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

Command-line ESIS interface

<table>
<thead>
<tr>
<th>Application</th>
<th>ESIS Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command line call</td>
<td>SGML/XML Parser</td>
</tr>
</tbody>
</table>

An event call-back application

<table>
<thead>
<tr>
<th>Application Main Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDocument()</td>
</tr>
<tr>
<td>&quot;A&quot;, (i = &quot;1&quot;)</td>
</tr>
<tr>
<td>startElement()</td>
</tr>
<tr>
<td>&quot;hi&quot;</td>
</tr>
<tr>
<td>characters()</td>
</tr>
<tr>
<td>endElement()</td>
</tr>
<tr>
<td>Parse()</td>
</tr>
</tbody>
</table>

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"

Object Model Interfaces

- Application interacts with
  - a parser object, which builds ...
  - a document object consisting of 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)
**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 different parsers (i.e., say, JDBC is a common interface to various RDBs)
  - Supported directly by many XML parsers
    - many Java based, and free; Examples: Apache Xerces, Oracle's XML Parser for Java; MSXML (in IE 5), James Clark's XP

**My classification of SAX 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)

**Auxiliary interfaces**

- **Attributes**
  - methods to access a list of attributes, e.g: `int getLength();` `String 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)

**Namespaces in SAX: Example**

```
<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="/"/>
  <xsl:template match="/html">
    <![CDATA[<xsl:variable name="total" select="//total"]>
</xsl:template>
</xsl:stylesheet>
```

**StartElement** for this would pass following parameters:

- `nsURI` http://www.w3.org/1999/XSL/Transform
- `localName` template, `rawName` xsl:template

**SAX 2.0: Auxiliary Interfaces**

- **Attributes**
  - methods to access a list of attributes, e.g: `int getLength();` `String 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)
Namespaces: Example (2)

```xml
<?xml version="1.0" ... encoding="ISO-8859-1T"?>
<xml:stylesheet version="1.0" ... xml:namespace="http://www.w3.org/TR/xhtml1/strict">
  <html> ... </html>
</xml:stylesheet>

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

SAX Processing Example (1/9)

- Input: XML representation of a personnel database:

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

SAX Processing Example (2/9)

- Task: Format the document as a list like this:
  - Peeka Kilpeläinen (1234)
  - Matti Mottonen (5678)
  - Maija Romppänen (9012)

- Event-based processing strategy:
  - at the start of a 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:
  - First import relevant interfaces & 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;
    // 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;
  }
  ```

SAX Processing Example (5/9)

- Call-back methods:
  - record the start of first and last elements,
    and the idnum attribute of a person
    ```java
    public void startElement{
      String nsURI, String localName, String rawName, Attributes attrs) {
        if (rawName.equals("person")
            IdNum = attrs.getValue("idnum");
        if (rawName.equals("first")
            InFirst = true;
        if (rawName.equals("last")
            InLast = true;
        } // startElement
    }
    ```

SAX Processing Example (6/9)

- Call-back 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
    ```
**SAX Processing Example (7/9)**

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)
    }
}
```

**SAX Processing Example (8/9)**

Application main method:

```java
public static void main (String args[])
    throws Exception {
    // Instantiate an XMLReader [from JAXP]
    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]);
        }
        // main
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

**SAX Processing Example (9/9)**

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]);
}
```

**SAX: Summary**

- A low-level parser-interface for XML documents
- Reports document parsing events through method call-backs
  - > efficient; does not create in-memory representation of the document
  - > used often on servers, and to process LARGE documents