SAP XML Toolkit for Java

SAP J2EE Engine 6.20

Contents

1. Basic Concepts

1.1. XSL Overview

XSL (Extensible Style Language) consists of two separate halves – a transformation language and a formatting language. Each of these can function independently from each other. The transformation language basic function is to transform an XML document using a XSL Stylesheet Processor into another XML document. The transformed XML document can use DTD and markup from the original or only a specified set of tags.

1.2. XSL Stylesheet Processing

1.2.1. Tree Transformation

An XML Stylesheet Processor executes the whole process of transforming. It accepts an XML document and an XSL stylesheet and transforms the source XML document into another. Thus the tree transformation phase of the process is accomplished, constructing the result tree.

1.2.2. Formatting Phase

The second phase is performed through a formatter. For instance it can be a rendering engine inside a browser. The whole process is executed by including formatting semantics in the result tree. These semantics represent classes of formatting objects. Each node of the result tree is a formatting object. Additional formatting properties facilitate representation of the result tree.

1.3. JAXP Overview

JAXP is a standard Sun API that allows different XML-processing Java software to be easily plugged in and interchanged. The whole process consists of simply changing a system property, without dealing with all the pre-existing code. Problems may occur if the new parser does not support some of the features that the old one supplied.

JAXP includes interfaces and abstract classes that ensure access to DOM and SAX parsers, as well as to XSLT transformers.

In order to obtain a `SAXParser`, the user must first get its corresponding `SAXParserFactory` through the static `SAXParserFactory.newInstance()` method and then the `SAXParser` itself by the factory `newSAXParser()` method.

Similarly, DocumentBuilder-s (DOM parsers) and their factories are obtained through `DocumentBuilderFactory`, and Transformer-s (XSLT transformers) are obtained through `TransformerFactory`. Every factory's `newInstance()` method checks the corresponding system property to obtain information on the name of the class that implements it. If no such property has been set, a default implementation is returned.

For instance, a `SAXParser` can be obtained through:

```
SAXParserFactory f = SAXParserFactory.newInstance();
SAXParser p = f.newSAXParser();
```

And used through:

```
org.xml.sax.helpers.DefaultHandler h = new MyHandler();
p.parse(filename, h);
```

The JAXP view of SAP J2EE Engine XML tools is placed in the `com.inqmy.lib.jaxp` package. It supports the two basic API-s, DOM, SAX, and XSLT transformations.

In order to use SAP J2EE Engine XML tools, the following system properties should be set:

```
System.setProperty("javax.xml.parsers.DocumentBuilderFactory",
"com.inqmy.lib.jaxp.DocumentBuilderFactoryImpl");
System.setProperty("javax.xml.parsers.SAXParserFactory",
"com.inqmy.lib.jaxp.SAXParserFactoryImpl");
System.setProperty("javax.xml.transform.TransformerFactory",
"com.inqmy.lib.jaxp.TransformerFactoryImpl");
```

1.4. XML Schema Validator

XML Schema is a language defining the structure of an XML file. It is expected that soon it will completely replace the previous DTD language, because of its much greater expressive capabilities.

In order to use our implementation of XML Schema Validation, you have to do one of the following:

- Use the command line tool - *sch.bat*

- Use:
  ```
  DocumentBuilderFactory.setAttribute("http://www.inqmy.com/schem
  a", "myschema.xsd");
  ```

- For example:
  ```
  import com.inqmy.lib.schema.validator.SchemaValidator;
  boolean result =
  new SchemaValidator().validate(nodeXML, nodeXSD);
  ```
  where `nodeXML` and `nodeXSD` are the DOM trees of the XML instance and the schema respectively.

1.5. Schema-to-Java Generator

Though the purpose of the XML Schema is to validate XML instances. It is often used for defining abstract structures in a language-independent way. The development of a SOAP proxy generator for WSDL service descriptions enforced the appearance of such a utility.

The generator uses the Java naming convention when generating classes representing schema types. The transformation of a schema complex type name to a Java method or class identifier conforms to the part of the JAXRPC specification, which describes such name transformations.

One can use it in the following ways:

- Use the command line tool, *gen.bat*:

  ```
  Usage:
    sch filename.xsd output_directory_name output.package.name
  ```

- Use the following code:

  ```
  Schema schema = new Schema();
  schema.loadRaw(inputStream);
  schema.generateAll(new
  File("output_directory_name"),"output.package.name");
  ```

2. Java API for XML Parsing (JAXP)

The following section aims to represent to the reader several Java-oriented examples, which introduce the basic features of the parsing process.

2.1. Introduction to DOM

DOM is a standard, designed to provide the developer with easy to manipulate tree structure, which represents an XML document. This structure is appropriate for applications that need to access and manipulate the XML document directly. Since the whole tree is maintained in the memory DOM based application can deteriorate server performance.

DOM can be used either to traverse an existing DOM Tree, and extract specific information, or to build a DOM Document in memory, and then either forward it to some application, which will process it, or to serialize it, thus creating an XML File.

Now the most important methods and interfaces will be lined out. Although many not so important methods will be skipped, the whole definition and description of these methods can be found either in the javadoc documentation of the SAP XML Toolkit for Java, or on the *www.w3c.org* site – section DOM.

2.1.1. Building DOM Trees through Parsing

The easiest way to get a DOM Tree is to parse a XML file using a DocumentBuilder. Then you can traverse the `Document` object and extract what you need. The JAXP Framework provides several classes to do this.

In general all you need to do is to invoke `DocumentBuilderFactory.newInstance()` in order to get an instance of the `DocumentBuilderFactory`. Then you can use this instance to get a new `DocumentBuilder` – (`newDocumentBuilder`()). And finally use the `DocumentBuilder` instance to parse an XML file, using `DocumentBuilder.parse(InputSource)`. You can further specify whether you want your `DocumentBuilder` to be validating - `DocumentBuilderFactory.setValidating(boolean)`, or to be namespace-aware - `DocumentBuilderFactory.setNamespaceAware(boolean)`.

2.1.1.1. javax.xml.parsers.DocumentBuilderFactory

- `public static DocumentBuilderFactory newInstance()` – obtains a new instance of a `DocumentBuilderFactory`. This static method creates a new factory instance. The procedure to determine the `DocumentBuilderFactory` implementation class to be loaded is the following:
  - Use the `javax.xml.parsers.DocumentBuilderFactory` system property.
  - Use the *JAVA_HOME* (the parent directory where jdk is installed)*/lib/jaxp.properties* for a property file that contains the name of the implementation class keyed on the same value as the system property defined above.
  - Use the services API (as detailed in the JAR specification), if available, to determine the classname. The services API will look for a classname in the file *META-INF/services/javax.xml.parsers.DocumentBuilderFactory* in jars available to the runtime.
  - Platform default `DocumentBuilderFactory` instance.

Once an application has obtained a reference to a `DocumentBuilderFactory`, it can use the factory to configure and obtain parser instances.

- `public DocumentBuilder newDocumentBuilder()` - creates a new instance of a `DocumentBuilder` using the currently configured parameters.
- `public void setNamespaceAware(boolean awareness)` – specifies that the parser produced by this code will provide support for XML namespaces. By default the value of this is set to `false`.
- `public void setValidating(boolean validating)` – specifies that the parser produced by this code will validate documents as they are parsed. By default the value of this is set to `false`.

2.1.1.2. javax.xml.DocumentBuilder

- `public Document parse(InputSource is)`
- `public Document parse(InputStream is)`
- `public Document parse(Reader reader)`
- `public Document parse(File file)`
- `public Document parse(String uri)`

Parse the content of the given input source as an XML document and return a new DOM `Document` object.

- `public Document newDocument()` – obtains a new instance of a DOM `Document` object to build a DOM tree.

2.1.2. Traversing

2.1.2.1. org.w3c.dom.Node

This is the base class of all DOM Objects – `Document`, `Element`, `Attr`, `Text`, etc. It has several methods, which can be used to traverse a document.

- `public org.w3c.dom.Node getFirstChild()` - the first child of this node. If there is no such node, returns `null`.
- `public org.w3c.dom.Node getPreviousSibling()` - the node immediately preceding this node. If there is no such node, returns `null`.
- `public org.w3c.dom.Node getNextSibling()` - the node immediately following this node. If there is no such node, returns `null`.
- `public org.w3c.dom.NamedNodeMap getAttributes()` - a `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or `null` otherwise.
- `public org.w3c.dom.Document getOwnerDocument()` - the `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document` or a `DocumentType`, which is not used with any `Document` yet, this is `null`.
- `public org.w3c.dom.Node removeChild (org.w3c.dom.Node oldChild)` - removes the child node indicated by `oldChild` from the list of children, and returns it.
- `public org.w3c.dom.Node appendChild (org.w3c.dom.Node newChild)` - adds the node `newChild` to the end of the list of the children of this node. If the `newChild` is already in the tree, it is first removed.
- `public String getNodeName()` - the name of this node, depending on its type; see the table above.
- `public String getNodeValue()` - the value of this node, depending on its type; see the table above. When it is defined to be `null`, setting it has no effect.
- `public short getNodeType()` - a code representing the type of the underlying object, as defined above. The most used kinds of node-types are:
  - `Node.ELEMENT_NODE` – for element nodes
  - `Node.DOCUMENT_NODE` – for document nodes
  - `Node.TEXT_NODE` – for text nodes
  - `Node.COMMENT_NODE` – for comment nodes

- `public org.w3c.dom.Node getParentNode()` - the parent of this node. All nodes, except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `null`.
- `public org.w3c.dom.NodeList getChildNodes()` – returns a `NodeList` that contains all children of this node. If there are no children, this is a `NodeList` without nodes.

2.1.2.2. org.w3c.dom.Element

This class is used to represent an Element node. It may have children attributes. There are also methods to search for nodes with a specific name among the descendants of this node. For an Element node, invoking `Node.getNodeType()` returns `Node.ELEMENT_NODE`. Several methods from the Element interface are described below:

- `public String getAttribute (String name)` - retrieves an attribute value by name.
- `public void setAttribute (String name, String value)` - adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be the same as the value parameter. This value is a simple `String`. It is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` node, and then any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.
  To set an attribute with a qualified name and namespace URI, use the `setAttributeNS` method.
- `public NodeList getElementsByTagName (String name)` – returns a `NodeList` of all descendent `Elements` with a given tag name, in the order in which they are encountered in a preorder traversal of this `Element` tree.

2.1.2.3. org.w3c.dom.Text

This node type represents all the text content inside a `Document`.

- `public String getData()` - the character data of the node that implements this interface.

- `public void setData (String data)` – sets the character data for this node.

2.1.2.4. org.w3c.dom.Comment

This node type is used to represent a comment in the XML.

2.1.2.5. org.w3c.dom.Document

This is the root node of a XML Document. After the XML is passed, this is the type of node you get from the parser. It represents the whole XML document, that is – the root element, the processing instructions and comments on the root level, parts of the DTD (Entity declarations and Notations). For traversing the following relevant methods are used:

- `public Element getDocumentElement()` – this is a convenient attribute that allows direct access to the child node, that is the root element of the document.
- `public NodeList getElementsByTagName (String tagname)` – returns a `NodeList` of all the `Elements` with a given tag name in the order in which they are encountered in a preorder traversal of the `Document` tree.

2.1.2.6 org.w3c.dom.NodeList

This interface is used to represent a list of nodes. There are two methods, which are useful:

- `public Node item(int index)` – returns the `index-th` item in the collection. If `index` is greater than or equal to the number of nodes in the list, returns `null`.
- `public int getLength()` – the number of nodes in the list. The range of valid child node indices is `0` to `length-1` inclusively.

2.1.2.7 org.w3c.dom.NamedNodeMap

This interface is used to represent a collection of nodes, which are to be accessed by their names.

- `public Node getNamedItem(String name)` - retrieves a node specified by the name.

- `public Node item(int index)` – returns the `index-th` item in the map. If `index` is greater than or equal to the number of nodes in this map, returns `null`.
- `public int getLength()` – the number of nodes in this map. The range of valid child node indices is from `0` to `length-1` inclusively.

2.1.3. Building DOM Trees from a Document

To build a DOM Tree, you have to take an empty `Document` instance. In JAXP this task is performed by getting an instance of `DocumentBuilderFactory`. Then from this instance you get a new instance of a `DocumentBuilder`. And then from the instance of the `DocumentBuilder` you invoke `newDocument()`.

2.1.3.1. javax.xml.DocumentBuilder

- `public abstract Document newDocument()` – obtains new instance of a DOM `Document` object to build a DOM tree.

2.1.3.2. org.w3c.dom.Document

- `public Element createElement(String tagName)` – creates an element of the type specified. The instance returned implements the `Element` interface, so attributes can be specified directly on the returned object. In addition, if there are known attributes with default values, `Attr` nodes representing them are automatically created and attached to the element.
- `public Text createTextNode(String data)` - creates a `Text` node using the specified string.
- `public Comment createComment(String data)` - creates a `Comment` node using the specified string.
- `public ProcessingInstruction createProcessingInstruction (String target, String data)` - creates a `ProcessingInstruction` node using the specified name and data strings.
- `public Node importNode(Node importedNode, boolean deep)` - imports a node from another document to this document. The returned node has no parent (`parentNode` is `null`). The source node is not altered or removed from the original document. This method creates a new copy of the source node.

2.1.3.3. org.w3c.dom.Node

- `public Node insertBefore(Node newChild, Node refChild)` - inserts the node `newChild` before the existing child node `refChild`. If `refChild`

is `null`, insert `newChild` at the end of the list of children. If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed.

- `public Node replaceChild(Node newChild, Node oldChild)` - replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node. If `newChild` is a `DocumentFragment` object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is removed.
- `public Node removeChild(Node oldChild)` - removes the child node indicated by `oldChild` from the list of children, and returns it.
- `public Node appendChild(Node newChild)` - adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

2.1.4. DOM Examples

2.1.4.1. Building DOM through Parsing

Example 1:

```
public class JAXPDOMExample {
  public static void main(String args[]) {
    try {
      String xml= "data/rich_ii.xml";
      //get a DocumentBuilderFactory from the underlying
      //implementation
      DocumentBuilderFactory factory =
          DocumentBuilderFactory.newInstance();
      factory.setValidating(true);
      //get a DocumentBuilder from the factory
      DocumentBuilder builder = factory.newDocumentBuilder();
      //parse the document
      Document document = builder.parse(xml);

      DOMTraverser domTester = new DOMTraverser();
      domTester.traverse1(document);
    } catch (Exception e) {
      //if there was some error while parsing the file
      e.printStackTrace();
    }
  }
```

2.1.4.2. Traversing

Example 1:

```
public void traverse1(Node node) {
  do {
    System.out.println("Node: name=" + node.getNodeName() +
     ", value=" + node.getNodeValue() + ", type=" +
     node.getNodeType());
    if (node.getFirstChild() != null) {
      System.out.println("Processing children:");
      traverse1(node.getFirstChild());
    }
  }  while ((node = node.getNextSibling()) != null);
}
```

Example 2:

```
public void traverse2(Node node) {
  //Get the children of this Node
  NodeList children = node.getChildNodes();

  //go through all the children of the node
  for (int i=0; i<children.getLength(); i++) {
    //get the next child
    Node child = children.item(i);
    //get the type of the child
    short childType = child.getNodeType();
    if (childType == Node.ELEMENT_NODE) {
      //if the child is an Element then print the start and end
      //tags and recurse the content
      String nodeName = child.getNodeName();
      System.out.print("<" + nodeName + ">");
      traverse2(child);
      System.out.print("</" + nodeName + ">");
    } else if (childType == Node.TEXT_NODE) {
      //if the child is a Text node just print the text content
      String data = child.getNodeValue();
      System.out.print(data);
    }
  }
}
```

Example 3:

```
public void traverse3(Node node, int indent) {
  for (int i = 0; i < indent; i++) {
    System.out.print("   ");
  }
  int type = node.getNodeType();
  switch (type) {
  case Node.ATTRIBUTE_NODE:
    System.out.println("ATTRIBUTE_NODE");
    break;
  case Node.CDATA_SECTION_NODE:
    System.out.println("CDATA_SECTION_NODE");
    break;
```

```
    case Node.COMMENT_NODE:
      System.out.println("COMMENT_NODE");
      break;
    case Node.DOCUMENT_FRAGMENT_NODE:
      System.out.println("DOCUMENT_FRAGMENT_NODE");
      break;
    case Node.DOCUMENT_NODE:
      System.out.println("DOCUMENT_NODE");
      break;
    case Node.DOCUMENT_TYPE_NODE:
      System.out.println("DOCUMENT_TYPE_NODE");
      break;
    case Node.ELEMENT_NODE:
      System.out.println("ELEMENT_NODE");
      NamedNodeMap atts = node.getAttributes();
      for (int i = 0; i < atts.getLength(); i++) {
        Node att = atts.item(i);
        traverse3(att, indent + 1);
      }
      break;
    case Node.ENTITY_NODE:
      System.out.println("ENTITY_NODE");
      break;
    case Node.ENTITY_REFERENCE_NODE:
      System.out.println("ENTITY_REFERENCE_NODE");
      break;
    case Node.NOTATION_NODE:
      System.out.println("NOTATION_NODE");
      break;
    case Node.PROCESSING_INSTRUCTION_NODE:
      System.out.println("PROCESSING_INSTRUCTION_NODE");
      break;
    case Node.TEXT_NODE:
      System.out.println("TEXT");
      break;
    default:
      System.out.println("???");
      break;
    }
    for (Node c = node.getFirstChild(); c != null; c =
    c.getNextSibling()) {
      traverse3(c, indent + 1);
    }
  }
```

2.2. Introduction to SAX

SAX uses callback approach to parse an XML document. Thus greater efficiency is achieved when parsing large XML documents. Now we will introduce the most important methods and interfaces, used when processing an XML using SAX. For more information and full description please check the javadoc, or the references at the end of this document.

First a basic description of the interfaces used in the call-back will be provided, and then the ways of setting them to a SAXParser, and parsing XML Files.

2.2.1. org.xml.sax.ContentHandler

This is the main interface that most SAX applications implement. If the application needs to be informed of basic parsing events, it implements this interface and registers an instance with the SAX parser using the `setContentHandler` method. The parser uses the instance to report basic document-related events as the start and end of elements and character data.

- `public void startDocument()` - receives notification of the beginning of a document.
- `public void endDocument()` - receives notification of the end of a document.
- `public void startElement(String namespaceURI, String localName, String qName, Attributes atts)` - receives notification of the beginning of an element. The `namespaceURI`, `localName` and `qName` parameters are only relevant when `namespaceAware` processing is used. For the simple case where a simple XML file is parsed then `namespaceURI` is `null`, and `qName` is equal to `localName` – the name of the XML tag.
- `public void endElement(String namespaceURI, String localName, String qName)` - receives notification at the end of an element.
- `public void characters(char[] ch, int start, int length)` - receives notification of character data.
- `public void processingInstruction(String target, String data)` - receives notification of a processing instruction.

2.2.2. org.xml.sax.Attributes

This interface is used to represent a list of attributes. Each attribute can be accessed in three different ways: by index, by name or by NamespaceQualified name.

- `public int getLength()` – returns the number of attributes in the list.
- `public String getQName(int index)` – looks up an attribute's XML 1.0 qualified name by index.
- `public String getType(int index)` – looks up an attribute's type by index. The attribute type is one of the strings "CDATA", "ID", "IDREF",

"IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in capital letters).

- `public String getValue(int index)` - looks up an attribute's value by index.
- `public int getIndex(String qName)` – looks up the index of an attribute by XML 1.0 qualified name.
- `public String getValue(String qName)` – looks up an attribute's value by XML 1.0 qualified name.
- `public String getType(String qName)` – looks up an attribute's type by XML 1.0 qualified name.

2.2.3. Using SAX Parser

In order to process an XML document using SAX, you have to take SAXParser from a SAXParserFactory and supply it with a source document and an instance of a class, which extends `org.xml.sax.helpers.DefaultHandler`. This class – `DefaultHandler` is a class, which has default, empty, implementations of all the methods of the `ContentHandler` interface (and also - `ErrorHandler`, `DTDHandler`), and itself implements all these interfaces. So if you need to be notified when an element is starting, you just need to extend this class and override the `startElement()` method.

2.2.3.1. javax.xml.parsers.SAXParserFactory

- `public static SAXParserFactory newInstance()` – similar to the `DocumentBuilderFactory.newInstance()`.
- `public SAXParser newSAXParser()` – returns a `SAXParser` instance, which can be used to parse files.
- `public void setNamespaceAware(boolean awareness)` – specifies that the parser produced by this code will provide support for XML namespaces. By default the value is set to `false`.
- `public void setValidating(boolean validating)` – specifies that the parser produced by this code will validate documents as they are parsed. By default the value is set to `false`.

2.2.3.2. javax.xml.parsers.SAXParser

- `public void parse(InputSource is, DefaultHandler h)` – you can also use `InputStream`, `Reader`, `File`, `String`, as the source for the parse.

2.2.4. Examples

2.2.4.1. SAX Parsing

Example 1:

```
public class JAXPSAXExample {
  public static void main(String args[]) {
    try {
      String xml = "data/cars.xml";
      //get a SAXParserFactory from the underlying implementation
      SAXParserFactory factory = SAXParserFactory.newInstance();
      //get a new SAXParser from the factory
      SAXParser parser = factory.newSAXParser();
      //parse the specified file using the SAXTreeStructure
      //handler
      parser.parse(xml, new SAXTreeStructure());
    } catch (Exception e) {
      // handle exception
      e.printStackTrace();
    }
  }
```

2.2.4.2. ContentHandler, DefaultHandler Implementation

Example 1:

```
public class SAXTreeStructure extends DefaultHandler {
   public void startElement(String namespaceURI, String localName,
       String rawName, Attributes atts) {
     System.out.println("Start Element: " + rawName);
   }
   public void endElement(String namespaceURI, String localName,
       String rawName) {
     System.out.println("End Element  : " + rawName);
   }
   public void characters(char []data, int off, int length) {
     System.out.println("Characters   :    " + new String(data,
      off, length));
   }
}
```

Example 2:

```
public class JAXPSAXExample2 extends DefaultHandler {
  public void setDocumentLocator(Locator locator) {
    System.out.println("Method setDocumentLocator(" + locator +
      ")");
  }
  public void startDocument() throws SAXException {
    System.out.println("Method startDocument().");
```

```
}
public void endDocument() throws SAXException {
  System.out.println("Method endDocument().");
}
public void startPrefixMapping(String prefix, String uri) throws
     SAXException {
  System.out.println("Method startPrefixMapping(" + prefix + ",
    " + uri + ").");
}
public void endPrefixMapping(String prefix) throws SAXException{
  System.out.println("Method endPrefixMapping(" + prefix +
    ").");
}
public void startElement(String namespaceURI, String localName,
     String qName, Attributes atts) throws SAXException {
  System.out.println("Method startElement(" + namespaceURI + ",
     " + localName + ", " + qName + ", " + atts + ").");
}
public void endElement(String namespaceURI, String localName,
     String qName) throws SAXException {
  System.out.println("Method endElement(" + namespaceURI + ", "
    + localName + ", " + qName + ").");
}
public void characters(char[] ch, int start, int length) throws
    SAXException {
  System.out.println("Method characters(" + new String(ch,
    start, start + length) + ").");
}
public void ignorableWhitespace(char[] ch, int start, int
                                   length) throws SAXException {
  System.out.println("Method ignorableWhitespace(" + new
    String(ch, start, start + length) + ").");
}
public void processingInstruction(String target, String data)
     throws SAXException {
  System.out.println("Method processingInstruction(" + target +
     ", " + data + ").");
}
public void skippedEntity(String name) throws SAXException {
  System.out.println("Method skippedEntity(" + name + ").");
}
public InputSource resolveEntity(String publicId, String
                                  systemId) throws SAXException {
  System.out.println("Method resolveEntity(" + publicId + ", " +
    systemId + "). Resolving to: data\\" + systemId);
  return new InputSource("data/" + systemId);
}

public static void main(String[] args) throws Exception {
  String xml = "data/rich_ii.xml";
  SAXParserFactory factory = SAXParserFactory.newInstance();
  factory.setValidating(true);
  SAXParser parser = factory.newSAXParser();
  System.out.println(" validating:      " +
    parser.isValidating());
```

```
    System.out.println(" namespace-aware: " +
      parser.isNamespaceAware());
    System.out.println("");
    parser.parse(xml, new JAXPSAXExample2());
  }
}
```

2.3. XSL Transformations

You can use the XSL Transformation part of JAXP, to transform source documents into result documents, by applying a stylesheet or just convert from one type of source (for example DOM), to some different kind of result (for example SAX). The source and result will represent one and the same document. You could also apply indenting to a document, by giving it as a source document, setting the transformer an `OutputProperty` to use indenting, and then set the result to be some other file.

2.3.1. Source Types

2.3.1.1. javax.xml.transform.Source

This is the base class for all kinds of sources.

2.3.1.2. javax.xml.transform.stream.StreamSource

This class represents a source, which is being read from a stream. This stream could be an `InputStream`, `Reader`, `File`, `URL`. You just have to use the correct constructor: `StreamSource(InputStream)`, `StreamSource(Reader)`, `StreamSource(File)` or `StreamSource(String url)`. Keep in mind that when reading from a stream, it is not a duty of the transformer to close this stream. Only when reading from a `File` or `url`, the parser opens a stream to this resource and takes care to close it at the end.

2.3.1.3. javax.xml.transform.dom.DOMSource

This class represents a source, which is either a DOM `Document` (`org.w3c.dom.Document`), or any other kind of node. The parser uses this tree as the source document. The general constructor is `DOMSource(Node node);`

2.3.1.4. javax.xml.transform.sax.SAXSource

This class represents a source, which is a SAX `DefaultHandler`. That is, the parser reads the call-backs (called on a `DefaultHandler` implementation, by

some other parser) and constructs its source data. You might use the constructor – `SAXSource(DefaultHandler handler);`

2.3.2. Result Types

2.3.2.1. javax.xml.transform.Result

This is the base class for all kinds of results

2.3.2.2. javax.xml.transform.stream.StreamResult

Similarly to `StreamSource`, this class is used when you want the result to be written in a stream (`OutputStream`, `Writer`, `File`). The constructors are: `StreamResult(OutputStream), StreamResult(Writer), StreamResult(File), StreamResult(String url).` Keep in mind that it is not a responsibility of the transformer to close a `StreamResult` constructed with `OutputStream` or `Writer`.

2.3.2.3. javax.xml.transformer.dom.DOMResult

Using this kind of result you might specify that you want the result to be a DOM Tree, which will be built by the transformer. The constructor is `DOMResult(Node root).`

2.3.2.4. javax.xml.transformer.sax.SAXResult

Using this result you might supply an instance of a class extending `DefaultHandler`, which methods will be called in order to let the user create the result himself. The constructor is `SAXResult(DefaultHandler handler).`

2.3.3. Output Properties

These properties can be set on the transformer to change some properties like indent, xml-declaration, type (XML, HTML, text), etc. of the result. They are the same that can be applied to a transformation when using the xsl:output – node in the stylesheet. Here only the most important ones will be lined out. For further information check the javadoc documentation of the XML Toolkit, or Section 16 of the XSLT Documentation.

Usage

These are predefined strings in the class `javax.xml.transform.OutputKeys`, which you might set to the transformer using: `transformer.setOutputProperty(String name, String value),` on an instance of the transformer.

- `OutputKeys.VERSION` – to control the version of the result document
- `OutputKeys.METHOD` – can be xml, html, text or some namespace qualified name.
- `OutputKeys.ENCODING` – sets the encoding of the result. It is very useful when you want to transform one source document in UTF-8 to ISO-8859-1, or some other encoding.
- `OutputKeys.DOCTYPE_PUBLIC` and `OutputKeys.DOCTYPE_SYSTEM` – adds a `<!DOCTYPE (root-tag) SYSTEM (System_key) (Public_key)>` to the result.
- `OutputKeys.INDENT` – "yes" or "no". Allows you to control, weather the result should be indented or not. In this way you might add indenting to some XML file, just by transforming it from a `StreamSource` to a `StreamResult`, and setting this `OutputProperty` to "yes".

2.3.4. Transforming without Stylesheet – Converting

To transform from one kind of source to some other kind of result you must get an instance of `TransformerFactory` and then use `newTransformer()` to get a transformer, which is not associated with a stylesheet:

```
TransformerFactory factory = TransformerFactory.newInstance();
Transformer transformer = factory.newTransformer();
```

Now you might, optionally, set some `OutputProperties` to the transformer:

```
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
```

And finally do a transformation:

1.Stream -> Stream (Changing the indent)

```
transformer.transform(new StreamSource("source.xml"), new
StreamResult("Result.xml");
```

2.Stream -> DOM (also known as DOM Parsing)

```
transformer.transform( new StreamSoruce("source.xml"), new
DOMResult(dom_node));
```

3.DOM -> SAX

```
transformer.transform( new DOMSource(dom_node), new
SAXResult(sax_defaulthandler));
```

4.Check the examples in the SAP J2EE XML Toolkit for more.

2.3.5. Transforming without Stylesheet

This procedure is almost the same as was described in 2.3.4. The difference is that when getting the transformer instance with `newTransformer`, another overload is used. That is `newTransformer`(`Source` source):

```
Transformer transformer = factory.newTransformer( new StreamSource
( new FileReader("mystylesheet.xsl"));
```

2.3.6. Examples

2.3.6.1. Converting Example

```
public class JAXPSourceConvertExample {
  public static void main(String[] args) throws Exception {
    String xml = "data/current.xml";
    // Obtaining an instance of the factory
    TransformerFactory factory = TransformerFactory.newInstance();
    Transformer transformer = factory.newTransformer();
    DOMResult domResult = new DOMResult();

    // stream -> dom
    System.out.println("\nStream -> DOM ");
    transformer.transform(new StreamSource(xml), domResult);

    // sax -> stream
    System.out.println("\nSAX -> Stream ");
    transformer.transform(new SAXSource(new InputSource(xml)), new
      StreamResult(System.out));

    // dom -> stream
    System.out.println("\nDOM -> Stream");
    transformer.transform(new DOMSource(domResult.getNode()), new
      StreamResult(System.out));

    // stream -> sax
    System.out.println("\nStream -> SAX");
    transformer.transform(new StreamSource(xml), new
      SAXResult((ContentHandler)new SAXTreeStructure()));

    // sax -> dom
    domResult = new DOMResult();
    System.out.println("\nSAX -> DOM");
    transformer.transform(new SAXSource(new InputSource(xml)),
      domResult);

    // dom -> sax
    System.out.println("\nDOM -> SAX");
    transformer.transform(new DOMSource(domResult.getNode()), new
      SAXResult((ContentHandler)new SAXTreeStructure()));
  }
}
```

2.3.6.2. XSL Transformation

```
public class JAXPXSLTExample {
  public static void main(String args[]) {
    //get a new TransformerFactory from the underlying
    //implementation
    TransformerFactory factory = TransformerFactory.newInstance();
    Transformer transformer = null;
    Templates template = null;
    StreamSource xmlSource = null;
    StreamSource xslSource = null;
    StreamResult result = null;
    try {
      // a new StreamSource from a file name
      xmlSource = new StreamSource("data/cars.xml");
      xslSource = new StreamSource("data/cars1.xsl");
      // a new StreamResult to a file name (it is automatically
      //closed on exit)
      result    = new StreamResult("data/cars1.html");
      // get a new Templates object for this stylesheet
      template = factory.newTemplates(xslSource);
      // get a new Transformer from the Templates
      transformer = template.newTransformer();
      // perform the transformation
      transformer.transform(xmlSource, result);
      // it is not needed to close the Stream result since it is
      //output to a file name
    }
  }
}
```

3. Advanced Techniques

3.1. Integration of XSL with Other Languages

3.1.1. Java Integration (Java Language Binding)

Allows method invocation upon existing Java classes and objects (i.e. invocation of static and instance methods) and object creation (constructor invocation) upon existing classes.

To implement an extension function in Java, stylesheet developers associate an implementation to their extension function namespace prefix using an `xsl:script` element with `language="java"` as in the following example:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0"
xmlns:date="uri.any">

<xsl:script implements-prefix="date" language="java"
src="java:com.example.datestuff.DateRoutines"/>

<xsl:template match="/">
<OrderDate>
<xsl:value-of select="date:format(/order/date,'MM/DD/YY')"/>
</OrderDate>
</xsl:template>
</xsl:stylesheet>
```

For Java, the value of the `src` attribute on `xsl:script` is a URI reference identifying a Java class. The methods of this class implement the functions, which expanded names, have the namespace URI specified by the `implements-prefix` attribute. The URI specified by the `src` attribute may use the `java:` URI scheme: `src="java:fully.qualified.ClassName"`; the XSLT processor may also allow the URI to use other URI schemes to locate a resource containing the Java class file.

Here is an example XSL document that uses Java Language Binding:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:myobj="com.inqmy.lib.xml.parser.Myobj">
```

```
<xsl:param name="myobj"/>

<xsl:template match = "/">
   At this moment:<xsl:value-of
select="string(myobj:getToday())"/>
   we offer:
   <xsl:apply-templates/>

   <xsl:value-of select="myobj:callNodeList($myobj, /)"/>

</xsl:template>
</xsl:stylesheet>
```

Applying the latter to the following XML document:

```
<?xml version="1.0"?>
<grocery name = "The red tomato">
  <fruit>
    <article> Apples</article>
     <article> Pears </article>
     <article> Bananas </article>
  </fruit>
  <vegetables>
    <article>Cucumbers </article>
    <article> Tomatoes </article>
     <article>  Peppers </article>
  </vegetables>
  </grocery>
```

Results in:

```
<?xml version='1.0' encoding='utf-8'?>
   At this moment:Wed Oct 17 17:48:53 EEST 2001
   we offer:

    Apples

    Pears

    Bananas

    Cucumbers

    Tomatoes

     Peppers
```

In order to run this example, a parameter should be added to the transformer, prior to transformation in the following way:

```
Myobj my = new Myobj();
transformer.setParameter("myobj", my);
```

Where the class definition of the class `Myobj` looks like:

```
import org.w3c.dom.*;

public class Myobj {

  public void callNodeList(Node n) {
    System.out.println(n);
  }

  public static java.util.Date getToday() {
    java.util.Calendar cal =
ava.util.Calendar.getInstance();
    return cal.getTime();

  }

}
```

3.1.2. Integration with Scripting Languages

Allows execution of scripts in virtually any scripting language that offers a scripting engine, compatible with the Bean Scripting Framework (BSF). Also allows function calls for the functions in the script. Short example is described briefly below:

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:my-ext="ext1"
    extension-element-prefixes="my-ext">

  <xsl:script language="javascript" implements-prefix="my-ext">
      var multiplier=2;

      function timelapse() {return 4;}
      function getdate(numdays)
      {
        var d = new Date();
        var totalDays = parseInt(numdays) * multiplier;
        d.setDate(d.getDate() + totalDays);
        return d.toLocaleString();
      }
  </xsl:script>


  <xsl:template match="deadline">
    <p><my-ext:timelapse multiplier="4"/>We have logged your
enquiry and will
      respond by <xsl:value-of select="string(my-
ext:getdate(string(@numdays)))"/>.</p>
```

```
  </xsl:template>

</xsl:stylesheet>
```

The script has to be situated in a top-level `<xsl:script>` element, with its language specified as an attribute.

3.2. Multiple Output Documents

The `xsl:document` element is used to create multiple result documents. As well as the main result document, subsidiary result documents may be available. Each subsidiary result document is created using an `xsl:document` element. The content of the `xsl:document` element is a template; this is instantiated to create a sequence of nodes; a root node is created with this sequence of nodes as its children; the tree with this root node represents the subsidiary result document. The `href` attribute specifies where the subsidiary document should be stored; it must be an absolute or relative URI; it must not have a fragment identifier.

For instance the following XML file:

```
<?xml version="1.0"?>
<grocery name = "The red tomato">
  <fruit>
    <article>Apples </article>
                <article>Pears </article>
                <article>Bananas</article>
  </fruit>
  <vegetables>
    <article>Cucumbers</article>
                <article>Tomatoes</article>
                <article>Peppers</article>
  </vegetables>
            </grocery>
```

will be processed with the following stylesheet:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <main_output>
      Vegetables and Fruit
      <xsl:apply-templates/>
    </main_output>
  </xsl:template>

  <xsl:template match="grocery">
    <In_stock>
      <xsl:apply-templates/>
```

```
        <fruit>
         <xsl:value-of select =
'document("fruit\fruit.out")/fruit'>
         </xsl:value-of>
        </fruit>
        <vegetables>
         <xsl:value-of select =
'document("vegetables\vegetables.out")/vegetables'>
         </xsl:value-of>
        </vegetables>

     </In_stock>
   </xsl:template>

    <xsl:template match = "/grocery/fruit">
      <xsl:document href = "fruit\fruit.out">
        <fruit>
         <xsl:apply-templates/>
        </fruit>
      </xsl:document>
    </xsl:template>

<xsl:template match = "/grocery/vegetables">
      <xsl:document  href = "vegetables\vegetables.out">
        <vegetables>
          <xsl:apply-templates/>
        </vegetables>
      </xsl:document>
    </xsl:template>

</xsl:stylesheet>
```

Then two files will be created - *fruit*|*fruit.out* and |*vegetables*|*vegetables.out*, containing the fruit and vegetables in stock, respectively. The combined information will be stored in the main output.

3.3. Active XML Scanning

The JAXRPC Community draft version 0.5 specifies an interface called `XMLReader`. There is an interface called ActiveXMLParser, which provides a similar functionality and is located in the `com.inqmy.lib.xml.parser` package of the SAP XML Toolkit for Java. Although not all of the methods correspond exactly to those, specified in the draft, the purpose of this interface is the same – it specifies the behavior of the class, responsible for carrying out the step-by-step parsing process. The interface specifies six specific states in which the parser can be: initial state, start of element, end of element, character data, processing instruction and end of file. These are mapped to the numbers from 0 to 5 respectively. The methods of the interface allow the

user some sort of manual control over the process – the user has the freedom (and responsibility) to notify the parser to proceed to the next stage of the parsing process. Left "on his own", the parser never does a forward step.

As an example of the features, described above, one could consult the *XMLReaderExapmle.java* file, available in the *<SAPj2eeEngine_install_dir>/docs/examples/xml_toolkit* directory coming with the SAP XML Toolkit for Java (it uses a specific implementation of the described interface, that is located in the `com.inqmy.lib.xml.parser.ActiveXMLParser` package.)

3.4. Evaluating XPath Queries on DOM Objects

The XPath 1.0 recommendation (available at *http://www.w3.org/TR/xpath*) stated: "XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer [XPointer]." This confines XPath to a very specific area of use. It has become obvious, though, that XPath could be useful for a wider area of purposes, including DOM. This problem has been addressed to by W3C in the DOM level3 XPath Specification (available at *http://www.w3.org/TR/2001/WD-DOM-Level-3-XPath-20011031/*.) The functionality described in this document is implemented in the SAP XML Toolkit for Java – it provides you with the opportunity to create XPath queries and evaluate them with respect to different context.

As an example of the features, described above, one could consult the *DomXpathExapmle.java* file, available in the *<SAPj2eeEngine_install_dir>/docs/examples/xml_toolkit* directory coming with the SAP XML Toolkit for Java (it uses a specific implementation of the described interface, that is located in the `com.inqmy.lib.xml.dom.xpath` package.)

3.5. Changing the Default Parser

In order to change the default parser and set a different one, replace the following code fragment in the *<SAPj2eeEngine_install_dir>/alone/managers/library.txt* or *<SAPj2eeEngine_install_dir>/cluster/server/managers/library.txt*:

`library inqmyxml inqmyxml.jar` with `library inqmyxml newparser_jar_files`, where the `newparser_jar_files` represents a JAR files

list of the new parser. Also the following line must be added to the *library.txt* file:

```
reference core server-xml
```

Note: This section implies for the default parser as a part of the SAP J2EE Engine. Keep in mind that the new parser must be obligatory JAXP compliant in order to function properly.

4. SAP XML Toolkit for Java FAQ

4.1. Frequently Asked Questions (FAQ)

4.1.1. Questions

1. How can I build a DOM tree from an XML document?

2. How can I use a SAX -> XML Serializer?

3. How to match an XPath expression to an XML document, for getting a boolean result?

4. Using the Toolkit you receive a strange exception:

```
java.lang.NoSuchMethodError at
com.inqmy.lib.xml.util.DOMToDocHandler.process(DOMToDocHandler.jav
:50)
at
com.inqmy.lib.xml.util.DOMToDocHandler.process(DOMToDocHandler.jav
a:113)
at
com.inqmy.lib.xml.util.DOMToDocHandler.process(DOMToDocHandler.jav
a:37)
at
```

4.1.2. Answers

1. Please, consider with the `javax.xml.parsers.*` JAXP package documentation.

2. Since SAX is object-oriented, the class providing for this serialization should implement the following SAX interfaces: `EntityResolver`, `DTDHandler`, `ContentHandler` and `ErrorHandler`. You can use the provided `com.inqmy.lib.xml.util.SAXToOutputStreamHandler` class. It implements `org.xml.sax.ContentHandler`, `org.xml.sax.DTDHandler`, `org.xml.sax.ErrorHandler`, `org.xml.sax.ext.DeclHandler` and `org.xml.sax.ext.LexixalHandler`. To use this class, instantiate it through one of the provided constructors: `SAXToOutputStreamHandler(String filename)` – creates a new file and an XML, serialized to it. The file is closed at the end; `SAXToOutputStreamHandler(OutputStream stream)` – the XML is serialized to this stream, which is not closed at the end;

`SAXToOutputStreamHandler(Writer writer)` – the XML is serialized to this `Writer`, which is not closed at the end.
Example:

```
SAXToOutputStreamHandler handler = new
SAXToOutputStreamHandler("d:/temp/xml.out");
SAXParser sax = new SAXParser();
sax.setContentHandler(handler);
handler.parse("d:/temp/data.xml");
```

3. Using JAXP 1.1 cannot easily perform this process. There is a class, which encapsulates the usage of our `XPathProcessor` as a stand-alone class, leaving the user distanced from the implementation. This class is very useful and provides a great functionality, which can be extended including any other suggestion.
   Class:`com.inqmy.lib.xml.xsl.xpath.XpathMatcher`
   Constructor: `XPathMatcher(String filename) throws FileNotFoundException, XpathException.`

   This constructs a new `XPathMatcher` object, and initializes it with the specified file. If the file cannot be found then a `FileNotFoundException` is thrown. If there is any other error then an `XPathException` (`com.inqmy.lib.xml.xsl.xpath.XPathException`) is thrown. Also this exception is thrown when invoking the XML through any other method.
   Methods:
   `init(String filename)` – initializes an existing `XPathMatcher` instance.
   `match(String query)` – matches this query with the current context (by default, this is the root node) and returns `true` or `false`.

   An `XnodeSet` result – returns `false` if the node set is empty, else `true`.
   An `XString` result – returns `false` if the string is empty, else `true`.
   An `XNumber` result – returns `false` if the number is 0, else `true`.
   An `XBoolean` result – it is clear.

   Advanced Use

   The following methods provide advanced functionality. To use them, check javadoc (Java code documentation) first, to understand the class purposes.

- `process(String query)` – processes this query and returns a `XObject`. See the javadoc for more details.
- `setContext(String query)` - sets a current context. For example "/" sets the root node context, "/table" sets the first table node context,

"/table[name = 'Bob']" sets the first table node context, which has 'name' child with 'Bob' value. In this case you can set the current context to any node and then to match the query, using this context.

- `setContextNode(int node)` – changes the context Node – see DTM and `XpathContext`.
- `setContextSize(int node)` - changes the context size.
- `setContextPosition(int pos)` – changes the context position.

4. You have several packages in your classpath, which include the `org.xml.sax` or `org.w3c.dom`, interfaces and the classloader loaded the older version. This usually happens when you try to use Apache (Tomcat) and SAP J2EE Engine. In this case you have either to move SAP J2EE Engine in front of apaches (*xerces.jar*, *xalan.jar*) in the classpath, to remove the other parser (it might be any other), or just to download new interfaces (for example 1.4.3 version of Xerces)

5. References

1. W3C Recommendation "XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2 May 2001. (See *http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/*.)

2. W3C Recommendation "XML Schema Part 2: Datatypes", Paul V. Biron, Ashok Malhotra, 2 May 2001. (See *http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/*.)

3. W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See *http://www.w3.org/TR/1999/REC-xml-names-19990114/*.)

4. W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)", Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 6 October 2000. (See *http://www.w3.org/TR/2000/REC-xml-20001006/.)*

5. Java API for XML Parsing - *http://java.sun.com/xml/jaxp/index.html*

6. DOM 2 Specification - *http://www.w3.org/DOM/*