



Lecture 7: SAX

In which we examine some of the features and uses of the SAX – the simple API for XML.

1. Overview

SAX is an API – that is an application programmer interface. It includes a number of procedures/methods that application programmers can use.

2. SAX vs. DOM

We can only assume that S in SAX stands for simple-to-implement rather than simple-to-use. SAX is generally rather complicated to use – DOM usually provides an easier way to access XML documents.

SAX has one main advantage over DOM – it is much less expensive to execute for large documents. A DOM program must load the entire document into a program structure. Typically the memory required is several times the size of the document and this can be very expensive – particularly when the memory required exceeds the available RAM. In this case the program starts "paging" (swapping memory to disk) resulting in very poor performance.

Implementations of DOM are often built on top of SAX parsers.

3. Java

We use Java examples. SAX was originally defined and implemented in Java. It is available for many languages.

In Java we implement the ContentHandler class in order to write a SAX program. In other languages different mechanism may be used.

4. class ContentHandler

The class ContentHandler has an interface defined at

<http://www.saxproject.org/apidoc/org/xml/sax/ContentHandler.html>

Some important methods of ContentHandler

void	<u>characters</u> (char[] ch, int start, int length) Receive notification of character data.
void	<u>endDocument</u> () Receive notification of the end of a document.
void	<u>endElement</u> (java.lang.String uri, java.lang.String localName, java.lang.String qName) Receive notification of the end of an element.
void	<u>startDocument</u> () Receive notification of the beginning of a document.
void	<u>startElement</u> (java.lang.String uri, java.lang.String localName, java.lang.String qName, <u>Attributes</u> atts) Receive notification of the beginning of an element.

There are six other methods that must be implemented `setDocumentLocator` `startPrefixMapping` `endPrefixMapping` `skippedEntity` `processingInstruction` `ignorableWhitespace` – we will not consider them here.

A SAX program will typically implement `ContentHandler`. The appropriate methods are called by the parser as the document is processed.

4.1 Step by step behaviour of a parser

Input document	Sequence
	<code>startDocument</code>
<code><x v='1' w='2'></code>	<code>StartElement:</code> <code>localName = "x"</code> <code>atts = represents structure with v="1" and w="2"</code>
<code>abc</code>	<code>characters</code> <code>ch = "<x v='1' w='2'>abc</x>"</code> <code>start = 15</code> <code>length = 3</code>
<code></x></code>	<code>EndElement</code> <code>localName = "x"</code>
	<code>endDocument</code>

5. Using the ContentHandler

S.java

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.IOException;
public class S {
    public static void main(String[] args) {
        XMLReader parser;
        try {
            parser = XMLReaderFactory.createXMLReader(
                "org.apache.xerces.parsers.SAXParser");
        }
        catch (SAXException e) {
            return;
        }
        parser.setContentHandler(new NodeCounter());
        try {
            parser.parse("../stock.xml");
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

NodeCounter.java

```
import org.xml.sax.*;
public class NodeCounter implements ContentHandler {
    int nodeCount;
    public void startDocument() throws SAXException {
        nodeCount=0;
    }
    public void startElement(String namespaceURI, String localName,
        String qualifiedName, Attributes atts) throws SAXException {
        nodeCount++;
    }
    public void endElement(String namespaceURI, String localName,
        String qualifiedName) throws SAXException {
    }
    public void endDocument() throws SAXException {
        System.out.println("Number of nodes: " + nodeCount);
    }
    // Do-nothing methods
    public void setDocumentLocator(Locator l) {}
    public void startPrefixMapping(String p, String u) throws SAXException {}
    public void endPrefixMapping(String p) throws SAXException {}
    public void skippedEntity(String n) throws SAXException {}
    public void processingInstruction(String t, String d) throws SAXException {}
    public void characters(char[] t, int s, int l) throws SAXException {}
    public void ignorableWhitespace(char[] t, int s, int l) throws SAXException {}
}
```

6. Keeping track of context

Event based programs – where we get hooks into events that occur can be difficult. The routine **characters** is executed many times during the processing of a document – unless we keep track of the context we have no way of knowing what node we are in.

6.1 Using global variables to record state.

Suppose we need to count only the leaf nodes of a document. We can start with the NodeCounter handler shown – but we need to count only those nodes that are at the tip of the tree. We use a global variable nodeName – this is set as we enter an element and tested as we leave an element.

We can be sure we are at a leaf if this variable matches the node name as we exit the element.

```
...
int leafCount;
String nname;

public void startDocument() throws SAXException {
    leafCount=0;
    nname = "";
}

public void startElement(String namespaceURI, String localName,
    String qualifiedName, Attributes atts) throws SAXException {
    nname = localName;
}

public void endElement(String namespaceURI, String localName,
    String qualifiedName) throws SAXException {
    if (nname == localName) leafCount++;
    nname = "";
}

// Now that the document is done, we can print out the final results
public void endDocument() throws SAXException {
    System.out.println("Number of leaves: " + leafCount);
}
...
```

7. DOM vs SAX

SAX is an event based parser. Instead of loading the whole XML document SAX "fires-off" events at various points. We can require that an event occurs every time a particular node is started or when it has completed.

Consider a web browser. For long documents the browsers usually start displaying before the entire document is loaded – in many cases the browser will be able to refresh the display after each <p> node is read. However a table cannot be displayed until the entire table has loaded (why?)






While DOM provides facilities for changing the document SAX cannot (of course it can build up a copy as it goes).

DOM constructs an in-memory copy of the whole document. SAX need not. SAX can deal with very large documents efficiently.

As XML files are often loaded over (relatively) slow networks the performance of a SAX program may well be significant.

There is an overhead to DOM – the structure in memory may be many times the size of the raw file on disk.

End of unit summary

-  SAX is a "Simple API for XML" – for simple read "primitive"
-  SAX allows serial access to XML files
-  SAX uses an event driven style
-  State variables are pretty much unavoidable
-  Event-based parsers such as SAX operate allow the document to be processed as it loads.

SAX Questions

```
<?xml version="1.0" encoding="UTF-8" ?>
<stock andrew="was here">
  <item price="50" legend="Pr-Burger" BarCode="E1" />
  <item price="15" legend="Crisp S+V" BarCode="E5" />
  <item price="15" legend="Crisp C+O" BarCode="E6" />
  <item price="50" legend="Flat Cola" BarCode="E7" />
</stock>
```

Describe the output of each of the following programs fragment. You should assume that all the missing required methods are present but do nothing:

1)

```
int depth=0;

public void startElement(String namespaceURI, String localName,
String qualifiedName, Attributes atts) throws SAXException {
    depth++;
    System.out.println(localName + ":" + depth);
}

public void endElement(String namespaceURI, String localName,
String qualifiedName) throws SAXException {
    depth--;
}
```

2)

```
int depth=0;
int maxDepth=0;

public void startElement(String namespaceURI, String localName,
String qualifiedName, Attributes atts) throws SAXException {
    depth++;
    maxDepth=Math.max(maxDepth,depth);
}

public void endElement(String namespaceURI, String localName,
String qualifiedName) throws SAXException {
    depth--;
}

public void endDocument() throws SAXException {
    System.out.println(maxDepth);
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE movie SYSTEM "movie.dtd">
<movie id="_0034583">
  <title year="1942">Casablanca</title>
  <director>Michael Curtiz</director>
  <cast>
    <part>
      <actor>Humphrey Bogart</actor>
      <character>Richard "Rick" Blaine</character>
    </part>
    <part>
      <actor>Ingrid Bergman</actor>
      <character>Ilsa Lund Laszlo</character>
    </part>
    <part>
      <actor>Paul Henreid</actor>
      <character>Victor Laszlo</character>
    </part>
    <part>
      <actor>Claude Rains</actor>
      <character>Captain Louis Renault</character>
    </part>
  </cast>
  <genre style="drama romance"/>
  <tagline>They had a date with fate in Casablanca!</tagline>
  <rating voteCount="41057" date="17 Feb 2003">8.8</rating>
  <runtime>102 min</runtime><color value="false"/>
</movie>
```

```
boolean printOn;
public void startDocument() throws SAXException {
  printOn=false;
}
public void startElement(String namespaceURI, String localName,
  String qualifiedName, Attributes atts) throws SAXException {
  printOn = (localName.equals("director"));
}
public void characters(char[] t, int s, int l) throws SAXException {
  if (printOn)
    System.out.print(new String(t,s,l));
  printOn=false;
}
```

How would we print the cast?

How could we print only the leading actor?

How could we count the cast?