# [Difference between SAX Parser and DOM Parser](#Diff_SAX_DOM_Parser)

XML: eXtensible Markup Language is a uniform data representation and exchange mechanism.

<http://anupjani.wordpress.com/2009/04/10/simple-xmlxsdxslt-example-for-beginner-that-works/>

CSS is Style Sheet for HTML  
HTML uses predefined tags, and the meaning of each tag is well understood.  
The <table> tag in HTML defines a table – and a browser knows how to display it.  
Adding styles to HTML elements are simple.  
Telling a browser to display an element in a special font or color, is easy with CSS.

XSL is Style Sheet for XML  
On the other hand,  
XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is not well understood.  
A <table> tag could mean an HTML table, a piece of furniture, or something else – and a browser does not know how to display it.

XSL describes how the XML document should be displayed!

XSLT is a declarative programming language, you write an XSLT stylesheet to transform XML to HTML or XML or plain text.

[**What does “xmlns” in XML mean?**](http://stackoverflow.com/questions/1181888/what-does-xmlns-in-xml-mean)

[**http://stackoverflow.com/questions/1181888/what-does-xmlns-in-xml-mean**](http://stackoverflow.com/questions/1181888/what-does-xmlns-in-xml-mean)

**I saw the following line in an XML file:**

**xmlns:android="http://schemas.android.com/apk/res/android"**

**I have also seen xmlns in many other XML files that I've come across.**

**What is it?**

|  |  |
| --- | --- |
|  | It defines an [XML Namespace](http://www.w3.org/TR/REC-xml-names/).  In your example, the **Namespace Prefix** is "*android*" and the **Namespace URI** is "[*http://schemas.android.com/apk/res/android*](http://schemas.android.com/apk/res/android)"  In the document, you see elements like: *<android:foo />*  Think of the namespace prefix as a variable with a short name alias for the full namespace URI. It is the equivalent of writing *<http://schemas.android.com/apk/res/android:foo />* with regards to what it "means" when an XML parser reads the document.  ***NOTE:*** *You cannot actually use the full namespace URI in place of the namespace prefix in an XML instance document.*  Check out this tutorial on namespaces: <http://www.sitepoint.com/xml-namespaces-explained/> |

<http://stackoverflow.com/questions/740404/whats-the-difference-between-xslt-and-xsl-fo>

Difference between XSL AND XSLT?

aside from the similarities in the acronyms the two technologies used to be part of the same W3C spec(in draft form). It was later decided to split XSL(XSL-fo) and XSLT out into two separate specifications.

I was wondering about the basis of the question because I thought it was easy to answer, however as soon as you go here: <http://www.w3.org/TR/xsl/> it becomes clear that its actually a good question - because pretty much the first thing on the page is this statement:

This specification defines the features and syntax for the Extensible Stylesheet Language (XSL), a language for expressing stylesheets. It consists of two parts:

1. **a language for transforming XML documents (XSLT), and**
2. **an XML vocabulary for specifying formatting semantics.**

However, back in the real world, XSLT (which is also what most people will assume you mean by XSL) is a means for transforming XML documents into something else - that something else more often than not being another well structured document probably also formatted as XML (though I've used XSLT to render XML to csv and plain(ish) text).

XSL-FO on the other hand is about formatting - about laying out content on a page or a sequence of pages in a fairly strict fashion. Its useful because it allows you to manage content that is spread across multiple pages, its relatively straightforward to specify the format of a page (or even and odd pages) including headers, footers, borders, columns, etc and have your content flow into that. One would therefore take a load of data in, say, an XML format and then **use XSLT to convert that data into a document consisting of XSL-FO** that in turn is rendered using an appropriate tool in say PDF for print or other means of distribution.

The use case I have is to take a pile of tabular data, export that data as XML, render that into XSL-FO and from there to PDF which goes to a printer who turns said PDF into a 500 page directory. One specifies in the XSL-FO that one wants page numbers, page breaks in specific circumstances, that there is a table of contents and an index (each based on particular elements) and the rendering process (to PDF) handles filling in the page numbers across the board.

## Introducing XSL, XSLT, and XPath

The Extensible Stylesheet Language (XSL) has three major subcomponents:

**XSL-FO**

The Formatting Objects standard. By far the largest subcomponent, this standard gives mechanisms for describing font sizes, page layouts, and other aspects of object rendering. This subcomponent is *not* covered by JAXP, nor is it included in this tutorial.

**XSLT**

This is the transformation language, which lets you define a transformation from XML into some other format. For example, you might use XSLT to produce HTML or a different XML structure. You could even use it to produce plain text or to put the information in some other document format. (And as you'll see in [Generating XML from an Arbitrary Data Structure](http://docs.oracle.com/javaee/1.4/tutorial/doc/JAXPXSLT5.html#wp64712), a clever application can press it into service to manipulate non-XML data as well.)

**XPath**

At bottom, XSLT is a language that lets you specify what sorts of things to do when a particular element is encountered. But to write a program for different parts of an XML data structure, you need to specify the part of the structure you are talking about at any given time. XPath is that specification language. **It is an addressing mechanism that lets you specify a path to an element so that**, for example, <article><title> can be distinguished from <person><title>. In that way, you can describe different kinds of translations for the different <title> elements.

The remainder of this section describes the packages that make up the JAXP Transformation APIs.

### The JAXP Transformation Packages

Here is a description of the packages that make up the JAXP Transformation APIs:

**javax.xml.transform**

This package defines the factory class you use to get a Transformer object. You then configure the transformer with input (source) and output (result) objects, and invoke its transform() method to make the transformation happen. The source and result objects are created using classes from one of the other three packages.

**javax.xml.transform.dom**

Defines the DOMSource and DOMResult classes, which let you use a DOM as an input to or output from a transformation.

**javax.xml.transform.sax**

Defines the SAXSource and SAXResult classes, which let you use a SAX event generator as input to a transformation, or deliver SAX events as output to a SAX event processor.

**javax.xml.transform.stream**

Defines the StreamSource and StreamResult classes, which let you use an I/O stream as an input to or output from a transformation.

# [What is the difference between JAXP and JAXB?](http://stackoverflow.com/questions/2801502/what-is-the-difference-between-jaxp-and-jaxb)

JAXP (Java API for XML Processing) is a rather outdated umbrella term covering the various low-level XML APIs in JavaSE, such as DOM, SAX and StAX.

JAXB (Java API for XML Binding) is a specific API (the stuff under javax.xml.bind) that uses annotations to bind XML documents to a java object model.

# Difference between SAX Parser and DOM Parser

In SAX, events are triggered when the XML is being **parsed**. When the parser is parsing the XML, and encounters a tag starting (e.g. <something>), then it triggers the tagStarted event (actual name of event might differ). Similarly when the end of the tag is met while parsing (</something>), it triggers tagEnded. Using a SAX parser implies you need to handle these events and make sense of the data returned with each event.

In DOM, there are no events triggered while parsing. The entire XML is parsed and a DOM tree (of the nodes in the XML) is generated and returned. Once parsed, the user can navigate the tree to access the various data previously embedded in the various nodes in the XML.

In general, DOM is easier to use but has an overhead of parsing the entire XML before you can start using it.

It depends on the situation, SAX is preferable if you are accessing the entire XML in its sequential order as it takes lesser memory. DOM is preferable if you want random access of middle elements in an XML But it needs more memory. This balance is similar to memory Vs Time space balance in every program

**SAX**: (Simple API for XML)

* Parses node by node
* Doesn’t store the **XML** in memory
* We cant insert or delete a node
* **SAX** is an event based parser
* **SAX** is a Simple **API** for **XML**
* doesn’t preserve comments
* **SAX** generally runs a little faster than **DOM**

**DOM**: (Document Object Model)

* Stores the entire **XML** document into memory before processing
* Occupies more memory
* We can insert or delete nodes
* Traverse in any direction.
* **DOM** is a tree model parser
* **Document Object Model** (**DOM**) **API**
* Preserves comments
* **DOM** generally runs a little slower than **SAX**

If we need to find a node and doesn’t need to insert or delete we can go with **SAX** itself otherwise **DOM** provided we have more memory....

Examples:

**SAX parser** is work differently with DOM parser, it either load any XML document into memory nor create any object representation of the XML document. Instead, the SAX parser use callback function (org.xml.sax.helpers.DefaultHandler) to informs clients of the XML document structure.

SAX Parser is faster and uses **less** memory than DOM parser.

See following SAX callback methods :

* **startDocument()** and **endDocument()** – Method called at the start and end of an XML document.
* **startElement()** and **endElement()** – Method called at the start and end of a document element.
* **characters()** – Method called with the text contents in between the start and end tags of an XML document element.

#### 1. XML file

Create a simple XML file.

**<?xml** version="1.0"**?>**

**<company>**

**<staff>**

**<firstname>**yong**</firstname>**

**<lastname>**mook kim**</lastname>**

**<nickname>**mkyong**</nickname>**

**<salary>**100000**</salary>**

**</staff>**

**<staff>**

**<firstname>**low**</firstname>**

**<lastname>**yin fong**</lastname>**

**<nickname>**fong fong**</nickname>**

**<salary>**200000**</salary>**

**</staff>**

**</company>**

#### 2. Java file

Use SAX parser to parse the XML file.

**import** javax.xml.parsers.SAXParser;

**import** javax.xml.parsers.SAXParserFactory;

**import** org.xml.sax.Attributes;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** ReadXMLFile {

**public** **static** **void** main(String argv[]) {

**try** {

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

DefaultHandler handler = **new** DefaultHandler() {

**boolean** bfname = **false**;

**boolean** blname = **false**;

**boolean** bnname = **false**;

**boolean** bsalary = **false**;

**public** **void** startElement(String uri, String localName,String qName,

Attributes attributes) **throws** SAXException {

System.out.println("Start Element :" + qName);

**if** (qName.equalsIgnoreCase("FIRSTNAME")) {

bfname = **true**;

}

**if** (qName.equalsIgnoreCase("LASTNAME")) {

blname = **true**;

}

**if** (qName.equalsIgnoreCase("NICKNAME")) {

bnname = **true**;

}

**if** (qName.equalsIgnoreCase("SALARY")) {

bsalary = **true**;

}

}

**public** **void** endElement(String uri, String localName,

String qName) **throws** SAXException {

System.out.println("End Element :" + qName);

}

**public** **void** characters(**char** ch[], **int** start, **int** length) **throws** SAXException {

**if** (bfname) {

System.out.println("First Name : " + **new** String(ch, start, length));

bfname = **false**;

}

**if** (blname) {

System.out.println("Last Name : " + **new** String(ch, start, length));

blname = **false**;

}

**if** (bnname) {

System.out.println("Nick Name : " + **new** String(ch, start, length));

bnname = **false**;

}

**if** (bsalary) {

System.out.println("Salary : " + **new** String(ch, start, length));

bsalary = **false**;

}

}

};

saxParser.parse("c:**\\**file.xml", handler);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

<http://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser>

#### 1. DOM XML Parser Example

This example shows you how to get the node by “name”, and display the value.

/Users/mkyong/staff.xml

**<?xml** version="1.0"**?>**

**<company>**

**<staff** id="1001"**>**

**<firstname>**yong**</firstname>**

**<lastname>**mook kim**</lastname>**

**<nickname>**mkyong**</nickname>**

**<salary>**100000**</salary>**

**</staff>**

**<staff** id="2001"**>**

**<firstname>**low**</firstname>**

**<lastname>**yin fong**</lastname>**

**<nickname>**fong fong**</nickname>**

**<salary>**200000**</salary>**

**</staff>**

**</company>**

ReadXMLFile.java

**package** com.mkyong.seo;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.DocumentBuilder;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.NodeList;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.Element;

**import** java.io.File;

**public** **class** ReadXMLFile {

**public** **static** **void** main(String argv[]) {

**try** {

File fXmlFile = **new** File("/Users/mkyong/staff.xml");

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

*//optional, but recommended*

*//read this - http://stackoverflow.com/questions/13786607/normalization-in-dom-parsing-with-java-how-does-it-work*

doc.getDocumentElement().normalize();

System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("staff");

System.out.println("----------------------------");

**for** (**int** temp = 0; temp < nList.getLength(); temp++) {

Node nNode = nList.item(temp);

System.out.println("**\n**Current Element :" + nNode.getNodeName());

**if** (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

System.out.println("Staff id : " + eElement.getAttribute("id"));

System.out.println("First Name : " + eElement.getElementsByTagName("firstname").item(0).getTextContent());

System.out.println("Last Name : " + eElement.getElementsByTagName("lastname").item(0).getTextContent());

System.out.println("Nick Name : " + eElement.getElementsByTagName("nickname").item(0).getTextContent());

System.out.println("Salary : " + eElement.getElementsByTagName("salary").item(0).getTextContent());

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

<http://www.javacodegeeks.com/2012/01/xml-parsing-using-saxparser-with.html>

# ****XML parsing using SaxParser with complete code****

SAX parser use callback function (org.xml.sax.helpers.DefaultHandler) to informs clients of the XML document structure. You should extend DefaultHandler and override few methods to achieve xml parsing.

The methods to override are

* startDocument() and endDocument() – Method called at the start and end of an XML document.
* startElement() and endElement() – Method called at the start and end of a document element.
* characters() – Method called with the text contents in between the start and end tags of an XML document element.

The following example demonstrates the uses of DefaultHandler to parse and XML document. It performs mapping of xml to model class and generate list of objects.

**Sample XML Document :**

|  |  |  |
| --- | --- | --- |
| 01 | <?xml version="1.0" encoding="UTF-8"?> | |
| 02 | <catalog> |

|  |  |
| --- | --- |
| 03 | <book id="001" lang="ENG"> |
| 04 | <isbn>23-34-42-3</isbn> | |

|  |  |
| --- | --- |
| 05 | <regDate>1990-05-24</regDate> |
| 06 | <title>Operating Systems</title> | |

|  |  |  |
| --- | --- | --- |
| 07 | <publisher country="USA">Pearson</publisher> | |
| 08 | <price>400</price> |

|  |  |
| --- | --- |
| 09 | <authors> |
| 10 | <author>Ganesh Tiwari</author> | |

|  |  |  |
| --- | --- | --- |
| 11 | </authors> | |
| 12 | </book> |

|  |  |
| --- | --- |
| 13 | <book id="002"> |
| 14 | <isbn>24-300-042-3</isbn> | |

|  |  |
| --- | --- |
| 15 | <regDate>1995-05-12</regDate> |
| 16 | <title>Distributed Systems</title> | |

|  |  |  |
| --- | --- | --- |
| 17 | <publisher country="Nepal">Ekata</publisher> | |
| 18 | <price>500</price> |

|  |  |
| --- | --- |
| 19 | <authors> |
| 20 | <author>Mahesh Poudel</author> | |

|  |  |  |
| --- | --- | --- |
| 21 | <author>Bikram Adhikari</author> | |
| 22 | <author>Ramesh Poudel</author> |

|  |  |  |
| --- | --- | --- |
| 23 | </authors> | |
| 24 | </book> |

|  |  |
| --- | --- |
| 25 | </catalog> |

**Model Class for Book Object for Mapping xml to object**

|  |  |
| --- | --- |
| 01 | /\*\* |
| 02 | \* Book class stores book information, after parsing the xml | |

|  |  |  |
| --- | --- | --- |
| 03 | \* @author Ganesh Tiwari | |
| 04 | \*/ |

|  |  |  |
| --- | --- | --- |
| 05 | public class Book { | |
| 06 | String lang; |

|  |  |  |
| --- | --- | --- |
| 07 | String title; | |
| 08 | String id; |

|  |  |
| --- | --- |
| 09 | String isbn; |
| 10 | Date regDate; | |

|  |  |  |
| --- | --- | --- |
| 11 | String publisher; | |
| 12 | int price; |

|  |  |  |
| --- | --- | --- |
| 13 | List<String> authors; | |
| 14 | public Book(){ |

|  |  |  |
| --- | --- | --- |
| 15 | authors=new ArrayList<String>(); | |
| 16 | } |

|  |  |  |
| --- | --- | --- |
| 17 | //getters and setters | |
| 18 | } |

**Java Code for XML Parsing (Sax) :**

|  |  |
| --- | --- |
| 01 | import java.io.IOException; |
| 02 | import java.text.ParseException; | |

|  |  |  |
| --- | --- | --- |
| 03 | import java.text.SimpleDateFormat; | |
| 04 | import java.util.ArrayList; |

|  |  |  |
| --- | --- | --- |
| 05 | import java.util.List; | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | import javax.xml.parsers.ParserConfigurationException; | |
| 08 | import javax.xml.parsers.SAXParser; |

|  |  |  |
| --- | --- | --- |
| 09 | import javax.xml.parsers.SAXParserFactory; | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | import org.xml.sax.Attributes; |
| 12 | import org.xml.sax.SAXException; | |

|  |  |
| --- | --- |
| 13 | import org.xml.sax.helpers.DefaultHandler; |
| 14 | public class MySaxParser extends DefaultHandler { | |

|  |  |
| --- | --- |
| 15 | List<Book> bookL; |
| 16 | String bookXmlFileName; | |

|  |  |  |
| --- | --- | --- |
| 17 | String tmpValue; | |
| 18 | Book bookTmp; |

|  |  |  |
| --- | --- | --- |
| 19 | SimpleDateFormat sdf= new SimpleDateFormat("yy-MM-dd"); | |
| 20 | public MySaxParser(String bookXmlFileName) { |

|  |  |  |
| --- | --- | --- |
| 21 | this.bookXmlFileName = bookXmlFileName; | |
| 22 | bookL = new ArrayList<Book>(); |

|  |  |  |
| --- | --- | --- |
| 23 | parseDocument(); | |
| 24 | printDatas(); |

|  |  |
| --- | --- |
| 25 | } |
| 26 | private void parseDocument() { | |

|  |  |
| --- | --- |
| 27 | // parse |
| 28 | SAXParserFactory factory = SAXParserFactory.newInstance(); | |

|  |  |
| --- | --- |
| 29 | try { |
| 30 | SAXParser parser = factory.newSAXParser(); | |

|  |  |
| --- | --- |
| 31 | parser.parse(bookXmlFileName, this); |
| 32 | } catch (ParserConfigurationException e) { | |

|  |  |  |
| --- | --- | --- |
| 33 | System.out.println("ParserConfig error"); | |
| 34 | } catch (SAXException e) { |

|  |  |  |
| --- | --- | --- |
| 35 | System.out.println("SAXException : xml not well formed"); | |
| 36 | } catch (IOException e) { |

|  |  |  |
| --- | --- | --- |
| 37 | System.out.println("IO error"); | |
| 38 | } |

|  |  |
| --- | --- |
| 39 | } |
| 40 | private void printDatas() { | |

|  |  |  |
| --- | --- | --- |
| 41 | // System.out.println(bookL.size()); | |
| 42 | for (Book tmpB : bookL) { |

|  |  |  |
| --- | --- | --- |
| 43 | System.out.println(tmpB.toString()); | |
| 44 | } |

|  |  |
| --- | --- |
| 45 | } |
| 46 | @Override | |

|  |  |  |
| --- | --- | --- |
| 47 | public void startElement(String s, String s1, String elementName, Attributes attributes) throwsSAXException { | |
| 48 | // if current element is book , create new book |

|  |  |  |
| --- | --- | --- |
| 49 | // clear tmpValue on start of element | |
| 50 |  |

|  |  |  |
| --- | --- | --- |
| 51 | if (elementName.equalsIgnoreCase("book")) { | |
| 52 | bookTmp = new Book(); |

|  |  |
| --- | --- |
| 53 | bookTmp.setId(attributes.getValue("id")); |
| 54 | bookTmp.setLang(attributes.getValue("lang")); | |

|  |  |
| --- | --- |
| 55 | } |
| 56 | // if current element is publisher | |

|  |  |
| --- | --- |
| 57 | if (elementName.equalsIgnoreCase("publisher")) { |
| 58 | bookTmp.setPublisher(attributes.getValue("country")); | |

|  |  |  |
| --- | --- | --- |
| 59 | } | |
| 60 | } |

|  |  |
| --- | --- |
| 61 | @Override |
| 62 | public void endElement(String s, String s1, String element) throws SAXException { | |

|  |  |  |
| --- | --- | --- |
| 63 | // if end of book element add to list | |
| 64 | if (element.equals("book")) { |

|  |  |  |
| --- | --- | --- |
| 65 | bookL.add(bookTmp); | |
| 66 | } |

|  |  |  |
| --- | --- | --- |
| 67 | if (element.equalsIgnoreCase("isbn")) { | |
| 68 | bookTmp.setIsbn(tmpValue); |

|  |  |
| --- | --- |
| 69 | } |
| 70 | if (element.equalsIgnoreCase("title")) { | |

|  |  |  |
| --- | --- | --- |
| 71 | bookTmp.setTitle(tmpValue); | |
| 72 | } |

|  |  |  |
| --- | --- | --- |
| 73 | if(element.equalsIgnoreCase("author")){ | |
| 74 | bookTmp.getAuthors().add(tmpValue); |

|  |  |
| --- | --- |
| 75 | } |
| 76 | if(element.equalsIgnoreCase("price")){ | |

|  |  |  |
| --- | --- | --- |
| 77 | bookTmp.setPrice(Integer.parseInt(tmpValue)); | |
| 78 | } |

|  |  |  |
| --- | --- | --- |
| 79 | if(element.equalsIgnoreCase("regDate")){ | |
| 80 | try { |

|  |  |  |
| --- | --- | --- |
| 81 | bookTmp.setRegDate(sdf.parse(tmpValue)); | |
| 82 | } catch (ParseException e) { |

|  |  |  |
| --- | --- | --- |
| 83 | System.out.println("date parsing error"); | |
| 84 | } |

|  |  |  |
| --- | --- | --- |
| 85 | } | |
| 86 | } |

|  |  |
| --- | --- |
| 87 | @Override |
| 88 | public void characters(char[] ac, int i, int j) throws SAXException { | |

|  |  |  |
| --- | --- | --- |
| 89 | tmpValue = new String(ac, i, j); | |
| 90 | } |

|  |  |  |
| --- | --- | --- |
| 91 | public static void main(String[] args) { | |
| 92 | new MySaxParser("catalog.xml"); |

|  |  |  |
| --- | --- | --- |
| 93 | } | |
| 94 | } |

**Output of Parsing :**

[view source](http://www.javacodegeeks.com/2012/01/xml-parsing-using-saxparser-with.html#viewSource)[print](http://www.javacodegeeks.com/2012/01/xml-parsing-using-saxparser-with.html#printSource)[?](http://www.javacodegeeks.com/2012/01/xml-parsing-using-saxparser-with.html#about)

|  |  |
| --- | --- |
| 1 | Book [lang=ENG, title=Operating Systems, id=001, isbn=23-34-42-3, regDate=Thu May 24 00:00:00 NPT 1990, publisher=USA, price=400, authors=[Ganesh Tiwari]] |
| 2 | Book [lang=null, title=Distributed Systems, id=002, isbn=24-300-042-3, regDate=Fri May 12 00:00:00 NPT 1995, publisher=Nepal, price=500, authors=[Mahesh Poudel, Bikram Adhikari, Ramesh Poudel]] |