.dom;
import java.io.File;
import java.io.IOException;
import javax.xml.XMLConstants;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

public class MyDOMXsdValidator
{
	public static boolean isXMLValidWithSchema( String schemaFilePath , String
xmlFilePath )
	{
		boolean flag = false;
		try {
			SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
			Schema schema = factory.newSchema(new File(schemaFilePath));
			Validator validator = schema.newValidator();
			DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance();
			docFactory.setNamespaceAware(true);
			DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
			Document document = docBuilder.parse(new File(xmlFilePath));
			validator.validate(new DOMSource(document));
			flag = true;
		}
		catch( SAXException se )
		{
			se.printStackTrace();
		}
		catch( ParserConfigurationException pce )
		{
			pce.printStackTrace();
		}
		catch( IOException ie )
		{
			ie.printStackTrace();
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
		return flag ;
	}

	public static void main(String[] args)
	{
//		String xmlFilePath = "xmlfiles/org-xsd.xml";
		String xmlFilePath = "xmlfiles/bad-org-xsd.xml";
		String schemaFilePath = "xsd/org.xsd";

		System.out.println(isXMLValidWithSchema(schemaFilePath, xmlFilePath));
		System.out.println("Validation ok");
	}

}
```

XML validation with XSD using SAX

The java program is given below.

```
package com.ddlab.rnd.xml.validation.sax;
import java.io.File;
import java.io.IOException;
import javax.xml.XMLConstants;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.transform.sax.SAXSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.Attributes;
import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

class MySAXHandler extends DefaultHandler
{
	private String data = null;
	@Override
	public void characters(char[] ch, int start, int length)
			throws SAXException
	{
			data = new String(ch,start,length);
			System.out.println("Data :::"+data);
	}

	@Override
	public void startElement(String uri, String localName, String qName,
			Attributes attributes) throws SAXException
	{
		System.out.println("QName :::"+qName);
	}
```

```
    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException
    {
        System.out.println("QName in End Element:::"+qName);
    }
}

class MySAXErrorHandler implements ErrorHandler
{
    public void warning(SAXParseException e) throws SAXException
    {
//          e.printStackTrace();
        System.out.println("SAXParseException warning...\n"+e.getMessage());
        throw e;
    }

    public void error(SAXParseException e) throws SAXException
    {
//          e.printStackTrace();
        System.out.println("SAXParseException error...\n"+e.getMessage());
        throw e;
    }

    public void fatalError(SAXParseException e) throws SAXException
    {
//          e.printStackTrace();
     System.out.println("SAXParseException fatalError...\n"+e.getMessage());
     throw e;
    }
}

public class MySAXXsdValidator
{
    public static boolean isXMLValidWithSchema( String schemaFilePath , String 
xmlFilePath )
    {
        boolean flag = false;
        try {
            SchemaFactory factory = 
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
            Schema schema = factory.newSchema(new File(schemaFilePath));
            Validator validator = schema.newValidator();

            SAXParserFactory saxFactory = SAXParserFactory.newInstance();
            /*
             * The following line is important to
             * validate an XMl file with inline dtd.
             */
            validator.validate(new SAXSource( new InputSource(xmlFilePath)));
            flag = true;
```

```
                SAXParser saxParser = saxFactory.newSAXParser();
                MySAXHandler myHandler = new MySAXHandler();
                XMLReader reader = saxParser.getXMLReader();
                reader.setErrorHandler(new MySAXErrorHandler());
                reader.setContentHandler(myHandler);
                reader.parse(new InputSource(xmlFilePath));
           }
           catch( SAXException se )
           {
                se.printStackTrace();
           }
           catch( ParserConfigurationException pce )
           {
                pce.printStackTrace();
           }
           catch( IOException ie )
           {
                ie.printStackTrace();
           }
           catch (Exception e)
           {
                e.printStackTrace();
           }
           return flag ;
      }

      public static void main(String[] args)
      {
//         String xmlFilePath = "xmlfiles/org-xsd.xml";
           String xmlFilePath = "xmlfiles/bad-org-xsd.xml";
           String schemaFilePath = "xsd/org.xsd";

           System.out.println(isXMLValidWithSchema(schemaFilePath, xmlFilePath));
           System.out.println("Validation ok");
      }


}
```

XML validation with XSD using Stax

```
package com.ddlab.rnd.xml.validation.stax;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.XMLConstants;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.SAXException;

public class MyStaxXsdValidator
{
     private static XMLEventReader getXMLEventReader(String filename) {
         XMLInputFactory xmlif = null;
         XMLEventReader xmlr = null;
         try {
           xmlif = XMLInputFactory.newInstance();
           xmlif.setProperty(XMLInputFactory.IS_REPLACING_ENTITY_REFERENCES,
Boolean.TRUE);
           xmlif.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES,
Boolean.FALSE);
           xmlif.setProperty(XMLInputFactory.IS_NAMESPACE_AWARE, Boolean.TRUE);
           xmlif.setProperty(XMLInputFactory.IS_COALESCING, Boolean.TRUE);
           FileInputStream fis = new FileInputStream(filename);
           xmlr = xmlif.createXMLEventReader(filename, fis);
         } catch (Exception ex) {
           ex.printStackTrace();
         }
         return xmlr;
       }

     public static boolean isXMLValidWithSchema( String schemaFilePath , String
xmlFilePath )
     {
          boolean flag = false;
          try {
                SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
                Schema schema = factory.newSchema(new File(schemaFilePath));
                Validator validator = schema.newValidator();
                DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance();
                docFactory.setNamespaceAware(true);
                javax.xml.transform.Result xmlResult = new
javax.xml.transform.stax.StAXResult(
                          XMLOutputFactory.newInstance().createXMLStreamWriter(new
FileWriter(schemaFilePath)));
                      javax.xml.transform.Source xmlSource = new
javax.xml.transform.stax.StAXSource(
                          getXMLEventReader(xmlFilePath));
                      validator.validate(xmlSource, xmlResult);
                flag = true;
           }
           catch( SAXException se )
           {
                se.printStackTrace();
           }
           catch( IOException ie )
           {
                ie.printStackTrace();
           }
           catch (Exception e)
           {
                e.printStackTrace();
           }
           return flag ;
      }

      public static void main(String[] args)
      {
//         String xmlFilePath = "xmlfiles/org-xsd.xml";
           String xmlFilePath = "xmlfiles/bad-org-xsd.xml";
           String schemaFilePath = "xsd/org.xsd";

           System.out.println(isXMLValidWithSchema(schemaFilePath, xmlFilePath));
           System.out.println("Validation ok");
      }

}
```

We can not validate an XML document with DTD using Stax.

What is XML processing pipelines ?

An XML pipeline is a way to express the various steps in processing XML documents. It encompasses the **XML transformations** and validations. For Example for ETL operations.

What are the benefits of Xpath ?

XPath is designed for XML documents. It provides a single syntax that you can use for queries, addressing, and patterns.
XPath is concise, simple, and powerful. XPath has many benefits, as follows:

Queries are compact.
Queries are easy to type and read.
Syntax is simple for the simple and common cases.
Query strings are easily embedded in programs, scripts, and XML or HTML attributes.
Queries are easily parsed.

You can specify any path that can occur in an XML document and any set of conditions for the nodes in the path.
You can uniquely identify any node in an XML document.
Queries return any number of results, including zero.
Query conditions can be evaluated at any level of a document and are not expected to navigate from the top node of a document.
Queries do not return repeated nodes.
For programmers, queries are declarative, not procedural. They say what should be found, not how it should be found.
This is important because a query optimizer must be free to use indexes or other structures to find results efficiently.
XPath is designed to be used in many contexts. It is applicable to providing links to nodes, for searching repositories, and for many other applications.

Choosing between Cursor and Iterator APIs - When to use Cursor API and Iterator API ?

It is reasonable to ask at this point,
"What API should I choose? Should I create instances of XMLStreamReader or XMLEventReader?
Why are there two kinds of APIs anyway?"

If you are programming for a particularly memory-constrained environment, like Java ME, you can make smaller, more efficient code with the cursor API.

If performance is your highest priority—for example, when creating low-level libraries or infrastructure—the cursor API is more efficient.
If you want to create XML processing pipelines, use the iterator API.
If you want to modify the event stream, use the iterator API.
If you want your application to be able to handle pluggable processing of the event stream, use the iterator API.
In general, if you do not have a strong preference one way or the other, using the iterator API is recommended because it is more flexible and extensible, thereby "future-proofing" your applications.

Memory Constrained, Performance : Cursor
Pipeline Processing : Iterator

XML generation

We can generate an XML document using DOM,SAX,STAX and JAXB. We can update an XML document using DOM and JAXB.

XML generation using DOM

```
package com.ddlab.rnd.xml.dom;
import java.io.StringWriter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

public class DomXmlCreate
{
      public void generateXml( Document doc ) throws Exception
      {
            TransformerFactory transfactory = TransformerFactory.newInstance();
        Transformer trans = transfactory.newTransformer();
        trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");//Mention yes if you
do not want xml prologue
        trans.setOutputProperty(OutputKeys.INDENT, "yes");//Mention no if you want xml
document in a single line

        //create string from xml tree
        StringWriter sw = new StringWriter();
        StreamResult result = new StreamResult(sw);
        DOMSource source = new DOMSource(doc);
        trans.transform(source, result);
        String xmlString = sw.toString();

        //print xml
        System.out.println("Here's the xml:\n\n" + xmlString);
      }

      public static void main(String[] args)
      {
            try
            {
                  DocumentBuilderFactory docBuildFact = DocumentBuilderFactory.newInstance();
                  DocumentBuilder docBuilder = docBuildFact.newDocumentBuilder();
            Document doc = docBuilder.newDocument();
            //Creation of Root Element for Organisation
            Element orgRoot = doc.createElement("Organisation");
            orgRoot.setAttribute("type", "Reasearch and Development");//Setting the attribute
            orgRoot.setAttribute("name", "DDLAB INC");
            doc.appendChild(orgRoot);//Finally append to the root
            //Element for description
            Element descElement = doc.createElement("Description");
            descElement.setAttribute("url", "http://www.ddlabinc.com");
            //Creation of text node
            Text text = doc.createTextNode("A division for innovative development");
            descElement.appendChild(text);
            orgRoot.appendChild(descElement);
            //Child Element Employees for list of Employees
            Element empsElement = doc.createElement("Employees");
            //Element for first employee
            Element empElement = doc.createElement("Employee");
            empElement.setAttribute("Department", "Developement");//Setting the attribute
            empElement.setAttribute("ID", "007");
            //Element for Address
            Element adrsElement = doc.createElement("Address");
            adrsElement.setAttribute("Type", "Permanent");
            adrsElement.setAttribute("Id", "001");//Setting the attribute
            //Element for location which is the child of Address Element
            Element locationElement = doc.createElement("Location");
            //Text Node for location element
            Text locationText = doc.createTextNode("Bangalore");
            locationElement.appendChild(locationText);
            //Text node for country
            Element countryElement = doc.createElement("Country");
            Text countryText = doc.createTextNode("India");
            countryElement.appendChild(countryText);
            adrsElement.appendChild(countryElement);

            adrsElement.appendChild(locationElement);
            empElement.appendChild(adrsElement);

            //Add Another address
            Element adrsElement1 = doc.createElement("Address");
            adrsElement1.setAttribute("Type", "Present");
            adrsElement1.setAttribute("Id", "002");//Setting the attribute
            //Element for location which is the child of Address Element
            Element locationElement1 = doc.createElement("Location");
            //Text Node for location element
            Text locationText1 = doc.createTextNode("Helsinki");
            locationElement1.appendChild(locationText1);
            //Text node for country
            Element countryElement1 = doc.createElement("Country");
            Text countryText1 = doc.createTextNode("Finland");
            countryElement1.appendChild(countryText1);
            adrsElement1.appendChild(countryElement1);

            adrsElement1.appendChild(locationElement1);
            empElement.appendChild(adrsElement1);
            //End of Address

            empsElement.appendChild(empElement);

            orgRoot.appendChild(empsElement);

            new DomXmlCreate().generateXml(doc);
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }

}
```

XML generation using SAX

```
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.sax.TransformerHandler;
import javax.xml.transform.stream.StreamResult;

import org.xml.sax.helpers.AttributesImpl;

public class TestXmlGenerationSAX
{
      public static void main(String[] args) throws Exception
      {
            TransformerFactory tf = TransformerFactory.newInstance();

            SAXTransformerFactory stf = (SAXTransformerFactory)tf;
            TransformerHandler th = stf.newTransformerHandler();
            th.setResult(new StreamResult(System.out));
            th.startDocument();

            //Attributes for Organisation Element
            AttributesImpl fieldAttrs = new AttributesImpl();
            fieldAttrs.addAttribute("", "", "name", "", "DDLAB INC");
            fieldAttrs.addAttribute("", "", "type", "", "Reasearch and Development");
            //Element creation for Organisation
            th.startElement("", "", "Organisation", fieldAttrs);
            {
                  //Element for Description
                  AttributesImpl descAttrs = new AttributesImpl();
                  descAttrs.addAttribute("", "", "url", "", "http://www.ddlabinc.com");
                  th.startElement("", "", "Description", descAttrs);
                  String descElementTextData = "A division for innovative development";
                  th.characters(descElementTextData.toCharArray(), 0,
descElementTextData.length());
                  th.endElement("", "", "Description");//End of Element for Description
                  {
                        th.startElement("", "", "Employees", null);
                        {
                              //For Permanent Address
                              AttributesImpl adrsAttrs = new AttributesImpl();
                              adrsAttrs.addAttribute("", "", "Id", "", "001");
                              adrsAttrs.addAttribute("", "", "Type", "", "Permanent");
                              th.startElement("", "", "Address", adrsAttrs);
                              th.startElement("", "", "Country", null);
                              String countryElementTextData = "India";
                              th.characters(countryElementTextData.toCharArray(), 0,
countryElementTextData.length());
                              th.endElement("", "", "Country");
                              th.startElement("", "", "Location", null);
                              String locationElementTextData = "Bangalore";
                              th.characters(locationElementTextData.toCharArray(), 0,
locationElementTextData.length());
                              th.endElement("", "", "Location");
                              th.endElement("", "", "Address");

                              //For Present Address
                              AttributesImpl adrsAttrs1 = new AttributesImpl();
                              adrsAttrs1.addAttribute("", "", "Id", "", "002");
                              adrsAttrs1.addAttribute("", "", "Type", "", "Present");
                              th.startElement("", "", "Address", adrsAttrs1);
                              th.startElement("", "", "Country", null);
                              String countryElementTextData1 = "Finland";
                              th.characters(countryElementTextData1.toCharArray(), 0,
countryElementTextData1.length());
                              th.endElement("", "", "Country");
                              th.startElement("", "", "Location", null);
                              String locationElementTextData1 = "Helsinki";
                              th.characters(locationElementTextData1.toCharArray(), 0,
locationElementTextData1.length());
                              th.endElement("", "", "Location");
                              th.endElement("", "", "Address");
                        }
                        th.endElement("", "", "Employees");//End of employee
                  }
                  th.endElement("", "", "Organisation");//End of organisation
            }
            th.endDocument();
      }

}
```

XML generation using Stax(Event API)

```
package com.ddlab.rnd.xml.gen.stax;

import javax.xml.stream.XMLEventFactory;
import javax.xml.stream.XMLEventWriter;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndDocument;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartDocument;
import javax.xml.stream.events.StartElement;

public class MyStaxEventGenerator
{
      public static void main(String[] args) throws Exception
      {
            XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();

            XMLEventWriter eventWriter = outputFactory
            .createXMLEventWriter( System.out );

            // Create a EventFactory
            XMLEventFactory eventFactory = XMLEventFactory.newInstance();

            StartDocument startDocument = eventFactory.createStartDocument();
            eventWriter.add(startDocument);

            StartElement configStartElement = eventFactory.createStartElement("",
                        "", "Organisation");
            eventWriter.add(configStartElement);

            {
                  StartElement se1 = eventFactory.createStartElement("",
                              "", "Address");
                  eventWriter.add(se1);

                  Attribute attrib1 = eventFactory.createAttribute("Id", "007");
                  Attribute attrib2 = eventFactory.createAttribute("Type", "Permanent");
                  eventWriter.add(attrib1);
                  eventWriter.add(attrib2);

                  Characters characters = eventFactory.createCharacters("Some Value");
                  eventWriter.add(characters);

                  EndElement ee1 = eventFactory.createEndElement("", "", "Address");
                  eventWriter.add(ee1);
            }

            EndElement endElement = eventFactory.createEndElement("", "","Organisation");
            eventWriter.add(endElement);

            EndDocument endDocument = eventFactory.createEndDocument();
            eventWriter.add(endDocument);

            eventWriter.close();
      }
  }
```

XML generation using Stax(Stream API)

```
package com.ddlab.rnd.xml.gen.stax;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;

public class MyStaxStreamGenerator
{
    public static void main(String[] args) throws Exception
    {
        XMLStreamWriter xsw = XMLOutputFactory.newInstance()
        .createXMLStreamWriter(System.out);
      xsw.writeStartDocument();
      xsw.writeStartElement("Root");
      xsw.writeAttribute("Name", "Value");
      for( int i = 0 ; i < 5 ; i++ )
      {
        xsw.writeStartElement("Child");
            xsw.writeCharacters("some-value-"+i);
            xsw.writeEndElement();
      }
      xsw.writeEndElement();
      xsw.writeEndDocument();
      xsw.flush();
      xsw.close();
    }
}
```

Difference between Push and Pull Parsers in XML

In the field of XML parsing there are broadly two parsing models called Pull and Push. Pull parsing is ideally suited for applications XML pipeline processing ie XML transformation. In pull parsing, application can be naturally structured and information can be pulled from XML when needed as application can pull next event when is ready to process it. Streaming pull parsing refers to a programming model in which a client application calls methods on an XML parsing library when it needs to interact with an XML infoset--that is, the client only gets (pulls) XML data when it explicitly asks for it. A push parsing model is a model where the parser pushes parsing events to the application. In case of push parsing model all the information will be available whether we require in the application or not.

Pull parsing has these general advantages over push parsing:

In pull parsing, events are generated by the parsing application, thus providing parsing regulation to the client, rather than the parser.
Pull parsing's code is simpler and it has less libraries than push parsing.
Pull parsing clients can read multiple XML documents simultaneously.
Pull parsing allows you to filter XML documents and skip parsing events.

A push model parser generates events until the XML document is completely parsed. But pull parsing is regulated by the application—so parse events are generated by the application. This means with StaX, you can suspend parsing, skip elements while parsing, and parse multiple documents. With the DOM API, you end up parsing the complete XML document into a DOM structure, thus reducing parsing efficiency.
With StAX, parsing events get generated as the XML document gets parsed. Click here to see a performance comparison of StAX with other parsers.

Push parsers are Crimson, Xerces, JAXP, DOM4J
SAX is also a push parser.
Stax is a pull parser.

<u>How many ways can you parse an XML document ?</u>

Basically there are four ways of parsing the XML document.
1. Tree based parsing which is DOM
2. Event based parsing which SAX
3. XPath API for parsing the pin point XML element
4. Streaming API for parsing which is STAX.

More About JAXB

Capsule

In case of generation of XML using JAXB, the following annotations are used.

@XmlRootElement()
@XmlAccessorType()
@XmlElement()
@XmlAttribute()
@XmlType()
@XmlValue()
@XmlTransient()

@XmlAccessorType is used to create binding for the fields, getter or setter methods or for public members.
Basically it tells to JAXB from which component to receive the data to create xml.
If you write @XmlAccessorType(XmlAccessType.FIELD), all the data from the fields will be considered to create the XML. For that reason we have to specify the annotation for the fields of the bean.
Let us see the example below. It has the following flavours.

@XmlAccessorType(XmlAccessType.FIELD)
@XmlAccessorType(XmlAccessType.PROPERTY) ---Default
@XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
@XmlAccessorType(XmlAccessType.NONE)

```
package com.ddlab.rnd.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement( name="Org" )
@XmlAccessorType( XmlAccessType.FIELD )
@XmlType(propOrder={"name","id","emps"}) //~~ Proper sequence of elements
public class Org
{
	@XmlElement(name="Id") private String id;
	@XmlElement(name="Name") private String name;
	private Employees emps;

	public Employees getEmps() {
		return emps;
	}
	public void setEmps(Employees emps) {
		this.emps = emps;
	}
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
```

```
        public void setName(String name) {
                this.name = name;
        }

}
```

In the above case, @XmlAccessorType(XmlAccessType.FIELD) has been specified to get the actual value from the fields and all the fields have been provided with annotations like the below.

```
@XmlElement(name="Id") private String id;
@XmlElement(name="Name") private String name;
```

@XmlAccessorType(XmlAccessType.PROPERTY)
If you specify XmlAcccessType.PROPERTY, there will be binding between java beans getter() or/and setter() method
to create the XML file. Let us see the example below.

```
@XmlElement(name="Name")
public String getName() {
      return name;
}

@XmlElement(name="Id")
public String getId() {
      return id;
}
```

You can specify either in case of getter() or setter() method. Let us see the example below.

```
@XmlElement(name="Id")
public void setId(String id) {
      this.id = id;
}

@XmlElement(name="Name")
public void setName(String name) {
      this.name = name;
}
```

@XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
It is used to create binding for the public members. The public members can be public methods() or public fields.Example is give below.

```
@XmlElement(name="Profile") public String profile;

@XmlElement(name="Id")
public void setId(String id) {
      this.id = id;
}
```

@XmlAccessorType(XmlAccessType.NONE)
In this particular case, none of the fields will be taken into consideration for binding unless they are
specified by annotations. You can freely specify the annotations in the private fields, public methods any where. Xml file will be created. For safer side you can provide this and provide annotations for the

public fields and methods of your choice.

If you do not provide the annotation @XmlAccessorType() in the class level, by default it is @XmlAccessorType(XmlAccessType.PROPERTY). If you provide annotation in the either gettter or setter, xml bind will happen and xml file will be created.

@XmlType(propOrder={"name","id","profile"})
The above annotation is used to create the Xml document with the xml elements in a proper order. It is related to the sequence in the xml schema. Sometimes it is necessary to have the proper order of the xml elements. For this reason it is used. There is another point in this case, if you have any field or method and if you do not specify in the @XmlType and propOrder, it will throw exception.

@XmlTransient()
This annotation is used when you do not want to create an Xml element in a document. For example there will be a password field or method and you do not want to create an element Example below.

@XmlTransient() public String pwd;

@XmlValue()
This annotation is used to create the text content in the Xml element when an element has some attributes. Let us see the example below.

We want to create an element like this "<Desc type="IT" unit="111">Some Description</Desc>"
In this case we have to create a class called "Desc" and we have to create an extra filed called value and we have to provide the annotation like "@XmlValue()". Let use see the complete example.

```
package com.ddlab.rnd.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlValue;

@XmlAccessorType(XmlAccessType.FIELD)
public class Desc
{
	@XmlAttribute(name="unit") private String unit;
	@XmlAttribute(name="type") private String type;
	@XmlValue() private String value;

	public String getValue() {
		return value;
	}
	public void setValue(String value) {
		this.value = value;
	}
	public String getUnit() {
		return unit;
	}
	public void setUnit(String unit) {
		this.unit = unit;
	}
	public String getType() {
```

```
                return type;
        }
        public void setType(String type) {
                this.type = type;
        }

}
```

@XmlElement(defaultValue="defaultElVal")

This annotation with the defaultValue is very special because it creates confusion. **Always remember that it does not creat the xml element with the defaultValue whatever it is mentioned at the time of XML generation. It is only used at the time of unmarshalling the xml payload.**

Let us see a complete example on Xml generation using Jaxb.
The initial proposed Xml struture is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Org>
    <Name>Ideal</Name>
    <Id>1234</Id>
    <Desc type="IT" unit="2">Software Company</Desc>
    <Employees>
        <Emp id="E001">
            <Name>Deba</Name>
            <Address type="Permanent">
                <RoadNo>12</RoadNo>
                <Street>aaa</Street>
            </Address>
            <Address type="Current">
                <RoadNo>212</RoadNo>
                <Street>sss</Street>
            </Address>
        </Emp>
        <Emp id="E002">
            <Name>Piku</Name>
            <Address type="Permanent">
                <RoadNo>12</RoadNo>
                <Street>aaa</Street>
            </Address>
            <Address type="Current">
                <RoadNo>212</RoadNo>
                <Street>sss</Street>
            </Address>
        </Emp>
    </Employees>
</Org>
```

To generate the above structure we have the following java classes.

```
package com.ddlab.rnd.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
@XmlAccessorType(XmlAccessType.FIELD)
public class Address
{
	@XmlElement( name="RoadNo") private String roadNo;
	@XmlElement( name="StreetNo") private String streetNo;
	@XmlAttribute( name="Type") private String type;

	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	public String getRoadNo() {
		return roadNo;
	}
	public void setRoadNo(String roadNo) {
		this.roadNo = roadNo;
	}
	public String getStreetNo() {
		return streetNo;
	}
	public void setStreetNo(String streetNo) {
		this.streetNo = streetNo;
	}
}
```

```
package com.ddlab.rnd.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlValue;

@XmlAccessorType(XmlAccessType.FIELD)
public class Desc
{
	@XmlAttribute(name="unit") private String unit;
	@XmlAttribute(name="type") private String type;
	@XmlValue() private String value;

	public String getValue() {
		return value;
	}
	public void setValue(String value) {
		this.value = value;
	}
	public String getUnit() {
		return unit;
	}
	public void setUnit(String unit) {
		this.unit = unit;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}

}
```

```
package com.ddlab.rnd.jaxb;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;

@XmlAccessorType( XmlAccessType.FIELD )
public class Emp  {
	@XmlElement( name="Id") private String id;
	@XmlElement( name="Name") private String name;
	@XmlElement( name="Address") private List<Address> adrs;

	public List<Address> getAdrs() {
		return adrs;
	}
	public void setAdrs(List<Address> adrs) {
		this.adrs = adrs;
	}
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}

package com.ddlab.rnd.jaxb;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement( name="Employees")
@XmlAccessorType( XmlAccessType.FIELD )
public class Employees
{
	@XmlElement(name="Emp") private List<Emp> emp;

	public List<Emp> getEmp() {
		return emp;
	}

	public void setEmp(List<Emp> emp) {
		this.emp = emp;
	}
}
```

```
package com.ddlab.rnd.jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement( name="Org" )
@XmlAccessorType( XmlAccessType.FIELD )
@XmlType(propOrder={"name","id","desc","emps"})
public class Org
{
	@XmlElement(name="Id") private String id;
	@XmlElement(name="Name") private String name;
	private Employees emps;
	@XmlElement(name="Desc") private Desc desc ;

	public Desc getDesc() {
		return desc;
	}

	public void setDesc(Desc desc) {
		this.desc = desc;
	}

	public Employees getEmps() {
		return emps;
	}

	public void setEmps(Employees emps) {
		this.emps = emps;
	}
	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

For XML generation, refer the following java class.

```
package com.ddlab.rnd.jaxb;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;

public class TestXmlGenerator
{
        public static void main(String[] args) throws Exception
        {
                Desc desc = new Desc();
                desc.setType("IT");
                desc.setUnit("Banking");
                desc.setValue("A virtual company");

                Org org = new Org();
                org.setId("Id-001");
                org.setName("DDLAB Inc");

                org.setDesc(desc);
                Employees emps = new Employees();
                List<Emp> empList = new ArrayList<Emp>();
                for( int i = 0 ; i < 2 ; i++ )
                {
                        Emp emp = new Emp();
                        emp.setId("E-"+i);
                        emp.setName("Ename-"+i);
                        List<Address> adrsList = new ArrayList<Address>();
                        Address adrs = new Address();
                        adrs.setRoadNo("R-"+i);
                        adrs.setStreetNo("St-"+i);
                        adrs.setType("P");
                        adrsList.add(adrs);
                        emp.setAdrs(adrsList);
                        empList.add(emp);
                }

                emps.setEmp(empList);
                org.setEmps(emps);

                JAXBContext jctx = JAXBContext.newInstance(Org.class);
                Marshaller marshaller = jctx.createMarshaller();
                marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
                marshaller.marshal(org, System.out);
        }

}
```

For data retriever, refer the following java class.

```
package com.ddlab.rnd.jaxb;
import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;

public class TestXmlDataRetriever
{
	public static void main(String[] args) throws Exception
	{
		JAXBContext jctx = JAXBContext.newInstance(Org.class);
		Unmarshaller unmarshaller = jctx.createUnmarshaller();
		Org org = (Org)unmarshaller.unmarshal( new File("xml/text1.xml"));
		System.out.println("Org Id - "+org.getId());

	}

}
```

Schema generation from java class using JAXB
An xml Schema can be generated from the java class using a tool called "schemagen". This tools is a part of Jaxb and it has been introduced in JDK 6 version. Let us see a working example.
The direct command is given below. If you have a jar file containing java bean classes, you can use like this.
"schemagen <package.classname> -d <directory where .xsd file will be generated> -cp <classpath and jar file path>"

```
schemagen com.ddlab.rnd.jaxb.Org -d F:\javaworkspaces\jobs\jaxb\schema -cp
F:\javaworkspaces\jobs\jaxb\schema\myschema.jar
```

To execute the above command, follow the steps
1. Create the relevent java bean classes.
2. Create a jar file
3. Execute the above command for the root class, in the above case "com.ddlab.rnd.jaxb.Org"

-d referes to the location where the schema file will be generated.
-cp refers to the classpath , indicating the list of jar files.

Using Ant

```
<target name="generate.schema" depends="compile">
   <exec executable="schemagen">
     <arg line="com.ddlab.rnd.jaxb.Org"/>
     <arg line="-d ${schema.dir}"/>
     <arg line="-cp ${build.dir}/${jar.file.name}"/>
   </exec>
</target>
```

Let us see a complete example

```
package com.ddlab.rnd.jaxb;
public class Org
{
        private String id;
        private String name;
        private Employees emps;

        public Employees getEmps() {
                return emps;
        }
        public void setEmps(Employees emps) {
                this.emps = emps;
        }
        public String getId() {
                return id;
        }
        public void setId(String id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }

}

package com.ddlab.rnd.jaxb;
public class Employees
{
        private Emp emp;

        public Emp getEmp() {
                return emp;
        }

        public void setEmp(Emp emp) {
                this.emp = emp;
        }

}

package com.ddlab.rnd.jaxb;
public class Emp
{
        private String id;
        private String name;
        public String getId() {
                return id;
        }
```

```
            public void setId(String id) {
                        this.id = id;
            }
            public String getName() {
                        return name;
            }
            public void setName(String name) {
                        this.name = name;
            }

}

<project name="schema-gen" basedir="." default="generate.schema">

    <property name="src.dir"    value="${basedir}/src"/>
    <property name="bin.dir"    value="${basedir}/bin"/>
    <property name="build.dir"    value="${basedir}/build"/>
    <property name="schema.dir"    value="${basedir}/schema"/>
            <property name="jar.file.name"    value="beans.jar"/>

    <target name="init">
        <delete dir="${bin.dir}"/>
        <delete dir="${build.dir}"/>
            <delete dir="${schema.dir}"/>
        <mkdir dir="${bin.dir}"/>
        <mkdir dir="${build.dir}"/>
            <mkdir dir="${schema.dir}"/>
    </target>

    <target name="compile" depends="init">
           <javac srcdir="${src.dir}" destdir="${bin.dir}" />
               <jar destfile="${build.dir}/${jar.file.name}" basedir="${bin.dir}"></jar>
    </target>

            <target name="generate.schema" depends="compile">
                        <exec executable="schemagen">
                          <arg line="com.ddlab.rnd.jaxb.Org"/>
                          <arg line="-d ${schema.dir}"/>
                          <arg line="-cp ${build.dir}/${jar.file.name}"/>
                        </exec>
            </target>

</project>
```

XML to XSD generation using third party java library
XML to XSD generation or xml document to xml schema can be generated by using the java libraries like **trang (thaiopensource) and castor.**

You can use ant script to generate the xsd file.

Ant script to generate xsd file from xml document using castor
```
---------------------------------------------------------------
<project name="xml2xsd" basedir="." default="xml2xsdgen">

        <property name="lib.dir"    value="${basedir}/lib"/>
        <property name="xml.dir"    value="${basedir}/xml"/>
        <property name="schema.dir"    value="${basedir}/xsd"/>

        <path id="classpath">
              <fileset dir="${lib.dir}">
                 <include name="**/*.jar" />
              </fileset>
           </path>

        <target name="init">
                <delete dir="${schema.dir}"/>
                <mkdir dir="${schema.dir}"/>
        </target>

        <target name="xml2xsdgen" depends="init">
                <taskdef name="schema-gen"
                        classname="org.castor.anttask.XMLInstance2SchemaTask"
                        classpathref="classpath" />
                <schema-gen file="${xml.dir}/org.xml" xmlschemafilename="${schema.dir}/org-castor.xsd" />
        </target>
</project>
```

For castor, the following jar files are used.
castor-1.3.2/castor-1.3.2-anttasks.jar
castor-1.3.2/castor-1.3.2-codegen.jar
castor-1.3.2/castor-1.3.2-core.jar
castor-1.3.2/castor-1.3.2-ddlgen.jar
castor-1.3.2/castor-1.3.2-xml-schema.jar
castor-1.3.2/castor-1.3.2-xml.jar
castor-1.3.2/castor-1.3.2.jar
castor-1.3.2/commons-lang-2.4.jar
castor-1.3.2/commons-logging-1.0.4.jar
castor-1.3.2/log4j-1.2.16.jar

Ant script to generate xsd/dtd file from xml document using trang

```
<project name="xml2xsd" basedir="." default="xml2xsdgen">

	<property name="lib.dir"    value="${basedir}/lib"/>
	<property name="xml.dir"    value="${basedir}/xml"/>
	<property name="schema.dir"    value="${basedir}/xsd"/>

	<path id="classpath">
		<fileset dir="${lib.dir}/trang">
			<include name="**/*.jar" />
		</fileset>
	</path>

	<target name="init">
		<delete dir="${schema.dir}"/>
		<mkdir dir="${schema.dir}"/>
	</target>

	<target name="xml2xsdgen" depends="init">
		<!-- To generate the XSD -->
		<exec command="java -jar ${lib.dir}/trang/trang.jar ${xml.dir}/org.xml ${schema.dir}/org-
trang.xsd"/>
		<!-- To generate the DTD -->
		<exec command="java -jar ${lib.dir}/trang/trang.jar ${xml.dir}/org.xml ${schema.dir}/org-
trang.dtd"/>
	</target>
</project>
```

The jar file used for trang is "trang.jar".

The trang.jar can be used to generate XSD or DTD from the XML schema.

How to validate an xml file against an XML schema

The example is given below.

```
import java.io.File;
import javax.xml.XMLConstants;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

public class SchemaValidator
{
	public static void main(String[] args) throws Exception
	{
//		File schemaFile = new File("xsd/org-trang.xsd");
		File schemaFile = new File("xsd/org-castor.xsd");
		File xmlFile = new File("xml/org.xml");
		SchemaFactory factory = 
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
		// 2. Compile the schema.
//		Schema schema = factory.newSchema (new StreamSource (""));
		Schema schema = factory.newSchema(schemaFile);
		// 3. Get a validator from the schema.
		Validator validator = schema.newValidator();
		// 4. Check the input xml file
		validator.validate(new StreamSource(xmlFile));
		String STATUS="valid";

		System.out.println(STATUS);
	}

}
```

XML Schema to Java beans generation using JAXB XJC Compiler

JAXB provides the feature to generate java class binding from the XSD or DTD file to generate and retrieve XML payload. JAXB provides a tool called "XJC Compiler" which is used to generate java classes from XSD or DTD file. It is a part of JDK 6. We can write an Ant script to generate java classes from XSD or DTD.

Let us see the example below.

```
<project name="schema2javabeans" basedir="." default="schema2java">

	<property name="gen.dir"          value="${basedir}/generated"/>
	<property name="xsd.file.name"    value="${basedir}/schema/org.xsd"/>
	<property name="dtd.file.name"    value="${basedir}/schema/org.dtd"/>

	<target name="init">
		<delete dir="${gen.dir}"/>
		<mkdir dir="${gen.dir}"/>
	</target>

	<target name="schema2java" depends="init">
```

```
        <!-- XSD to Java bean classes -->
        <exec executable="xjc">
            <arg line="-xmlschema ${xsd.file.name}"/>
            <arg line="-d ${gen.dir}"/>
            <arg line="-p com.ddlab.rnd.xjc.xsd"/>
        </exec>
        <!-- DTD to Java bean classes -->
        <exec executable="xjc">
            <arg line="-dtd ${dtd.file.name}"/>
            <arg line="-d ${gen.dir}"/>
            <arg line="-p com.ddlab.rnd.xjc.dtd"/>
        </exec>
        </target>

</project>
```

XML generation from xml schema using JAXB XJC

Let us see the complete example below.

```
<project name="schema2javabeans" basedir="." default="build">

        <property name="bin.dir"          value="${basedir}/bin"/>
        <property name="gen.dir"          value="${basedir}/generated"/>
        <property name="build.dir"          value="${basedir}/build"/>
        <property name="xsd.file.name"     value="${basedir}/schema/org.xsd"/>
        <property name="dtd.file.name"     value="${basedir}/schema/org.dtd"/>
        <property name="jar.file.name"     value="${build.dir}/schema.jar"/>

        <target name="init">
                <delete dir="${gen.dir}"/>
                <mkdir dir="${gen.dir}"/>
                <delete dir="${bin.dir}"/>
                <mkdir dir="${bin.dir}"/>
                <delete dir="${build.dir}"/>
                <mkdir dir="${build.dir}"/>
        </target>

        <target name="schema2java" depends="init">
                <!-- XSD to Java bean classes -->
                <exec executable="xjc">
                        <arg line="-xmlschema ${xsd.file.name}"/>
                        <arg line="-d ${gen.dir}"/>
                        <arg line="-p com.ddlab.rnd.xjc.xsd"/>
                </exec>
        </target>

        <target name="build" depends="schema2java">
                <javac srcdir="${gen.dir}" destdir="${bin.dir}" />
           <jar destfile="${jar.file.name}/" basedir="${bin.dir}"/>
        </target>
</project>
```

The java code to generate the XML file is given below.

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import com.ddlab.rnd.xjc.xsd.ObjectFactory;
import com.ddlab.rnd.xjc.xsd.Org;

public class TestXmlGeneration
{
	public static void main(String[] args) throws Exception
	{
		Org org = new Org();
		org.setId("Id1");
		org.setName("Name1");

		ObjectFactory objFact = new ObjectFactory();
		JAXBContext jcontext = JAXBContext.newInstance(Org.class);
		Marshaller marshaller = jcontext.createMarshaller();
		marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
		marshaller.marshal(org, System.out);
	}
}
```

Design of a Pull and Push Parser System for Streaming XML

Aleksander Slominski
Indiana University Computer Science Department [*]
{e-mail: `aslom@extreme.indiana.edu` }

Abstract

An XML parser facilitates in simplifying the process of manipulating XML documents. The two commonly used models for parsing XML are pull and push. The Simple API for XML (SAX) [5] is the industry standard for parsing based on the push model. However, no standard exists for pull parsing. In this paper we propose the design, API and implementation, XML Pull Parser 2.0 (XPP2), that allows for both pull and push based parsing. We discuss the various features of SAX and provide a description of existing technologies to parse XML. We describe our implementation of this API and show how a SAX driver can be built on top of XPP2.

Keywords: XML, parsing, push, pull, SAX.

1 Introduction

This paper addresses the following questions:

- What are the shortcomings of the event based push model such as SAX? Why is not the optimal solution for applications?
- How does one design an XML parser that provides benefits of both the push and pull parsing model?
- Is it possible to have a simple and efficient pull API for this purpose?
- What are the design decisions that allow for high performance and low memory utilization?

2 Current XML APIs for streaming parsing

The XML 1.0 [6] specification was adopted in February 1998 as a W3C Recommendation. A number of programming APIs have since been developed to parser XML. These include SAX,

[*]Department of Computer Science, 150 S Woodlawn Avenue, Bloomington, IN 47405. Ph: 812 855 8305 Fax: 812 855 4829

DOM, JDOM, DOM4J, libxml, RXP and NanoXML. Amongst these SAX has emerged as a de-facto standard for event-based XML parsing. It was developed as collaborative effort by the members of the XML-DEV mailing list. The current version of SAX, named SAX2, has support for XML namespaces [3], filter chains, querying and setting features and properties in the parser.

There exist some APIs for pull parsing but none in widespread use. For example kXML [9] is pull parser specifically designed for small devices and needs to be used with Java 2 Micro Edition. It provides simple API however the API and the implementation are tied together. With kXML it is easy to create an XML object tree in memory. However it is not easy to achieve streaming performance as for each event a new object to represent it is created and returned to the user even if the user just wants to skip parts of XML.

Xerces Pullable is a pull parser API that is used by Xalan. It has a pullable SAX model. The version for Xerces1 and Xerces 2 are both different. An application needs to request the Xerces parser [2] to parse only some portion of input and as soon as the parser invokes SAX callback it stops parsing. Then the application can ask for more input that it needs to continue parsing. This is a very good approach for applications that have already invested in SAX infrastructure but want more control over parsing. However this API is not standardized and building more clearly defined pull API is desirable. This is exactly the intent of XPP2 API, described in later sections, to allow to use Xerces 2 Native Interface (XNI) with API that was specifically designed for XML pull parsing.

GNOME C libxml [4] and XMLIO [7] are other examples of XML parsers. GNOME C libxml is not a streaming parser as it just produces a list of events that can be traversed. XMLIO allows for pulling data from XML but its API is specifically designed to unmarshal data structures.

3 Push and Pull: complementary sides of XML parsing

Most SAX parsers are built on top of a pull parsing layer. It is an interesting challenge to expose to the user both pull and push layers. This allows an application to use pull parsing when needed without having to stop using SAX API.

Pull and push parsing models are not the only two ways to parse XML. It is possible to convert pull parser into a push model. This is possible as during pull parsing the caller has control over parsing and can push events (as an example please see description SAX2 driver for XPP2 at the end of this paper). It is also possible to convert push into pull parser but requires to buffer all events converted from SAX callbacks or an extra thread that can be used to ”pull” more data from SAX parser and is kept suspended until the user asks for more events.

This approach is best exemplified by Pull Parser Wrapper for SAX [8] that allows conversion from SAX compliant parser into a XML pull parser. However as it requires an extra thread to convert events and this requirement has proved to be very difficult in server environments and impossible in EJB container environments. For an example of such difficulties it is useful to look on design considerations of Apache SOAP Axis (see for example notes from Axis Face-to-Face Meeting at http://xml.apache.org/axis/docs/F2F-2.html)

Pull parsing is ideally suited for applications that needs to transform input XML to other formats. As such transformation is typically complex they must maintain state during parsing. Using SAX would require to maintain state between callbacks to be able to determine correct action to SAX event. In pull parsing, application can be naturally structured and information can be pulled from XML when needed as application can pull next event when is ready to process it.

Lets look at an example data record below that represents information about one person that has name and two addresses:

```
<person>
<name>Alek</name>
<home_address>
<street>101 Sweet Home</street>
<phone>333-3333</phone>
</home_address>
<work_address>
<street>303 Office Street</street>
<phone>444-4444</phone>
</work_address>
</person>
```

This XML input can naturally be mapped to Java classes:

```
class Person {
        String name;
        Address homeAddress;
        Address workAddress;
}

class Address {
        String street;
        String phone;
}
```

To process it with SAX one would need to have startElement callback to do some work depending on start tag name but this will be not enough as phone must be put in different place depending on whether or not the previous tag was home_address or work_address. Here we give an example of such code:

```
    Person person = new Person();
    StringBuffer elementContent = new StringBuffer();
    Address address;

    public void startElement(String uri, String local, String raw,
                             Attributes attrs) throws SAXException {
        buf.clear();
    }
    public void characters(char ch[], int start, int length)
        throws SAXException {
        buf.append(ch, start, length);
    }
    public void endElement(String uri, String local, String raw)
        throws SAXException {
        if("name".equals(local)) {
           person.name = elementContent.toString;
         } else if("home_address".equals(local)) {
           address = person.homeAddress = new Address();
         } else if("work_address".equals(local)) {
           address = person.workAddress = new Address();
         } else if("phone".equals(local)) {
           address.phone = buf.toString();
         } else if("street".equals(local)) {
           address.street = buf.toString();
         } else {
           throw new SAXException("unexpected element "+local);
         }

    }
```

Although on the first look the code may look sufficient there are some inherent problems because it does not maintain state between endElement callbacks. Therefore the code does not know what is its position in the parsed XML structure. One particular problem with such approach is that such SAX program will not validate input and can even produce incorrect conversions. For example this code would process input XML from below by incorrectly overriding home phone to 666-666 instead of 333-3333.

```
<person>
<name>Alek</name>
<home_address>
<phone>333-3333</phone>
</home_address>
<phone>666-6666</phone>
</person>
```

This can be fixed by keeping track of how deeply nested the start/end element is and using some extra flags to make sure that phone is set only once and address always corresponds only to the current home or work address. But it requires adding extra state variable and code to do validation.

When using pull parser the conversion of XML input into Person object is very natural and follows hierarchical relation between Person and Address objects. Let look on how it could be done if a simple pull parser was available:

```
        parser = new PullParser(input)
```

```
Person person = parsePerson(parser);

public Person parsePerson(PullParser parser) throws ValidationException
{
        Person person = new Person();
        while(true) {
           int eventTyppe = parser.next();
           if(eventType = parser.START_TAG) {
              String tag = parser.getStartTagName();
              if("name",equals(tag)) {
                person.name = readContent(parser);
              } else if("home_address",equals(tag)) {
                person.homeAddress = readAddress(parser);
              } else if("work_address",equals(tag)) {
                person.workAddress = readAddress(parser);
              } else {
                throw new ValidationException(
                  "unknown field "+tag+" in person record");
              }
           } else if(eventType == parser.END_TAG) {
             break;
           } else {
             throw new ValidationException("invalid XML input");
           }
        }
}
public Address parseAddress(PullParser parser) throws ValidationException
{
        Address person = new Address();
        while(true) {
           int eventTyppe = parser.next();
           if(eventType = parser.START_TAG) {
              String tag = parser.getStartTagName();
              if("street",equals(tag)) {
                address.street = readContent(parser);
              } else if("phone",equals(tag)) {
                address.phone = readContent(parser);
              } else {
                throw new ValidationException(
                  "unknown field "+tag+" in person record");
              }
           } else if(eventType == parser.END_TAG) {
             break;
           } else {
             throw new ValidationException("invalid XML input");
```

```
                }
        }
        public String readContent(PullParser parser) throws ValidationException
        {
                if(parser.next() != parser.CONTENT) {
                  throw new ValidationException("expected string content");
                }
                String content = parser,readContent();
                if(parser.next() != parser.END_TAG) {
                  throw new ValidationException(
                    "expected end tag after string content");
                }
                return content;
        }
```

The structure naturally reflects the organization of data structures and therefore is much easier to maintain. The state is kept naturally on the stack as a consequence of method calls that can be nested as much as necessary. Notice also that with pull parsing the second input will trigger a ValidationException.

4 Design of Pull API

As we have seen pull parsing has some advantages and therefore it is very important to provide developers with one consistent and simple API to work with.

For such an API to stay minimal we need to expose only a few things to the user. We need to expose the parser state, such as : START_TAG, END_TAG, CONTENT, END_DOCUMENT. This number of states is minimal for any XML parser. We will also need an ability to to get next state, for example by calling method named next(). Finally for each state we will need to retrieve information associated with the parser state (current event). When we reach END_DOCUMENT there is no more input and in this case no extra information is needed. However for CONTENT, to get element content as String, readContent() can be used. Then for start and end tags we need the name of the tag and its namesapace: getLocalName() and getNamespaceUri(). Finally START_TAG is the hard part as it can also contain the list of attributes declared in this start tag. It is a good idea to make it look like SAX Attributes/AttributeList to simplify potential conversion of pull events into SAX push callbacks.

So we can say that such minimal Pull Parser API should have at least:

```
public interface XmlPullParser {
    /** signal logical end of xml document */
    public final static byte END_DOCUMENT = 1;
    /** start tag was just read */
    public final static byte START_TAG = 2;
    /** end tag was just read */
    public final static byte END_TAG = 3;
    /** element content was just read */
    public final static byte CONTENT = 4;


    int next() throws PullParserException;
    public String readContent() throws XmlPullParserException;
    public void readEndTag(XmlEndTag etag) throws XmlPullParserException;
    public void readStartTag(XmlStartTag stag) throws XmlPullParserException;
    public String getLoacalName();
    public String getNamespaceUri();
}
```

It is useful to represent XmlStartTag and XmlEndTag as separate interfaces but sharing common functionality in XmlTag. It is possible to have state retrieval methods in XmlPullParser instead of creating extra interfaces such as XmlEndTag and XmlStartTag. However having separate interfaces allows us to maintain parsing state in those classes easily and allows to record state of parsing for any number of steps in the past.

```
public interface XmlTag {
  public String getNamespaceUri();
  public String getLocalName();
}

public interface XmlEndTag extends XmlTag {
}

public interface XmlStartTag extends XmlTag {

    public int getAttributeCount();
    public String getAttributeNamespaceUri(int index);
    public String getAttributeLocalName(int index);
    public String getAttributeValue(int index);
    public String getAttributeValueFromName(String namespaceUri,
                                            String localName);

}
```

This API will have to be extended to allow for more efficient handling of namespace declarations, setting parser input, resetting parser state for parser reuse and some other utility methods. However the core API for XML pull parser will probably be similar to what is shown above.

We should mention here that use of XmlTag interfaces is not essential for such API and may be abandoned when memory size constraints are of premium such as is the case with handheld devices in J2ME. In these cases embedding all state into one XmlPullParser interface will be very advantageous.

5 Implementing XML Pull Parser

Any XML parser will have to do XML tokenization. For push parsers this can be combined with higher level parser as push callbacks can be called as soon as interesting input is read. For pull parsing this is different as when interesting event is seen the pull parser must return to the user. Therefore it needs to maintain internal state to be able to continue parsing when the user requests it.

In XPP2 we have made it possible to parse some parts of input using pull API and then for some document XML subtrees to use push parses that provide SAX2 API. From a SAX2 callback user can continue to use pull parsing or even create another nested SAX2 push parser and this recursive nested parsing can be as deep as required.

5.1 Java Performance Constraints

When implementing XPP2 one of the most important task was to assure good overall performance. We identified that creation of XmlStartTag object for every start tag would be incurring a very high overhead but we did not want to remove possibility to record parser state for later use (otherwise building XML object model in memory from XPP2 events would be difficult). The other consideration was to make sure that the parser performance is not too low when compared with simple string tokenization and other XML parsers. This requirements also helped to estimate acceptable XML parsing overhead. An interesting fact was determining the difference in processing char[] as compared to String even when advanced JIT such as Hotspot is used (see [1] for more details on performance evaluations).

5.2 Tokenizer

The most important part of XML Pull Parser is tokenizer that is responsible for breaking input into tokens that are later used to produce events.

To make this task as efficient as possible the tokenizer in XPP2 is a state machine driven by input characters.

To avoid reading the input character by character, the input is internally buffered. In J2ME it is important to make sure that buffer size does not exceed some hard limits (so that XML parsing does not consume the available memory). If hard limit is exceed such as for very long element content an exception will be thrown and parsing will be stopped. It is also desirable to establish soft limit on internal buffer size to indicate what is desired buffer size (it must be always less than hard limit).

5.3 Parser

The parser in XPP2 is implements all of XML 1.0 specification requirements for non-validating XML parser except for parsing internal DTD. This decision was made to ensure that the size of XPP2 is not too big and moreover as XML schemas are going to replace DTDs we think that supporting DTD parsing is no longer desired by users. The other limitation of the parser is that input must be in UNICODE as represented by Java Reader and char type. The user is responsible for detecting input encoding (such as UTF8) and use Java built-in mechanisms to transform it to Reader.

5.4 Support for streaming

XPP2 pull model is inherently well suitable for streaming. The internal buffer support for soft and hard limit sizes allow us to ensure that during parsing memory consumption is kept under tight control.

5.5 Namespace handling

In some situation when XML input does not contain XML namespaces there can be slight boost in performance (around 5%) by avoiding overhead of validation and maintaining XML namespace prefixes declarations. Therefore XPP2 allows to decide before parsing if namespaces should switched on or off.

6 Supporting multiple implementations of XPP2 API

XPP2 consists of two distinctive APIs and a default implementation. The XPP2 API was designed to allow plugging different implementations in such way that user code does not have to change at all. This is achieved with use of XmlPullParserFactory that is responsible for creation of XmlPullParser instances. The factory to be used is parametrized based on existence of special file with factory name on CLASSPATH or name of factory class passed to XmlPullParserFactory.newInstance() method. We do not use java System properties by default as they are not available on J2ME platform, however are easy to use with this code:

```
XmlPullParserFactory.newInstance(
  System.getProperty(XmlPullParserFactory.DEFAULT_PROPERTY_NAME));
```

As an example of such additional implementation of XPP2 API we are providing implementation that uses Xerces 2 XNI pull model. Therefore the user can use XPP2 either with fast default implementation or fully validating Xerces2 engine.

6.1 SAX2 driver for XPP2

As it was mentioned earlier it is desirable and clearly feasible to translate pull events into SAX2 push callbacks. As part of XPP2 we are providing utility class that can take XPP2 API (implemented by default implementation or Xerces 2 or any other) and transform it into SAX2 XmlReader.

6.2 C++ version

We have also written a C++ version of XPP1 (Xml Pull Parser version 1). As of now, the C++ version of XPP2 is not available however differences between XPP1 and XPP2 are not that big for just XmlPullParser interface. C++ version has some additional limitations such as it does not support UNICODE fully (it is possible to compile it with chat or wchar_t support) and user needs to assume that input is encoded as UTF8 (or when compiled with wchar_t assume UTF16). Additionally XPP1/C++ does not support streaming and requires all input to be passed in one array.

7 Conclusions

We hope that this paper will be useful introduction to pull parsing and will give a motivation to use as an alternative to SAX API. It also demonstrates advantages of this alternative mode of parsing, compares pull with push models (with emphasize on SAX2) and argues that both are needed.

An API that allows both push and pull parsing should be made available to users to allow them to use the best of both parsing models (and assure that user has the most powerful tools to accomplish XML related tasks). XPP2 API and its implementations are hopefully a first step into this direction.

We would like to welcome readers to visit our web page http://www.extreme.indiana.edu/soap/xpp/ and browse more detailed documentation, download source code and send us comments.

References

[1] Aleksander Slominski. On Performance of Java XML Parsers, visited 05-01-01. http://www.cs.indiana.edu/~ aslom/exxp/.

[2] Apapche Foundation. Xerces Java Parser 2, visited 09-01-01. http://xml.apache.org/xerces2-j/.

[3] World Wide Web Consortium. Namespaces in XML, 1-14-99. http://www.w3.org/TR/REC-xml-names/.

[4] Daniel Veillard. The XML C library for Gnome (libxml), visited 04-01-01. http://xmlsoft.org/.

[5] D. Megginson et al. Sax 2.0: The simple api for xml, visited 07-01-00. http://www.saxproject.org/.

[6] Tim Bray et al. Extensible markup language (xml) 1.0 (second edition 6 october 2000), visited 03-01-01. http://www.w3.org/TR/2000/REC-xml.

[7] Paul T. Miller. XMLIO - An XML input/output library for C++ applications, visited 01-01-01. http://www.fxtech.com/xmlio/.

[8] Stefan Haustein. XML pull wrapper for SAX parsers, visited 02-01-01. http://www.trantor.de/xml/.

[9] Stefan Haustein. kXML Project, visited 05-01-01. http://www.kxml.org/.

Sun's 2005 Worldwide Java Developer Conference

StAX or SAX ? Push or Pull Parsing XML ?

Neeraj Bajaj

Sun Microsystems, inc.

http://www.sun.com

BoF 9778

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- Event APIs : XMLEventReader

Cursor 2 Event

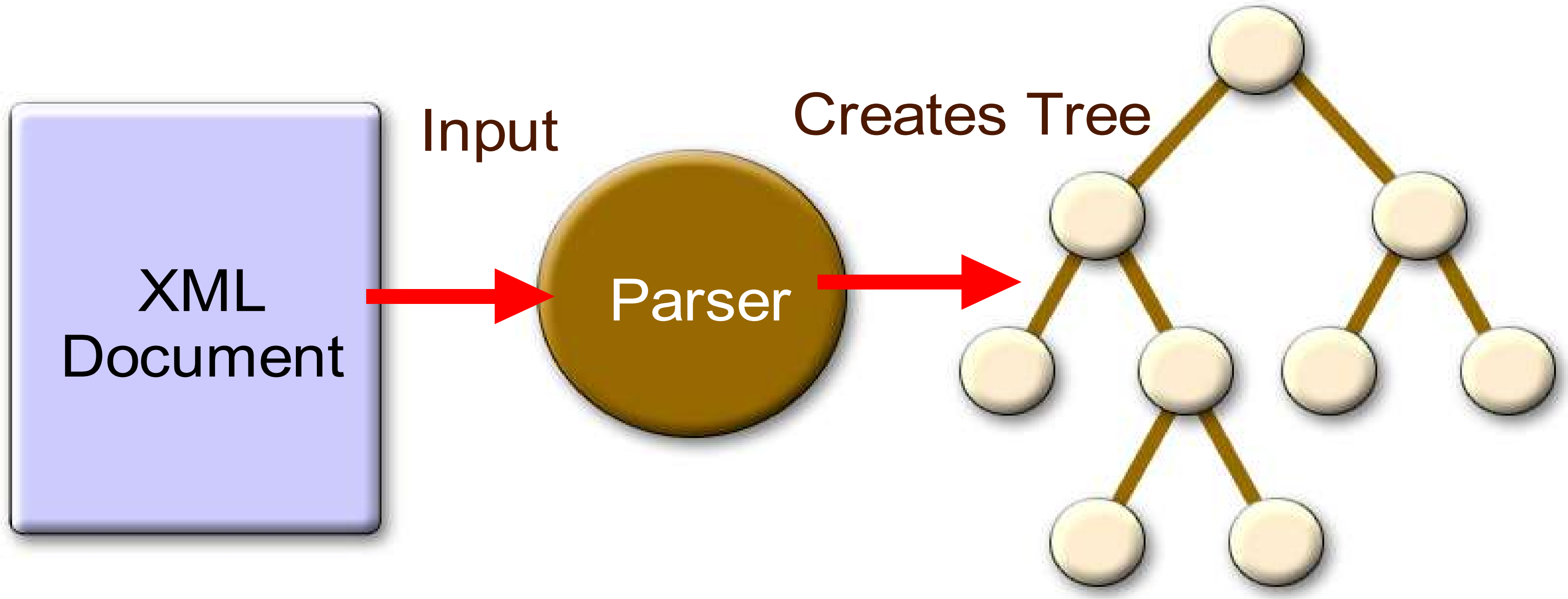
Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

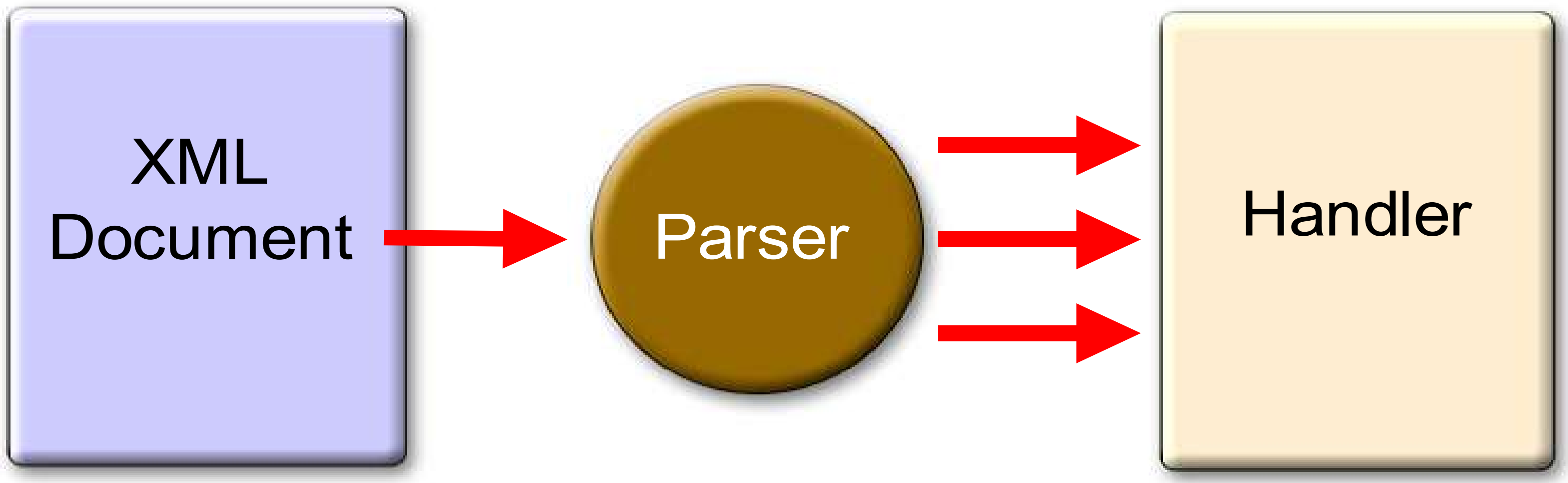
XML Parsing Models

- Object (DOM, JDOM etc.)
- Push (SAX)
- Pull (StAX) new

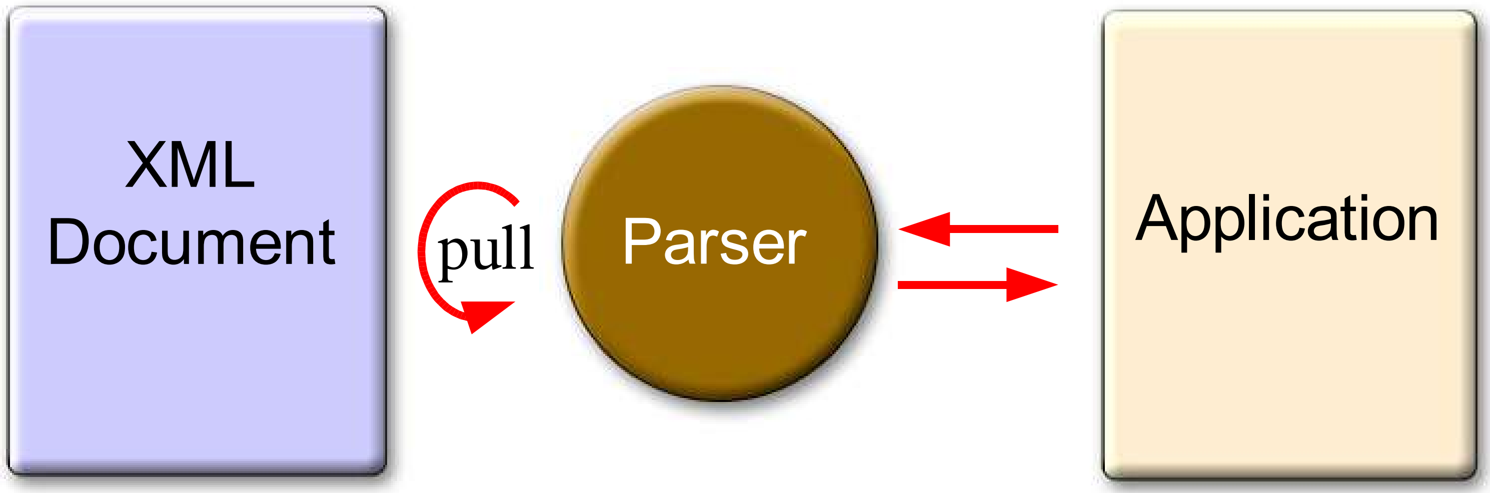
Tree Model, DOM



Push Model, SAX



Pull Model, StAX



StAX Pull Model

Two approach..

1. Cursor APIs: XMLStreamReader

2. Event APIs: XMLEventReader

StAX Events

1) Namespace
2) StartDocument
3) EndDocument
4) StartElement
5) EndElement
6) Attribute
7) EntityDeclaration
8) EntityReference
9) Notation
10) PI
11) DTD
12) Characters
13) Comment

Agenda

- XML Parsing Models
- StAX
  - **Cursor APIs : XMLStreamReader**
  - Event APIs : XMLEventReader
- Cursor 2 Event
- Stax Vs. SAX
- Sun Java Streaming XML Parser (SJSXP)

XMLStreamReader

- Based on “iterator” pattern
  - hasNext()
  - Next()
- Most efficient way to read XML data
- Consumes less memory
- Represents a *cursor* moving forward

Creating XMLStreamReader

```
//create XMLInputFactory
XMLInputFactory factory = XMLInputFactory.newInstance();

//configure factory
factory.setXMLReporter(myXMLReporter);
factory.setXMLResolver(myXMLResolver);
factory.setProperty(..);

//create XMLStreamReader
XMLStreamReader reader =
factory.createXMLStreamReader(..);
```

Reading XML using XMLStreamReader

```
//create XMLStreamReader
XMLStreamReader reader =
factory.createXMLStreamReader(..);
int eventType = reader.getEventType();

//continue reading until there are more events
while(reader.hasNext()){
   //move to the next event in the XML stream
   eventType = reader.next();
   //pass the reader
   process(reader);
}
```

next() function

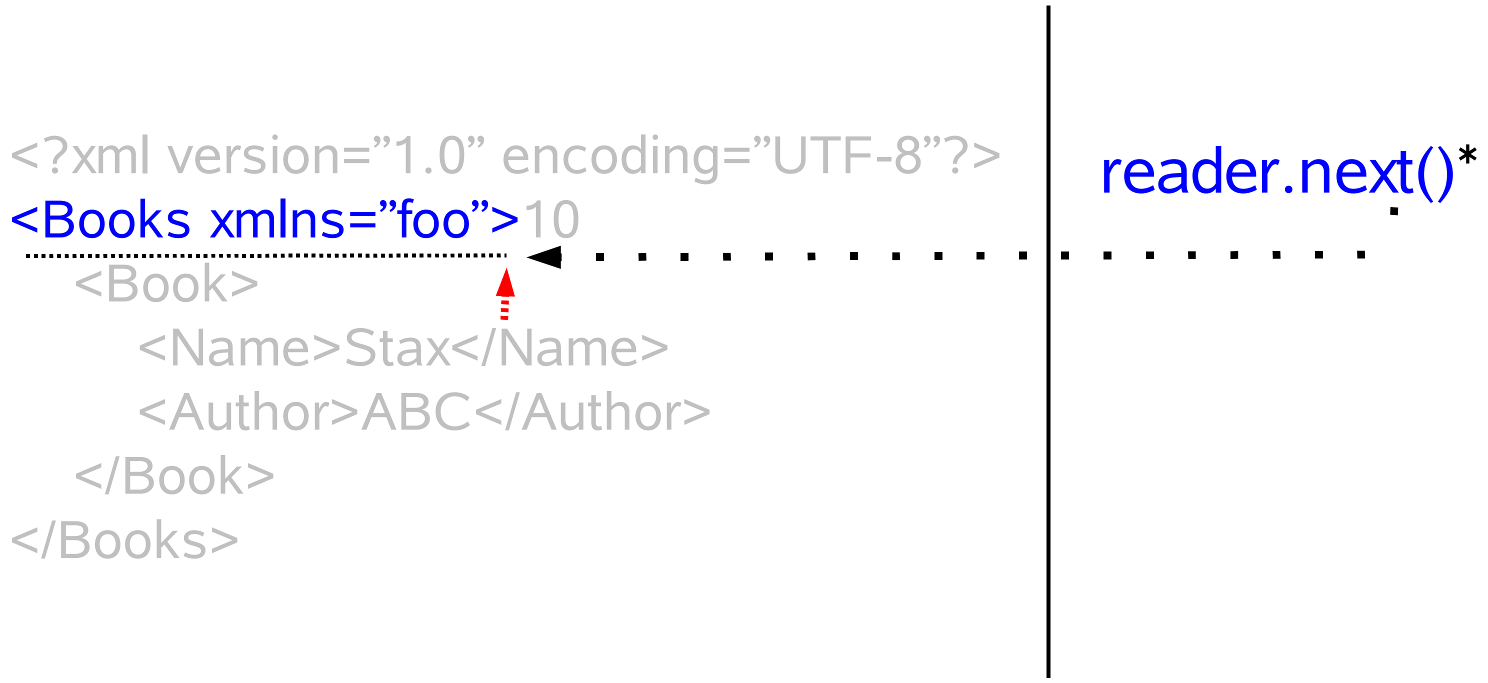
- drives the parser to read the next event on input stream.
- returns the integer code corresponding to the current parse event.
- After next() control is back with application, has the option to
  - Stop parsing
  - Call next() to go to next event
  - Read the information associated with the event

Reading XML decl.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Books xmlns=”foo”>10
   <Book>
      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

```
reader.getVersion()
reader.getEncoding()
reader.standAloneSet()
```

Navigation



```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns="foo">10
   <Book>
      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

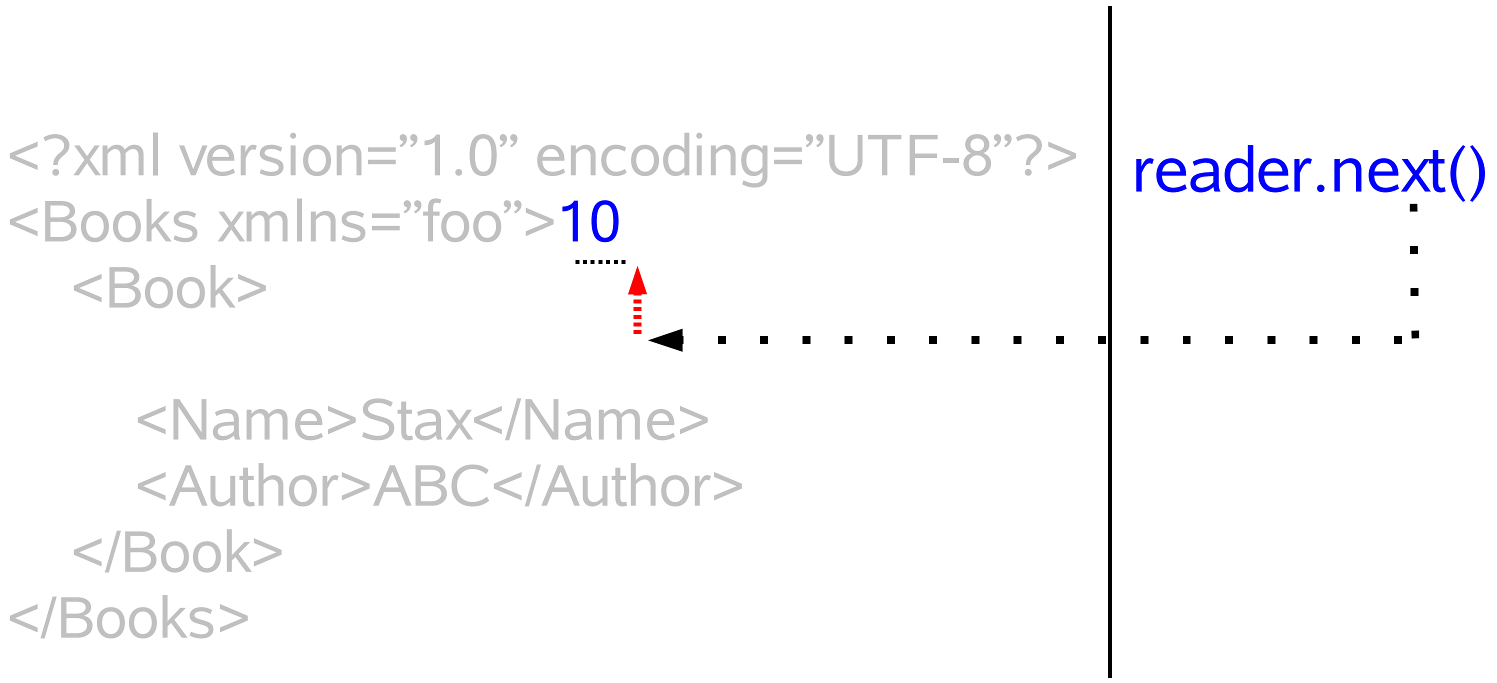
* No new line character after xml declaration

Reading Element /Attributes

```
<?xml version="1.0" ...>
<Books xmlns="foo">10

  <Book>
    <Name> Stax </Name>
    <Author> ABC</Author>
  </Book>
</Books>
```

```
reader.next() == ELEMENT

reader.getName():QName
reader.getLocalName():String
reader.getPrefix():String
reader.isAttributeSpecified()

 reader.getAttributeName(index):Q
 reader.getAttributePrefix(index)
 ........
 reader.getNamespaceURI()
 reader.getNamespaceContext()
 ........
```

Navigation



```
<?xml version="1.0" encoding="UTF-8"?>
<Books xmlns="foo">10
   <Book>

      <Name>Stax</Name>
      <Author>ABC</Author>
   </Book>
</Books>
```

Reading Characters

```
<?xml version=”1.0” ...>
<Books xmlns=”foo”>10

  <Book>
    <Name> Stax </Name>
    <Author> ABC</Author>
  </Book>
</Books>
```

reader.next() == CHARACTERS

reader.getText():String

reader.getTextCharacters():Char[]
reader.getTextStart():int
reader.getTextLength():int

Reading characters

- Don't assume next() has read all character data !
- Text data may be split into several calls
- Set “isCoalesce” property to true to receive all the adjacent character data as one event.

XMLStreamReader

- Behavior is function of its state.
- Need to understand which functions are valid for a particular state (or event)
- Certain functions are valid only for a particular state. For example:
  - illegal to call getPI() when next() returns 'Element' event

Agenda

XML Parsing Models

StAX

- Cursor APIs : XMLStreamReader
- **Event APIs : XMLEventReader**

Cursor 2 Event

Stax Vs. SAX

Sun Java Streaming XML Parser (SJSXP)

XMLEventReader

- Easy to use
- Flexible
- Easy pipelining
- Data returned as immutable XMLEvents

Creating XMLEventReader

```
//create XMLInputFactory
XMLInputFactory factory = XMLInputFactory.newInstance();

//configure factory
factory.setXMLReporter(myXMLReporter);
factory.setXMLResolver(myXMLResolver);
factory.setProperty(..);

//create XMLEventReader
XMLEventReader reader =
factory.createXMLEventReader(..);
```

Reading XML using XMLEventReader

```
//create XMLEventReader
XMLEventReader reader =
factory.createXMLEventReader(..);

//continue reading until there are more events
while(reader.hasNext()){
   //move to the next event in the XML stream
   XMLEvent event = reader.next();
   if(event.isStartElement()){
      StartElement se = event.asStartElement();
      QName  name = se.getName();
      Iterator attributes = se.getAttributes();
}
```

Agenda

- XML Parsing Models
- StAX
  - Cursor APIs : XMLStreamReader
  - Event APIs : XMLEventReader
- **Cursor 2 Event**
- Stax Vs. SAX

Cursor 2 Event

- You can switch from Cursor to Event while reading same XML document.
- Two ways
  - Create XMLEventReader wrapping XMLStreamReader
  - Allocate event using XMLEventAllocator

Create XMLEventReader Wrapping XMLStreamReader

```
XMLStreamReader sr = //get XMLStreamReader

if(sr.isStartElement() &&
    sr.getLocalName().equals("Book")){

//create XMLEventReader wrapping XMLStreamReader
XMLEventReader reader =
factory.createXMLEventReader(sr);

while(reader.hasNext()){
   //move to the next event in the XML stream
   XMLEvent event = reader.next();
}
}
```

Using XMLEventAllocator

```
XMLStreamReader sr = //get XMLStreamReader
XMLEventAllocator allocator =
          new MyXMLEventAllocator();

if(sr.isStartElement() && sr.getLocalName().
equals("Book")){

    StartElement se =
    allocator.allocate(sr).asStartElement();

}
```

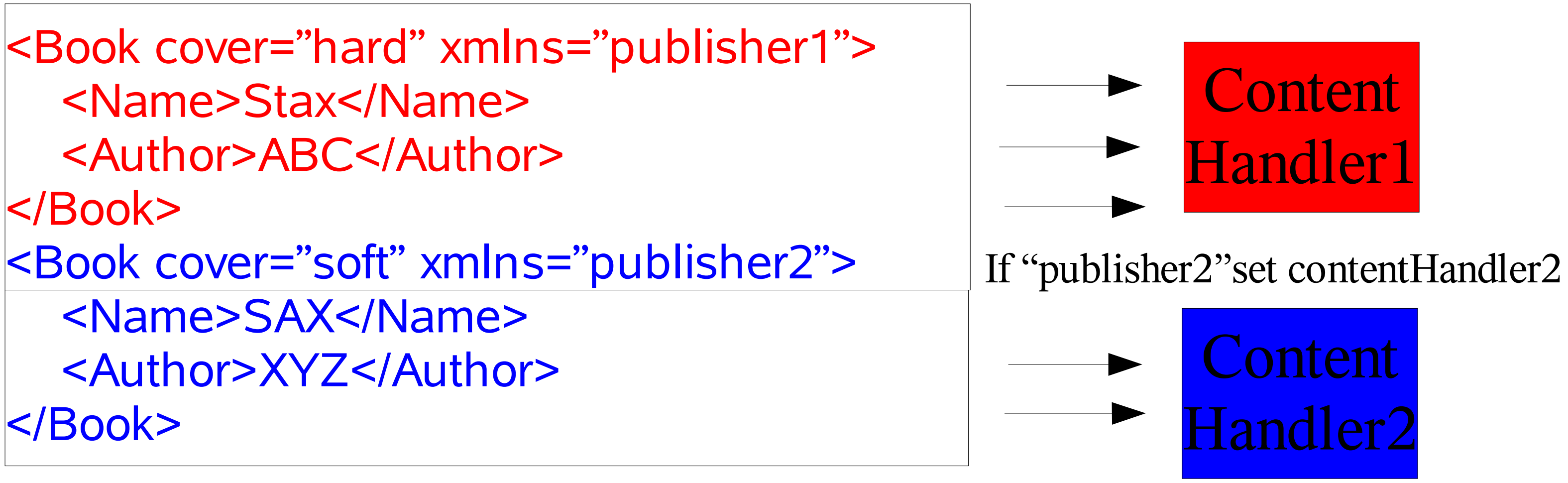
DEMO

Cursor 2 Event

Agenda

- XML Parsing Models
- StAX
  - Cursor APIs : XMLStreamReader
  - Event APIs : XMLEventReader
- Cursor 2 Event
- **Stax Vs. SAX**
- Sun Java Streaming XML Parser (SJSXP)

Reading with SAX

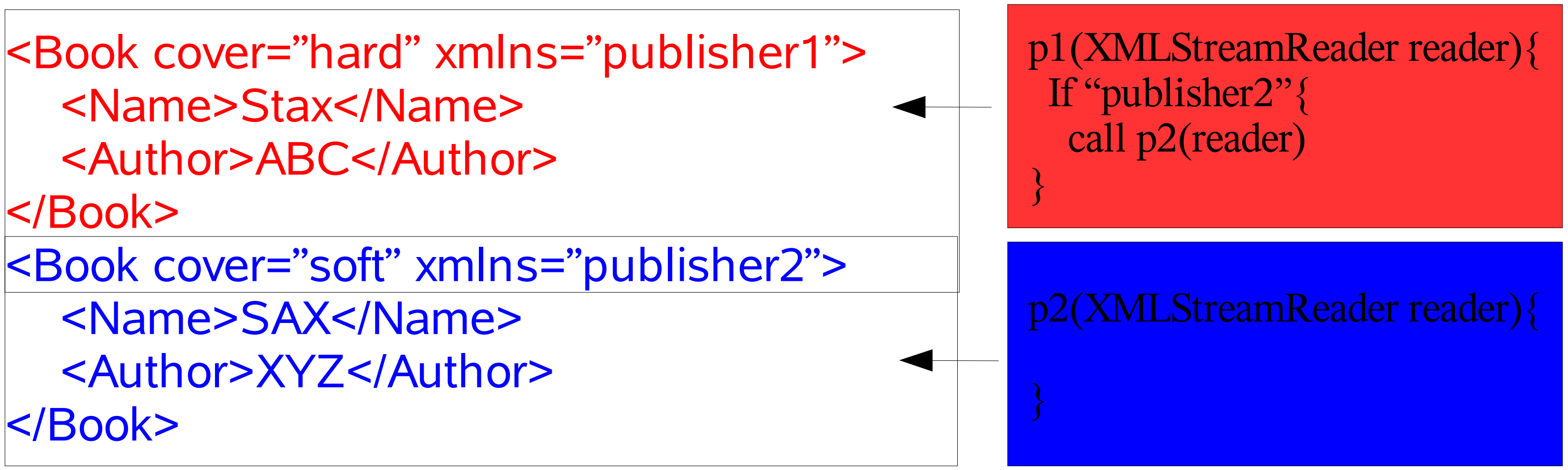


```
<Book cover=”hard” xmlns=”publisher1”>
    <Name>Stax</Name>
    <Author>ABC</Author>
</Book>
<Book cover=”soft” xmlns=”publisher2”>
    <Name>SAX</Name>
    <Author>XYZ</Author>
</Book>
```

Problem

- ContentHandler2 will not have details about 2nd Book element.
- Application has to do special handling to localize the processing of Book element in 'publisher2' domain.
- Less control over reading data.

Reading with StAX



```
<Book cover=”hard” xmlns=”publisher1”>
    <Name>Stax</Name>
    <Author>ABC</Author>
</Book>
<Book cover=”soft” xmlns=”publisher2”>
    <Name>SAX</Name>
    <Author>XYZ</Author>
</Book>
```

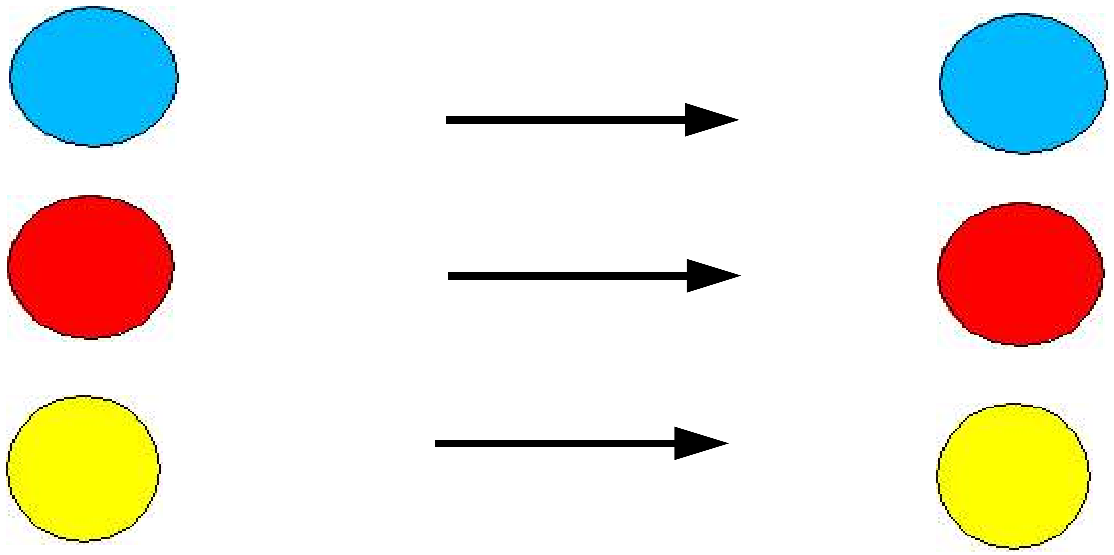
```
p1(XMLStreamReader reader){
  If “publisher2”{
    call p2(reader)
}
```

```
p2(XMLStreamReader reader){

}
```

Advantage

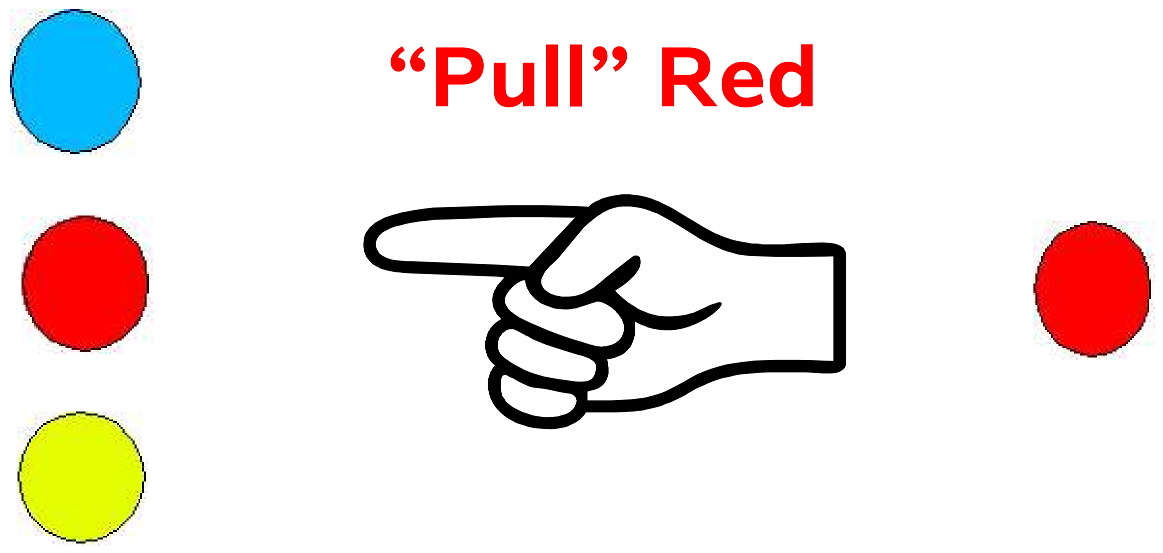
- Processing of Book elements can be done separately.
- Control over reading the data.
- Modular code.
- Easy to pass StreamReader to different parts of code.

SAX



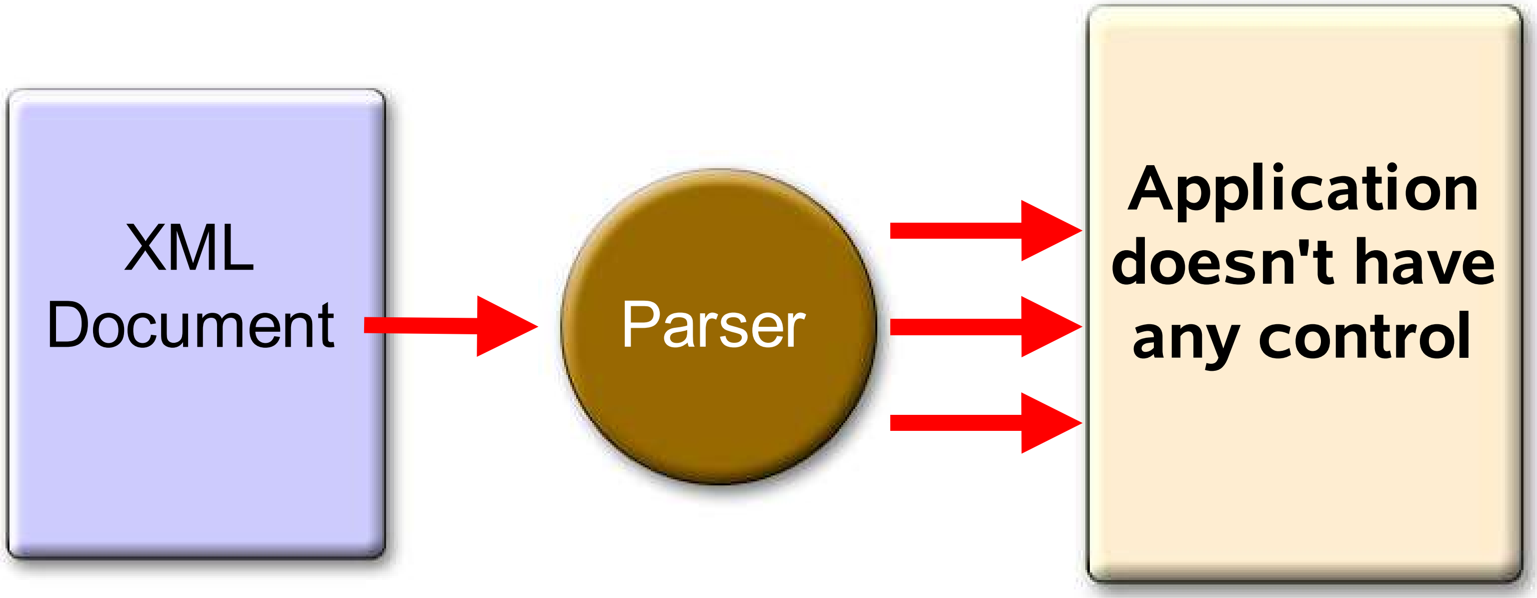
All the data is pushed

StAX



Allows selective reading

SAX Parsing



• It's like uncontrolled reaction of throwing events until all the data is consumed.
• Throw exception to “stop” parsing before endDocument callback

StAX Vs. SAX

- Application controls when and how much data to read
- easy to write code for parsing complex XML documents
- "Iterator" based

- Application has no control over parsing
- State machine becomes complex with complexity of XML doc.
- "Observer" based

StAX Vs. SAX

StAX

- Allows “selective” reading of data
- Allows computing of data lazily
- Easy and efficient to build 'push' layer on top of 'pull' layer.

SAX

- All data is pushed to application
- Values need to be calculated before pushing data
- Difficult to write efficient 'pull' layer on top of 'push' layer

StAX Vs. SAX

StAX

- Easy to read multiple docs. at a time with just a single thread.
- Easy to skip non-relevant parts of XML document.
- Very easy to stop, just do nothing.
- Read and Write APIs

SAX

- Very difficult to read multiple docs. at a time with single thread.
- Doesn't allow skipping parts of XML document.
- Throw exception to stop parsing.
- Need separate APIs to write

StAX Vs SAX

StAX	SAX
ErrorHandler can't be changed during parse	ErrorHandler can be changed during parse
EntityHandler can't be changed during parse	EntityHandler can be changed during parse
Modifying stream is not easy	Modifying stream is easy

Agenda

- XML Parsing Models
- StAX
    - Cursor APIs : XMLStreamReader
    - Event APIs : XMLEventReader
- Cursor 2 Event
- Stax Vs. SAX
- **Sun Java Streaming XML Parser (SJSXP)**

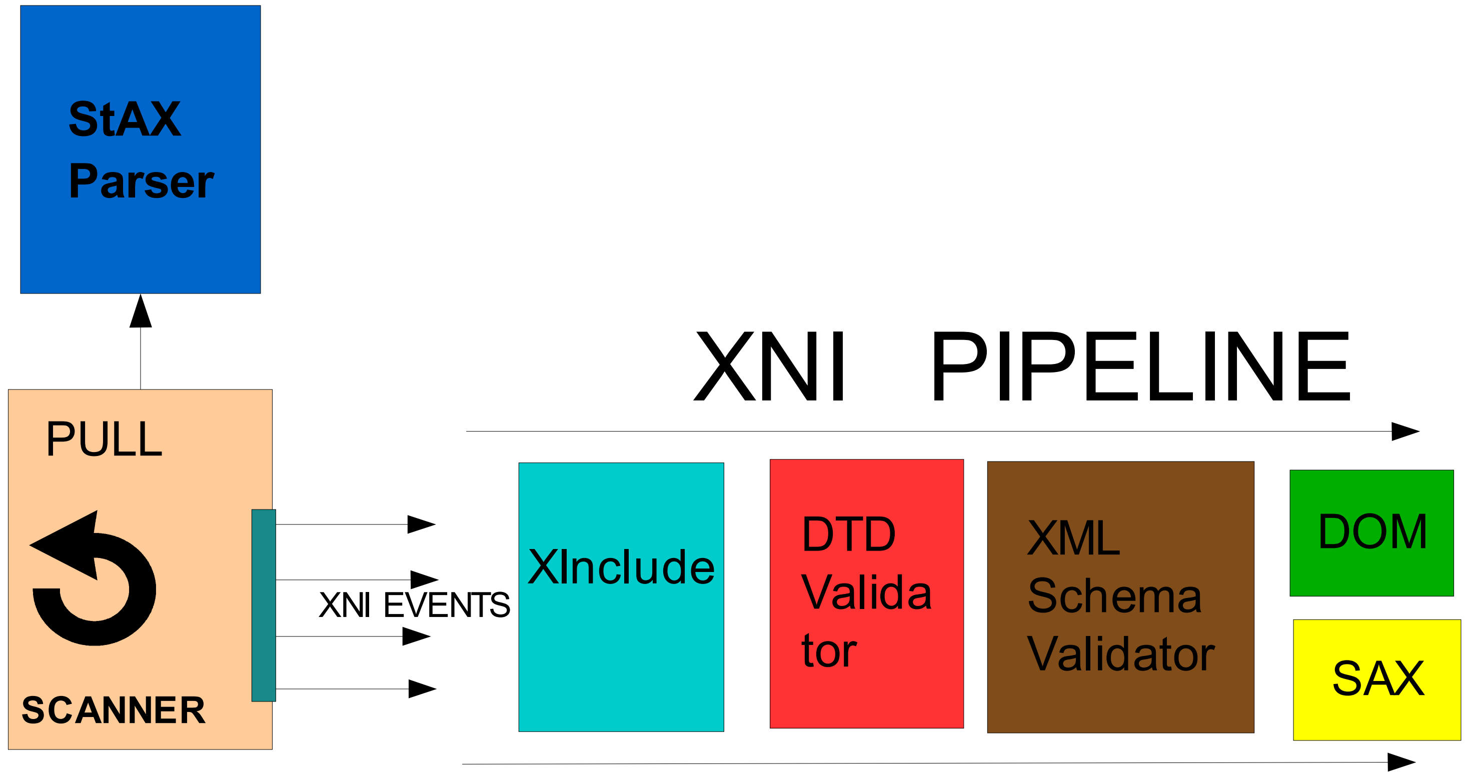
Sun Java^{TM} Streaming XML Parser (SJSXP)

- Sun's implementation of StAX, JSR 173.
- Highly performant.
- Built on Xerces2 code base
  - XML scanner has been “re-designed” to behave in pull fashion
  - Lot of performance improvements
- Fully compliant to W3C XML 1.0, Namespace 1.0
- Non Validating

Sun Java™ Streaming XML Parser (SJSXP)

- Binaries available on https://sjsxp.dev.java.net
- Stax impl. merged with JAXP impl. (Xerces) in JAXP 1.4
  - Development happens at http://jaxp.dev.java.net
  - Sources available at http://jaxp-sources.dev.java.net
- Will be part of Java™ platform (JDK 6.0)
- Bundled with Java™ Web Services Developer Pack 1.5, 1.6

StAX impl. & JAXP impl. merge



Sun Java™ Streaming XML Parser Road Map

- Add DTD validation support
- Add support for XML 1.1
- Add XInclude support

References

- JSR 173, http://www.jcp.org/en/jsr/detail?id=173
- JAXP 1.4 (with StAX), https://jaxp.dev.java.net
- Source code at https://jaxp-sources.dev.java.net
- Java Web Services Developer Pack 1.5/1.6
  http://java.sun.com/webservices/jwsdp/index.jsp
- Sun Stax implementation (SJSXP)
  https://sjsxp.dev.java.net
- StAX RI, http://dev2dev.bea.com/technologies/stax/index.jsp

Thank You

StAX or SAX ? Push or Pull Parsing XML ?

Neeraj Bajaj

Sun Microsystems, inc.

http://www.sun.com

BoF 9778