

# CHAPTER: 5 XML

## TOPIC:1 Introduction to XML

XML is a software- and hardware-independent tool for storing and transporting data

### 1.1 What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

### 1.2 The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are
- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability
- Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.
- XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.
- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc

HTML	XML
It focus on presenting data i.e. <b>how</b> data looks.	It focus on describing data, i.e <b>what</b> data is.
HTML tags are predefined. E.g. <table>, <form> etc.	XML tags are customized tags. E.g. <name>, <address>,<city> etc.
HTML provides predefined elements and attributes whose behavior is well specified.	XML allows to create your own elements that are specific to your application or business needs.
It does not support case-sensitivity	It support case-sensitivity
It specifies that it is not mandatory to close each and every tag.	It specifies that it is mandatory to close each and every tag.
It specifies that it is not mandatory to enclose attribute values in double quotes(" ")	It specifies that it is mandatory to enclose attribute values in double quotes(" ")

## 1.3 XML Does Not Use Predefined Tags

- The XML language has no predefined tags.
- The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
- HTML works with predefined tags like <p>, <h1>, <table>, etc.
- With XML, the author must define both the tags and the document structure.

### EXAMPLE

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

The XML above is quite self-descriptive:

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body.

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it:

### OUTPUT:

## Note

To: Tove

From: Jani

## Reminder

Don't forget me this weekend!

## 1.4 XML Features

- Structured language:

- Platform independent
- Open standard
- Language independent
- Web enabled
- Extensible

**Structured language:**

- XML defines the structure of a document using tags.
- XML is a user-defined language, implying that it allows you to create your own tags. – Platform **independent**:
- To execute the XML applications on any operating system.

**Open standard:**

- Creates a fair, competitive market for implementations.
- Customers need not to go to a particular vendor or group to use XML.

**Language independent:**

- Implies that you can use XML with any other technology or language, such as Java, .NET and Hypertext Preprocessor(PHP)

**Web enabled:**

- HTML content can be re-used to work with XML.
- XML is frequently used over the web for data transfer.

**Extensible:**

- Allow you to create user- define tages.

## 1.5 Advantage of XML

- stores data in form of plain text.
- Provides a basic syntax that can be used to share information among different applications.
- Allow you not to restrict to a limited set of tags.
- It is completely compatible with java.
- 100% portable that it can be used to large network with multiple platforms such as the Internet and it can be used on handhelds or palmtops.
- Represents the international standards of web.





## 1.6 Disadvantages of XML




- Represents redundant and similar syntax for binary data. This redundancy affects the efficiency of an application.
- XML does not provide any specific notion for integer, string, Boolean and so on.
- Requires a processing application so that anyone , anywhere in the world, can read your document.
- Requires XML specific browsers, which are

# TOPIC:2 XML Key Components

## XML Components

The following table lists the components that make up the XML layout in the Sterling B2B Integrator Map Editor, the icons that represent the components, and descriptions of the components. .

Component	Icon	Description
XML root element		The <i>XML root element</i> represents the XML document that is being mapped. The XML root element is a looping structure that contains elements and content particles that repeat in sequence until either the group data ends or the maximum number of times that the loop is permitted to repeat is exhausted. The root element cannot be referenced by standard rules or links.
Element		An <i>element</i> contains related elements and content particles. In addition, an element can contain one pcddata and one attribute container. These objects repeat in sequence until either the element data ends or the maximum number of times that the loop is permitted to repeat is exhausted. A repeating element that contains another repeating element corresponds to a nested looping structure. The XML Element object cannot be referenced by standard rules or links.
Abstract element		An <i>abstract element</i> is an inconcrete element from an XML schema (for example, the term “appliance” may be considered an abstract element while “dishwasher” is a concrete one). An abstract element must be substituted with a non-abstract element for a map to be successfully compiled.
Content particle		A <i>content particle</i> contains related subordinate objects that define either a choice, a sequence, or an all. A content particle can contain only one pcddata. If specified, these objects can repeat in sequence until either the content particle data ends or the maximum number of times that the loop is permitted to repeat is exhausted. If you create a content particle that is subordinate to another content particle, the content particle corresponds to a nested looping structure (a loop within a loop). A content particle cannot be referenced

		by standard rules or links.
Pcddata		<p>A <i>pcdata</i> contains character data in an XML document. Only one pcddata can be defined per element or content particle. Sterling B2B Integrator Map Editor automatically names the pcddata with the name of the parent element or content particle.</p> <p>When a pcddata has a link performed against it, a red check mark appears over the pcddata icon.</p> <p>When a pcddata contains an extended rule or a standard rule, an asterisk appears to the right of the pcddata icon.</p>
Attribute container		An <i>attribute container</i> does not correspond to an XML function. Sterling B2B Integrator Map Editor uses attribute containers to contain the attributes of an XML element. The attribute container has no properties. When you create the first attribute of an XML element, the Sterling B2B Integrator Map Editor automatically creates an attribute container object. An element can have only one attribute container, but the attribute container object can enclose many attribute objects.
Attribute		<p>An <i>attribute</i> specifies information associated with an element that further defines the element. The attribute is located within an attribute container. Attributes do not have to occur in sequence in the input data.</p> <p>When an attribute has a link performed against it, a red check mark appears over the attribute icon.</p> <p>When an attribute contains an extended rule or a standard rule, an asterisk appears to the right of the attribute icon.</p>

## Components of an XML document

An XML document can be decomposed into various sections and each section can be studied in detail. The components are:

- an XML Declaration
- a DOCTYPE declaration
- elements
- attributes on elements
- character data
- comments

- processing instructions
- entity references
- CDATA sections

An XML document consists of, in order:

- an XML declaration (*recommended*)
- leading comments, whitespace, or processing instructions (*optional*)
- a DOCTYPE declaration (*optional*)
- a document element
- trailing comments, whitespace, or processing instructions (*optional*)

## 1.1 XML Declaration!

A document's prolog should include an *XML Declaration*:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

This example explicitly indicates the document:

- is intended to conform to XML version 1.0 <http://www.w3.org/TR/REC-xml>
- is encoded using UTF-8 (*optional*).
- may not be parsable “standalone” (*optional*)

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

## 1.2 XML Elements

Every XML document contains one or more elements. Each element is delimited by *start-tags* and *end-tags*.

```
<catalog>...</ catalog>
```

```
<book isbn="1234-5678-90">...</book>
```

```
<title>The Wizard of Venice</title>
```

```
<author>William Shakespeare</author>
```

As you can see, end tags are mandatory. Element names starting “xml”, “Xml”, ... are reserved. Furthermore- an element that has no content is an *empty element*. An empty element may be represented by

an *empty-element tag*.(<hr/>)

Elements may nest – recursively – but may not overlap with other elements or markup.

- OK: <doc><section><p>text</p></section></doc>
- Not OK: <doc><p><section>text</p></section></doc>
- *Elements can contain content: markup (such as sub-elements), and character data.*
- *Legal characters are tab, carriage return, line feed, and Unicode graphic (non-control) characters. This effectively allows all possible text characters; it (unfortunately?) precludes arbitrary binary data.*
- *An XML document contains one or more elements.*
- *[Definition:] There is exactly one element, called the root, or document element, no part of which appears in the content of any other element.*

- For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

## 1.3 Element Attributes

- An element can have zero or more *attributes*.
- In a document, each an attribute has a *name* and a *value*: `<book isbn="1234-5678-90"> ...`
- XML allows you to control the *type* and the *default* value of attributes.
- Attribute names starting “xml”, “Xml”, ... are reserved.

### Elements or Attributes

#### Q: What should be an attribute, and what should be an element?

A: Often, using either an attribute *or* an element with content would both be reasonable choices.

- Sometimes you can't use an attribute: if the data itself has structure, or if its presence is dependent on structure, than you need to use an element.
- If you want to refer to an *XML unparsed entity* (for example, to refer to unparsable binary data), you need to use an attribute.
- If the content is large, you probably should use an element.
- If the information is more *metadata* than data, then you should probably use an attribute.

## 1.4 Comments(<!-- -->)

*Comments* may appear anywhere outside other markup. Comments start with the string “<!--” and end with “-->”:

<!-- This is a made up example -->

The content of a comment is not parsed. The comments are not part of the document's data. XML processors (do you know what a processor is?) are allowed to ignore them entirely. The string “--” must not occur within comments, meaning:

- Comments cannot end with “--->”
- Comments cannot nest
- Comments cannot overlap with other comments or markup.

## 1.5 Processing Instructions

*Processing instructions* (“PIs”) allow documents to contain instructions for specific applications:

<?latex optional-line-break?>

Like comments, processing instructions are not part of the document's data; unlike comments, XML processors must pass them to the application. PIs cannot nest, and cannot overlap with other PIs or markup and PI names starting with “xml”, “Xml”, ... are reserved.

## 1.6 Named Entity References

*Entity references* begin with a “&” and end with a “;”. To use a reference an entity by *name*, place its name between the “&” and the “;”.

XML pre-defines five entities:

- &gt; &lt; &apos; ' &quot; " &amp; &

This allows, for example: `&lt;element&gt;` All other entities must be defined in a DTD (*discussed later*). Entity names starting “xml”, “Xml”, ... are reserved.

## 1.7 Character References

*Character references* are used to insert arbitrary Unicode characters. A character reference is a hexadecimal or decimal number preceded by “&#” and followed by “;”, indicating the character’s Unicode code point:

- &#x20AC; € (note the leading “x”)
- &#8364; € (no leading “x”)

Character references do not need to be defined.

## 1.8 CDATA Sections

*CDATA sections* are used to handle blocks of text containing markup characters. A CDATA section begins with “<![CDATA[” and ends with “]]>”.

Within a CDATA section, only the string “]]>” is recognized as markup. Other markup characters are treated as *character data*:

- <![CDATA[<greeting>Hello,world!</greeting>]]>
- <![CDATA[</greeting>Hello,world!<greeting>]]>

CDATA sections cannot nest, and cannot overlap with other CDATA sections or markup. In a CDATA section, XML processors ignore markup characters other than “]]>”.

This can sometimes be useful for embedding data containing characters that would normally be treated as markup: wrapping it in a CDATA section causes the parser to treat those characters as character data:

- <code><![CDATA[if (i > 0 && i < 10) { i++; }]]> </code>

However, CDATA sections can only contain legal XML characters (not arbitrary binary!), and it is necessary to ensure that there are no occurrences of the string “]]>” in the embedded data.

# TOPIC:3 DTD and schemas

## 3.1 XML DTD

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

### What is a DTD?

- DTD stands for Document Type Definition.
- A DTD defines the structure and the legal elements and attributes of an XML document.

### Valid XML Documents

- A "Valid" XML document is "Well Formed", as well as it conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

- The DOCTYPE declaration above contains a reference to a DTD file. The content of the DTD file is shown and explained below.

- The purpose of a DTD is to define the structure and the legal elements and attributes of an XML document:

Note.dtd:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

The DTD above is interpreted like this:

- !DOCTYPE note - Defines that the root element of the document is note
- !ELEMENT note - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element to be of type "#PCDATA"
- !ELEMENT heading - Defines the heading element to be of type "#PCDATA"
- !ELEMENT body - Defines the body element to be of type "#PCDATA"

#PCDATA means parseable character data.

### When to Use a DTD?

- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- With a DTD, you can verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

### When NOT to Use a DTD?

- XML does not require a DTD.
- When you are experimenting with XML, or when you are working with small XML files, creating DTDs may be a waste of time.
- If you develop applications, wait until the specification is stable before you add a DTD. Otherwise, your software might stop working because of validation errors.

## 3.2 XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".
- XML Schema is an XML-based alternative to DTD:

```
<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>
```

The Schema above is interpreted like this:



- `<xs:element name="note">` defines the element called "note"
- `<xs:complexType>` the "note" element is a complex type
- `<xs:sequence>` the complex type is a sequence of elements
- `<xs:element name="to" type="xs:string">` the element "to" is of type string (text)
- `<xs:element name="from" type="xs:string">` the element "from" is of type string
- `<xs:element name="heading" type="xs:string">` the element "heading" is of type string
- `<xs:element name="body" type="xs:string">` the element "body" is of type string

## **XML Schemas are More Powerful than DTD**

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

### **Why Use an XML Schema?**

- With XML Schema, your XML files can carry a description of its own format.
- With XML Schema, independent groups of people can agree on a standard for interchanging data.
- With XML Schema, you can verify data.

### **XML Schemas Support Data Types**

One of the greatest strengths of XML Schemas is the support for data types:

- It is easier to describe document content
- It is easier to define restrictions on data
- It is easier to validate the correctness of data
- It is easier to convert data between different data types

### **XML Schemas use XML Syntax**

Another great strength about XML Schemas is that they are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT

### **Displaying XML with XSLT**

- XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
- XSLT is far more sophisticated than CSS. 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.
- XSLT uses XPath to find information in an XML document.

# **TOPIC:4 Transforming XML using XSL and XSLT**

## **What is XSL?**

- XSL stands for **EX**tensible **S**tylesheet **L**anguage.
- XSL = Style Sheets for XML
- 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

## 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.

## Explain XSL Transformation and XSL Elements

- The style sheet provides the template that transforms the document from one structure to another.
- In this case `<xsl:template>` starts the definition of the actual template, as the root of the source XML document.
- The `match = "/"` attribute makes sure this begins applying the template to the root of the source XML document.

### Linking

- The style sheet is linked into the XML by adding the connecting statement to the XML document:  
`<?xml-stylesheet type="text/xsl" href="libstyle.xsl" ?>`

### 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.
- XSLT Works as:
  - In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates.
  - When a match is found, XSLT will transform the matching part of the source document into the result document.

### XSL Elements

- XSL contains many elements that can be used to manipulate, iterate and select XML, for output.
  - value-of

- for-each
- sort
- if
- choose

### **<xsl:value-of> Element**

- The <xsl:value-of> element extracts the value of a selected node.
- The <xsl:value-of> element can be used to select the value of an XML element and add it to the output.

#### **Syntax**

`<xsl:value-of select="expression" />`

- **expression:** This is Required. An XPath expression that specifies which node/attribute to extract the value from. It works like navigating a file system where a forward slash (/) selects subdirectories.

### **<xsl:for-each> Element**

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

### **<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>`

### **<xsl:sort> Element**

- The <xsl:sort> element is used to sort the output.
- `<xsl:sort select="artist"/>`
- The select attribute indicates what XML element to sort on.

### **Example using value-of, for-each and if**

```
<?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 CD Collection</h2>

<table border="1">

  <tr bgcolor="#9acd32">
```

```

        <th>Title</th>

        <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
        <xsl:if test="price > 10">
            <tr>
                <td>

                    <xsl:value-of select="title"/>

                </td>
                <td><

                    xsl:value-of select="artist"/>

                </td>
            </tr>
        </if>
    </for-each>

```

```

    </xsl:if>
  </xsl:for-
    each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

### **<xsl:choose> Element**

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

#### **Syntax**

```

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

```

### **<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:

```

<?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 CD Collection</h2>

<xsl:apply-templates/>

</body>

</html>

</xsl:template>

<xsl:template match="cd">

<p>

<xsl:apply-templates select="title"/>

<xsl:apply-templates select="artist"/>

</p>

</xsl:template>

</xsl:stylesheet>

```

## Explain transforming with XSLT.

- It is possible to convert an XML document to XHTML using the browser's own parser. However, this is not always possible:
  - The browser at the client end may not be suitable or equipped to do the transformation.
  - It may not be a good idea to include the reference to the style sheet or even have the style sheet available!
- The answer to this process the document and style sheet outside of the browser's own mechanism for doing this task.
- This task can be done either on the client side or the server side.

### Using JavaScript

- One way to process and transform XML on the client side is using JavaScript, which has several features for doing the task very well.

```

<html>

<body>

<script type="text/javascript">

//Load the XML document

var xml= new

ActiveXObject("Microsoft.XMLDOM")

xml.async=false

```

```

xml.load("lib.xml")

//Load the XSL document

var xsl= new
ActiveXObject("Microsoft.XMLDOM")
xsl.async= false

xsl.load("libstyle.xsl")

//Do the actual transform
document.write(xml.transfo
rmNode(xsl))

</script>

</body>

</html>

```

- Above example shows one way to transform with JavaScript using Microsoft's proprietary Application Programming Interface (API) for the Internet Explorer browser.
- It is also possible to process XML using the DOM.
- Using both the of these mechanisms it is possible to also traverse an XML document and process either according to a style sheet or simply using the JavaScript to make the stylistic decisions.
- Apart from JavaScript, it is also possible to use other programming languages (such as Java and .Net) to process and then output a transformed document.