3. XML Processor APIs

How can (Java) applications manipulate structured (XML) documents?
   – An overview of XML processor interfaces

3.1 SAX: an event-based interface
3.2 DOM: an object-based interface
3.3 JAXP: Java API for XML Processing

Tasks of a Parser

- Document instance decomposition
  - elements, attributes, text, processing instructions, entities, ...
- Verification
  - well-formedness checking
    » syntactical correctness of XML markup
  - validation (against a DTD or Schema)
- Access to contents of the DTD (if supported)
  - SAX 2.0 Extensions provide info of declarations:
    element type names and their content model expressions

I: Event-based interfaces
   - Command line and ESIS interfaces
   - Element Structure Information Set, traditional interface to stand-alone SGML parsers
   - Event call-back interfaces: SAX

II: Tree-based (object model) interfaces
   - W3C DOM Recommendation
   - Java-specific object models: JAXB, JDOM, dom4J

Command-line ESIS interface

Application

ESIS Stream

SGML/XML Parser

<E l="1"> Hi! </E>

Event Call-Back Interfaces

- Application implements a set of call-back methods for handling parse events
  - parser notifies the application by method calls
  - parameters qualify events further, with
    » element type name
    » names and values of attributes
    » values of content strings, ...
- Idea behind "SAX" (Simple API for XML)
  - an industry standard API for XML parsers
  - could think as "Serial Access XML"
**An event call-back application**

```
startDocument()
startElement()
characters()
endElement()
```

**Object Model Interfaces**

- Application interacts with
  - a parser object
  - a document object consisting of objects for `document`, `elements`, `attributes`, `text`, …
- Abstraction level higher than in event based interfaces; more powerful access
  - to descendants, following siblings, …
- Drawback: Higher memory consumption
  - used mainly in client applications
  (to implement document manipulation by user)

**An Object-Model Based Application**

```
<?xml version='1.0'?><A i="1">Hi!</A>
```

**3.1 The SAX Event Callback API**

- A de-facto industry standard
  - Developed by members of the xml-dev mailing list
  - Version 1.0 in May 1998, Vers. 2.0 in May 2000
  - Not a parser, but a common interface for many different parsers (like, say, JDBC is a common interface to various RDBs)
- Supported directly by major XML parsers
  - most Java based and free:
    - Apache Xerces, Oracle's XML Parser for Java;
    - MSXML (in IE 5), James Clark's XP

**SAX 2.0 Interfaces**

- Interplay between an application and a SAX-conformant parser specified in terms of interfaces (i.e., collections of methods)
- One way to classify SAX interfaces:
  - Application-to-parser interfaces
    - to use the parser
  - Parser-to-application (or call-back) interfaces
    - to act on various parsing events
  - Auxiliary interfaces
    - to manipulate parser-provided information

**Application-to-Parser Interfaces**

- Implemented by `parser` (or a SAX driver):
  - `XMLReader`
    - methods to invoke the parser and to register objects that implement call-back interfaces
  - `XMLFilter` (extends `XMLReader`)
    - interface to connect `XMLReaders` in a row as a sequence of filters
    - obtains events from an `XMLReader` and passes them further (possibly modified)
Call-Back Interfaces

- Implemented by application to override default behaviour (of ignoring events quietly)
  - ContentHandler
    - methods to process document parsing events
  - DTDHandler
    - methods to receive notification of unparsed external entities and their notations declared in the DTD
  - ErrorHandler
    - methods for handling parsing errors and warnings
  - EntityResolver
    - methods for customised processing of external entity references

SAX 2.0: Auxiliary Interfaces

- Attributes
  - methods to access a list of attributes, e.g:
    - int getValue(String attrName)
- Locator
  - methods for locating the origin of parse events (e.g systemID, line and column numbers, say, for reporting semantic errors controlled by the application)

The ContentHandler Interface

- Information of general document events. (See API documentation for a complete list):
  - setDocumentLocator(Locator locator)
    - Receive a locator for the origin of SAX document events
  - startDocument(); endDocument()
    - notify the beginning/end of a document.
  - startElement(String nsURI, String localName, String rawName, Attributes atts);
  - endElement( ... ); similar (without attributes)

Namespaces in SAX: Example

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/xhtml1/strict">
    <xsl:template match="/">
      <html>
        <xsl:value-of select="//total"/>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

Namespaces: Example (2)

```xml
<xsl:stylesheet version="1.0"
    xmlns="http://www.w3.org/TR/xhtml1/strict">
    <xsl:template match="/">
      <html> ... </html>
    </xsl:template>
</xsl:stylesheet>
```

- startElement for this would pass following parameters:
  - nsURI = http://www.w3.org/1999/XSL/Transform
  - localname = template, rawName = xsl:template

- endElement for html would give
  - nsURI = http://www.w3.org/TR/xhtml1/strict
  - localname = html, rawName = html

ContentHandler interface (cont.)

- characters(char ch[], int start, int length)
  - notification of character data
- ignorableWhitespace(char ch[], int start, int length)
  - notification of ignorable whitespace in element content

```xml
<xdoc:doc>
  <xdoc:doctype>
    <xdoc:template>
      <xdoc:element>
        <xdoc:element/>
      </xdoc:template>
      <xdoc:template>
        <xdoc:element>
          <xdoc:element>
            <xdoc:element/>
          </xdoc:element>
        </xdoc:element>
      </xdoc:template>
      <xdoc:template>
        <xdoc:element>
          <xdoc:element/>
        </xdoc:element>
      </xdoc:template>
    </xdoc:template>
  </xdoc:doctype>
</xdoc:doc>
```
SAX Processing Example (1/9)

**Input:** XML representation of a personnel database:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<db>
  <person idnum="1234">
    <last>Kilpeläinen</last><first>Pekka</first>
  </person>
  <person idnum="5678">
    <last>Möttönen</last><first>Matti</first>
  </person>
  <person idnum="9012">
    <last>Möttönen</last><first>Maija</first>
  </person>
  <person idnum="3456">
    <last>Römppänen</last><first>Maija</first>
  </person>
</db>
```

SAX Processing Example (2/9)

**Task:** Format the document as a list like this:

- Pekka Kilpeläinen (1234)
- Matti Möttönen (5678)
- Maija Möttönen (9012)
- Maija Römppänen (3456)

**Event-based processing strategy:**
- At the start of `person`, record the `idnum` (e.g., 1234)
- Record starts and ends of `last` and `first` to store their contents (e.g., "Kilpeläinen" and "Pekka")
- At the end of a `person`, output the collected data

SAX Processing Example (3/9)

**Application:** Begin by importing relevant classes:

```java
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
// Default (no-op) implementation of interface ContentHandler:
import org.xml.sax.helpers.DefaultHandler;
// SUN JAXP to instantiate a parser:
import javax.xml.parsers.*;
```

SAX Processing Example (4/9)

**Implement relevant call-back methods:**

```java
public class SAXDBApp extends DefaultHandler{
  // Flags to remember element context:
  private boolean InFirst = false,
  InLast = false;
  // Storage for element contents and attribute values:
  private String FirstName, LastName, IdNum;
  // Callback methods:
  // Record the start of first and last elements, and the idnum attribute of a person:
  public void startElement (String nsURI, String localName, String rawName, Attributes atts) {
    if (rawName.equals("person"))
      IdNum = atts.getValue("idnum");
    if (rawName.equals("first"))
      InFirst = true;
    if (rawName.equals("last"))
      InLast = true;
  } // startElement
```

SAX Processing Example (5/9)

**Callback methods:**
- Record the start of first and last elements, and the idnum attribute of a person:

```java
  public void characters (char buf[], int start, int length) {
    if (InFirst) FirstName = new String(buf, start, length);
    if (InLast) LastName = new String(buf, start, length);
  } // characters
```

SAX Processing Example (6/9)

**Callback methods continue:**
- Record the text content of elements first and last:

```java
  public void characters (char buf[], int start, int length) {
    if (InFirst) FirstName = new String(buf, start, length);
    if (InLast) LastName = new String(buf, start, length);
  } // characters
```
At the end of `person`, output the collected data:

```java
public void endElement(String nsURI, String localName, String qName) {
    if (qName.equals("person")) {
        System.out.println(FirstName + " " + LastName + " (" + IdNum + ")");
        // Update the context flags:
        if (qName.equals("first"))
            InFirst = false;
        // (and the same for "last" and InLast)
    }
}
```

Application `main` method:

```java
public static void main (String args[])
throws Exception {
    // Instantiate an XMLReader (from JAXP
    // SAXParserFactory):
    SAXParserFactory spf = SAXParserFactory.newInstance();
    try {
        SAXParser saxParser = spf.newSAXParser();
        XMLReader xmlReader = saxParser.getXMLReader();
        // Instantiate and pass a new
        // ContentHandler to xmlReader:
        ContentHandler handler = new SAXDBApp();
        xmlReader.setContentHandler(handler);
        for (int i = 0; i < args.length; i++) {
            xmlReader.parse(args[i]);
        }
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

Main method continues:

```java
// Instantiate and pass a new
// ContentHandler to xmlReader:
    ContentHandler handler = new SAXDBApp();
    xmlReader.setContentHandler(handler);
    for (int i = 0; i < args.length; i++) {
        xmlReader.parse(args[i]);
    }
}
```