3. XML Processor APIs

How can applications manipulate structured documents?
- An overview of document parser 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
  - not always supported
  - SAX 2.0 Extensions provide info of declarations to applications

Document Parser Interfaces

I: Event-based interfaces
- Command line and ESIS interfaces
  - Command line and ESIS interfaces
  - Command line and ESIS interfaces
- Event call-back interfaces: SAX

II: Tree-based (object model) interfaces
- W3C DOM Recommendation

Event Call-Back Interfaces

- Application implements a set of callback methods for handling parse events
  - parser notifies the application by method calls
  - method parameters qualify events further
    - 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

Application Main Routine
- startDocument()
- startElement()
- characters()
- endElement()

Parse()

Object Model Interfaces

- Application interacts with an object-oriented representation of
  - the parser
  - the document parse tree consisting of objects like document, element, attribute, text, ...
- Abstraction level higher than in event based interfaces; more powerful access
  - to descendants, following siblings, ...
- Drawback: Higher memory consumption

Object Model Based Application

SAX 2.0 Interfaces

- Interplay between an application and a SAX-conformant parser specified in terms of interfaces (i.e., collections of methods)
- Classification of SAX interfaces:
  - Parser-to-application (or call-back) interfaces
    - to attach special behaviour to parser-generated events
  - Application-to-parser
    - to use the parser
  - Auxiliary
    - to manipulate parser-provided information

3.1 The SAX Event Callback API

- A de-facto industry standard
  - Not an official standard or W3C Recommendation
  - 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 JDBC is a common interface to various RDBs)
- Supported directly by major XML parsers
  - most Java based and free: Sun JAXP, IBM XML4J, Oracle’s XML Parser for Java, Apache Xerces; MSXML (in IE 5), James Clark’s XP

Call-Back Interfaces

- Implemented by application to override default behaviour (of ignoring any event 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
Application-to-Parser Interfaces

- Implemented by parser (or its SAX driver):
  - XMLReader
    - methods to allow the application to invoke the parser and
      to register objects that implement call-back interfaces
  - XMLFilter
    - interface to connect several XMLReaders in a row as a
      sequence of filters

Attributes
- methods to access a list of attributes

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)

SAX 2.0: Auxiliary Interfaces

Attributes
- methods to access a list of attributes

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

- Methods for receiving information of general document events. (See API documentation for a complete list):
  - setDocumentLocator(Locator locator)
    - Receive an object for locating the origin of SAX document events (e.g. for reporting semantic errors controlled by the application)
  - startDocument(); endDocument()
    - notification of the beginning/end of a document.
  - startElement(String namespaceURI, String localName, String qName, Attributes atts);
    - similar (without attributes)
  - parameters support namespaces.

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>
```

A SAX invocation of `startElement` for `xsl:template` would pass following parameters:
- namespaceURI = http://www.w3.org/1999/XSL/Transform
- localname = template, qName = xsl:template

Namespaces: Example (2)

```xml
<!DOCTYPE A [
<!ELEMENT A (B)>
<!ELEMENT B (#PCDATA)> ]>
<A>
  <B>
    Ignorable whitespace
  </B>
</A>
```

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)**

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

```xml
<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)**

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

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

- **Solution strategy using event-based processing:**
  - At the start of a `person`, record the `idnum` (e.g., 1234).
  - Keep track of starts and ends of elements `last` and `first`, in order to record content of those elements (e.g., "Kilpeläinen" and "Pekka").
  - At the end of each `person`, output the collected data.

**SAX Processing Example (3)**

- **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 used to obtain a SAX parser:
import javax.xml.parsers.*;
```

**SAX Processing Example (4)**

- **Define a class to implement relevant call-back methods:**

```java
public class SAXDBApp extends DefaultHandler{
  private boolean InFirst = false,
  InLast = false;
  private String FirstName, LastName, IdNum;
  public void startElement (String namespaceURI, String localName,
  String rawName, Attributes atts) {
    if (localName.equals("first"))
      InFirst = true;
    if (localName.equals("last"))
      InLast = true;
    if (localName.equals("person"))
      IdNum = atts.getValue("idnum");
  } // startElement
```

**SAX Processing Example (5)**

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

```java
  public void startElement (String namespaceURI, String localName,
  String rawName, Attributes atts) {
    if (localName.equals("first"))
      InFirst = true;
    if (localName.equals("last"))
      InLast = true;
    if (localName.equals("person"))
      IdNum = atts.getValue("idnum");
  } // startElement
```

**SAX Processing Example (6)**

- **Call-back methods continue:**
  - Record the text content of elements `first` and `last` in corresponding variables:

```java
  public void characters (char ch[], int start, int length) {
    if (InFirst) FirstName = new String(ch, start, length);
    if (InLast) LastName = new String(ch, start, length);
  } // characters
```
SAX Processing Example (7)

- Call-back methods continue:
  - at an exit from `person`, output the collected data:
    ```java
    public void endElement(String namespaceURI,
    String localName, String qName) {
        if (localName.equals("person"))
            System.out.println(FirstName + " 
            LastName + " + IdNum + ");
        // Update the context flags:
        if (localName.equals("first"))
            InFirst = false;
        // (Correspondingly for "last" and InLast)
    }
    ```

SAX Processing Example (8)

- 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();
          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);
      }
  }
  ```

SAX Processing Example (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]);
  }
  // main
  ```