

XML

- XML stands for eXtensible Markup Language
- XML was designed to carry data, not to display data
- XML tags are not predefined.
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

XML Document Example

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The Difference between XML and HTML

XML is not a replacement for HTML. XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.

HTML is about displaying information, while XML is about carrying information.

XML is Just Plain Text

XML is nothing special. It is just plain text. Software that can handle plain text can also handle XML. However, XML-aware applications can handle the XML tags specially. The functional meaning of the tags depends on the nature of the application.

XML is Not a Replacement for HTML

XML is a complement to HTML: It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data. My best description of XML is this:

XML is a software- and hardware-independent tool for carrying information.

XML is everywhere

We have been participating in XML development since its creation. It has been amazing to see how quickly the XML standard has developed, and how quickly a large number of software vendors have adopted the standard.

XML is now as important for the Web as HTML was to the foundation of the Web.

XML is everywhere. It is the most common tool for data transmissions between all sorts of applications, and is becoming more and more popular in the area of storing and describing information. XML is used in many aspects of web development, often to simplify data storage and sharing.

XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.

XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that different applications can share.

XML Simplifies Data Transport

With XML, data can easily be exchanged between incompatible systems.

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Data More Available

Since XML is independent of hardware, software and application, XML can make your data more available and useful.

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

If Developers Have Sense

If they DO have sense, future applications will exchange their data in XML.

The future might give us word processors, spreadsheet applications and databases that can read each other's data in a pure text format, without any conversion utilities in between.

We can only pray that all the software vendors will agree.

An Example XML Document

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the **root element** of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

XML Documents Form a Tree Structure

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

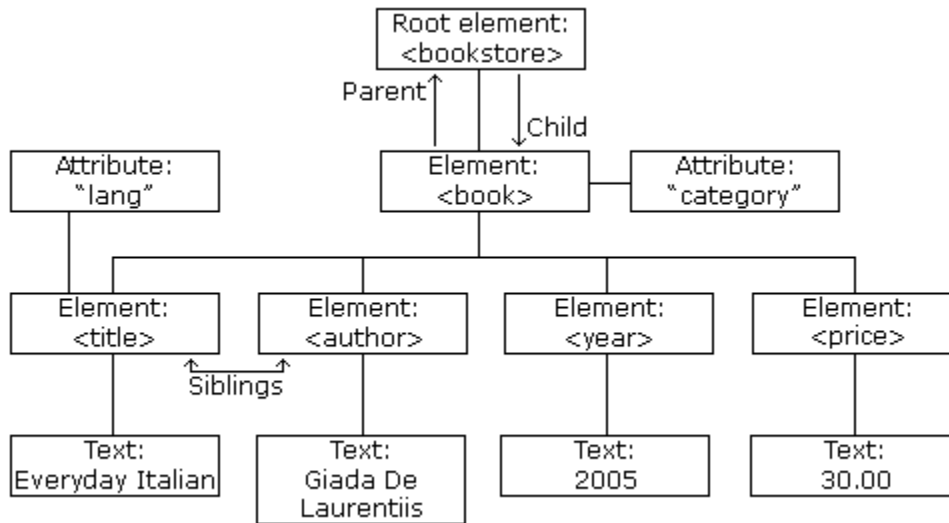
All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Example:



The image above represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is `<bookstore>`. All `<book>` elements in the document are contained within `<bookstore>`.

The `<book>` element has 4 children: `<title>`, `<author>`, `<year>`, `<price>`.

All XML Elements Must Have a Closing Tag

In HTML, you will often see elements that don't have a closing tag:

```
<p>This is a paragraph  
<p>This is another paragraph
```

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

```
<p>This is a paragraph</p>  
<p>This is another paragraph</p>
```

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

XML Tags are Case Sensitive

XML elements are defined using XML tags. XML tags are case sensitive. With XML, the tag <Letter> is different from the tag <letter>. Opening and closing tags must be written with the same case:

```
<Message>This is incorrect</message>  
<message>This is correct</message>
```

Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the previous example , "Properly nested" simply means that since the <i> element is opened inside the element, it must be closed inside the element.

XML Documents Must Have a Root Element

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

Entity References

Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element. This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

```
<!-- This is a comment -->
```

White-space is preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML:	Hello my name is Tove
Output:	Hello my name is Tove.

With XML, the white-space in a document is not truncated.

XML Stores New Line as LF

In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF). The character pair bears some resemblance to the typewriter actions of setting a new line. In Unix applications, a new line is normally stored as a LF character. Macintosh applications also use an LF to store a new line.

XML Element

An XML element is everything from (including) the element's start tag to (including) the element's end tag. An element can contain other elements, simple text or a mixture of both. Elements can also have attributes.

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <author> has **text content** because it contains text.

In the example above only <book> has an **attribute** (category="CHILDREN").

XML Naming Rules

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Any name can be used, no words are reserved.

Best Naming Practices

Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.

Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.

Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."

Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).

XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

XML Elements are Extensible

XML elements can be extended to carry more information.

Look at the following XML example:

```
<note>
<to>Tove</to>
<from>Jani</from>
<body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

MESSAGE

To: Tove
From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
<date>2008-01-10</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the `<to>`, `<from>`, and `<body>` elements in the XML document and produce the same output.

One of the beauties of XML, is that it can often be extended without breaking applications.

XML elements can have attributes in the start tag, just like HTML. Attributes provide additional information about elements.

XML Attributes

From HTML you will remember this: ``. The "src" attribute provides additional information about the `` element.

In HTML (and in XML) attributes provide additional information about elements:

```

<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

XML Attributes Must be Quoted

Attribute values must always be enclosed in quotes, but either single or double quotes can be used. For a person's sex, the person tag can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML Elements vs. Attributes

Take a look at these examples:

```
<person sex="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

```
<person>  
  <sex>female</sex>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.

There are no rules about when to use attributes and when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="10/01/2008">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
  <date>10/01/2008</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <day>10</day>
    <month>01</month>
    <year>2008</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Avoid XML Attributes?

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

Don't end up like this:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the ID attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The ID above is just an identifier, to identify the different notes. It is not a part of the note itself. What I'm trying to say here is that metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the ID attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The ID above is just an identifier, to identify the different notes. It is not a part of the note itself.

So the metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

XML Schema

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Viewing XML Files

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <note>
```

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Look at this XML file: [note.xml](#)

The XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Note: In Chrome, Opera, and Safari, only the element text will be displayed. To view the raw XML, you must right click the page and select "View Source".

Why Does XML Display Like This?

XML documents do not carry information about how to display the data.

Since XML tags are "invented" by the author of the XML document, browsers do not know if a tag like <table> describes an HTML table or a dining table.

Without any information about how to display the data, most browsers will just display the XML document as it is.

Displaying XML with XSLT

XSLT is the recommended style sheet language of XML.

XSLT (eXtensible Stylesheet Language Transformations).

XSLT can be used to transform XML into HTML, before it is displayed by a browser:

Transforming XML with XSLT on the Server

In the example above, the XSLT transformation is done by the browser, when the browser reads the XML file.

Different browsers may produce different result when transforming XML with XSLT. To reduce this problem the XSLT transformation can be done on the server.

Note that the result of the output is exactly the same, either the transformation is done by the web server or by the web browser.

Example

1.XML File (DepRec.xml)

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="DepRec.xsl"?>

<NewDataSet>
  <Table>
    <DepID>36</DepID>
    <DepName>Admin </DepName>
    <CategID>2</CategID>
    <CategName>Stationary</CategName>
    <ItemNo>1</ItemNo>
    <ItemName>A4 Paper</ItemName>
    <Description>box of 5 bundles</Description>
    <Qty>2</Qty>
    <PNo>3</PNo>
    <PDate>2006-07-26T00:00:00+04:00</PDate>
    <EmployeeName>Ahmed Ali</EmployeeName>
  </Table>
  <Table>
    <DepID>36</DepID>
    <DepName>Admin </DepName>
    <CategID>2</CategID>
```

```

<CategName>Stationary</CategName>
<ItemNo>53</ItemNo>
<ItemName>Tape (flamingo)</ItemName>
<Description>unit</Description>
<Qty>1</Qty>
<PNo>3</PNo>
<PDate>2006-07-26T00:00:00+04:00</PDate>
<EmployeeName>Saif GHassan</EmployeeName>
</Table>
<Table>
<DepID>36</DepID>
<DepName>Admin </DepName>
<CategID>2</CategID>
<CategName>Stationary</CategName>
<ItemNo>54</ItemNo>
<ItemName>Tape Dispenser</ItemName>
<Description>unit</Description>
<Qty>1</Qty>
<PNo>3</PNo>
<PDate>2006-07-26T00:00:00+04:00</PDate>
<EmployeeName>Hani Saad</EmployeeName>
</Table>
</NewDataSet>

```

2. XSL file (DepRec.xsl)

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<style>
.value { font-family:Arial;white-space=pre;}

```

```

</style>
<table border="1" cellspacing="0" cellpadding="3" bordercolor="gainsboro"
width="70%">
  <tr bgcolor="#ffc435">
    <th width="10%" align="center">Category Name</th>
    <th width="10%" align="center">Pickup No.</th>
    <th width="10%" align="center">Item No.</th>
    <th width="20%" align="center">ItemName</th>
    <th width="10%" align="center">Description</th>
    <th width="10%" align="center">Quantity</th>
    <th width="30%" align="center">Contact Person</th>
  </tr>
  <xsl:for-each select='NewDataSet/Table'>
    <tr>
      <td class="value" width="10%" align="center">
        <xsl:value-of select='CategName'/>
      </td>
      <td class="value" width="10%" align="center">
        <xsl:value-of select='PNo'/>
      </td>
      <td class="value" width="10%" align="center">
        <xsl:value-of select='ItemNo'/>
      </td>
      <td class="value" width="20%" align="center">
        <xsl:value-of select='ItemName'/>
      </td>
      <td class="value" width="20%" align="center">
        <xsl:value-of select='Description'/>
      </td>
      <td class="value" width="10%" align="center">
        <xsl:value-of select='Qty'/>

```

```
</td>

<td class="value" width="20%" align="center">
  <xsl:value-of select='EmployeeName' />
</td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

XSL Languages

It started with XSL and ended up with XSLT, XPath, and XSL-FO.

It Started with XSL

XSL stands for EXtensible Stylesheet Language.

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

CSS = Style Sheets 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 = Style Sheets for XML

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!

XSL - More Than a Style Sheet Language

XSL consists of three parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents

- XSL-FO - a language for formatting XML documents
-

XSLT Introduction

XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

XPath is a language for navigating in XML documents.

What is XSLT?

- XSLT stands for XSL Transformations
 - XSLT is the most important part of XSL
 - XSLT transforms an XML document into another XML document
 - XSLT uses XPath to navigate in XML documents
 - XSLT is a W3C Recommendation
 - XSLT = XSL Transformations
 - XSLT is the most important part of XSL.
 - XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.
 - With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
 - A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree.**
 - ---
 - XSLT Uses XPath
 - XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.
 - If you want to study XPath first, please read our
-

XSLT is a W3C Recommendation

XSLT became a W3C Recommendation 16. November 1999.

To read more about the XSLT activities at W3C, please read our

Example study: How to transform XML into XHTML using XSLT.

The details of this example will be explained in the next chapter.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is `<xsl:stylesheet>` or `<xsl:transform>`.

Note: `<xsl:stylesheet>` and `<xsl:transform>` are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

or:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute `version="1.0"`.

Start with a Raw XML Document

We want to **transform** the following XML document ("book_store.xml") into XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<book_store>
  <book>
    <author> Susan Matteson </author>
    <title> LINUX&#174; DESKTOP GARAGE </title>
    <publisher> Prentice Hall PTR </publisher>
    <year> 2005 </year>
  </book>
</book_store>
```

Viewing XML Files in Firefox and Internet Explorer: Open the XML file (typically by clicking on a link) - The XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Viewing XML Files in Netscape 6: Open the XML file, then right-click in XML file and select "View Page Source". The XML document will then be displayed with color-coded root and child elements.

Viewing XML Files in Opera 7: Open the XML file, then right-click in XML file and select "Frame" / "View Source". The XML document will be displayed as plain text.

XSLT - Transformation

Example study: How to transform XML into XHTML using XSLT.

The details of this example will be explained in the next chapter.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is `<xsl:stylesheet>` or `<xsl:transform>`.

Note: `<xsl:stylesheet>` and `<xsl:transform>` are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

or:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` points to the official W3C XSLT

namespace. If you use this namespace, you must also include the attribute version="1.0".

Start with a Raw XML Document

We want to **transform** the following XML document ("book_store.xml ") into XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<book_store>
  <book>
    <author> Susan Matteson </author>
    <title> LINUX&#174; DESKTOP GARAGE </title>
    <publisher> Prentice Hall PTR </publisher>
    <year> 2005 </year>
  </book>
</book_store>
```

Viewing XML Files in Firefox and Internet Explorer: Open the XML file (typically by clicking on a link) - The XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Viewing XML Files in Netscape 6: Open the XML file, then right-click in XML file and select "View Page Source". The XML document will then be displayed with color-coded root and child elements.

Viewing XML Files in Opera 7: Open the XML file, then right-click in XML file and select "Frame" / "View Source". The XML document will be displayed as plain text.

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("book_store.xsl") with a transformation template:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
```

```

        <th>Title</th>
        <th>Publisher</th>
    </tr>
    <xsl:for-each select=" book_store/book">
    <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="publisher"/></td>
    </tr>
    </xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("book_store.xml"):

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="book_store.xsl"?>
<book_store>
  <book>
    <author> Susan Matteson </author>
    <title> LINUX&#174; DESKTOP GARAGE </title>
    <publisher> Prentice Hall PTR </publisher>
    <year> 2005 </year>
  </book>
  .
  .
  .
</book_store>

```

If you have an XSLT compliant browser it will nicely **transform** your XML into XHTML.

The details of the example above will be explained in the next chapters.

XSLT <xsl:template> Element

An XSL style sheet consists of one or more set of rules that are called templates.

A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

Ok, let's look at a simplified version of the XSL file from the previous chapter:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="ISO-8859-1"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the root of the XML source document.

The content inside the `<xsl:template>` element defines some HTML to write to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next chapter you will learn how to use the `<xsl:value-of>` element to select values from the XML elements.

XSLT `<xsl:value-of>` Element

The `<xsl:value-of>` element is used to extract the value of a selected node.

The `<xsl:value-of>` Element

The `<xsl:value-of>` element can be used to extract the value of an XML element and add it to the output stream of the transformation:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <tr>
      <td><xsl:value-of select="book_store/book/title"/></td>
      <td><xsl:value-of select="book_store/book/publisher"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Example Explained

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The result of the transformation above will look like this:

The result from this example was also a little disappointing, because only one line of data was copied from the XML document to the output. In the next chapter you will learn how to use the `<xsl:for-each>` element to loop through the XML elements, and display all of the records.

XSLT `<xsl:for-each>` Element

The `<xsl:for-each>` element allows you to do looping in XSLT.

The `<xsl:for-each>` Element

The XSL `<xsl:for-each>` element can be used to select every XML element of a specified node-set:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <xsl:for-each select="book_store/book">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="publisher"/></td>
      </tr>
    </xsl:for-each>
  </table>
```

```
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

Filtering the Output

We can also filter the output from the XML file by adding a criterion to the **select** attribute in the `<xsl:for-each>` element.

```
<xsl:for-each select=" book_store/book[publisher =' Prentice Hall PTR ']">
```

Legal filter operators are:

- = (equal)
- != (not equal)
- < less than
- > greater than

Take a look at the adjusted XSL style sheet:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Author</th>
    </tr>
    <xsl:for-each select="book_store/book[publisher='Prentice Hall PTR']">
    <tr>
      <td><xsl:value-of select="title"/></td>
```

```

        <td><xsl:value-of select="publisher" /></td>
    </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

XSLT <xsl:sort> Element

The <xsl:sort> element is used to sort the output.

Where to put the Sort Information

To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file:

Example

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <xsl:for-each select="book_store/book">
      <xsl:sort select="publisher" />
      <tr>
        <td><xsl:value-of select="title" /></td>
        <td><xsl:value-of select="publisher" /></td>
      </tr>
    </xsl:for-each>
  </table>

```

```
</body>
</html>
</xsl:template>
</xs
l:stylesheet>
```

Note: The **select** attribute indicates what XML element to sort on.

XSLT <xsl:if> Element

The <xsl:if> element is used to put a conditional test against the content of the XML file.

The <xsl:if> Element

To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.

Syntax

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

Where to Put the <xsl:if> Element

To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
```

```

<h2>My Book Store</h2>
<table border="1">
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Publisher</th>
  </tr>
  <xsl:for-each select="book_store/book">
    <xsl:if test="price > 10">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="publisher"/></td>
      </tr>
    </xsl:if>
  </xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

Note: The value of the required **test** attribute contains the expression to be evaluated.

The code above will only output the title and author elements of the CDs that has a price that is higher than 10.

XSLT <xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

The <xsl:choose> Element

Syntax

```

<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>

```

Where to put the Choose Condition

To insert a multiple conditional test against the XML file, add the `<xsl:choose>`, `<xsl:when>`, and `<xsl:otherwise>` elements to the XSL file:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <xsl:for-each select="book_store/book">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <xsl:choose>
          <xsl:when test="price > 10">
            <td bgcolor="#ff00ff">
              <xsl:value-of select="publisher"/></td>
          </xsl:when>
          <xsl:otherwise>
            <td><xsl:value-of select="publisher"/></td>
          </xsl:otherwise>
        </xsl:choose>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

The code above will add a pink background-color to the "Auther" column WHEN the price of the CD is higher than 10.

Another Example

Here is another example that contains two <xsl:when> elements:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Publisher</th>
    </tr>
    <xsl:for-each select="book_store/book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
        <xsl:when test="price > 10">
          <td bgcolor="#ff00ff">
            <xsl:value-of select="publisher"/></td>
        </xsl:when>
        <xsl:when test="price > 9">
          <td bgcolor="#cccccc">
            <xsl:value-of select="publisher"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="publisher"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

The code above will add a pink background color to the "Author" column WHEN the price of the CD is higher than 10, and a grey background-color WHEN the price of the CD is higher than 9 and lower or equal to 10.

XSLT <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

The <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

Look at the following XSL style sheet:

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My Book Store</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="publisher"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

<xsl:template match="publisher">
  Autho: <span style="color:#00ff00">
```

```
<xsl:value-of select="." /></span>  
<br />  
</xsl:template>  
  
</xsl:stylesheet>
```