XML

Tutorial

Simply Easy Learning

www.tutorialspoint.com

SIMPLE EASY LEARNING

About the tutorial

XML Tutorial

This tutorial provides you the basic understanding of Extensible Markup Language and its features.

Audience

This tutorial is designed for the readers pursuing education in software development and Web development domain and for all the enthusiastic readers.

Prerequisites

This tutorial is designed and developed for absolute beginners. Though, awareness of Web browsers, handling of webpages, software development process and computer fundamentals would be beneficial.

Copyright & Disclaimer

1 XML Overview

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.

- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So *what exactly is a markup language?* Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
   <text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>...</text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instructs computer to perform specific tasks. XML does not qualify to be a programming language as it does not

perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

XML Syntax

2

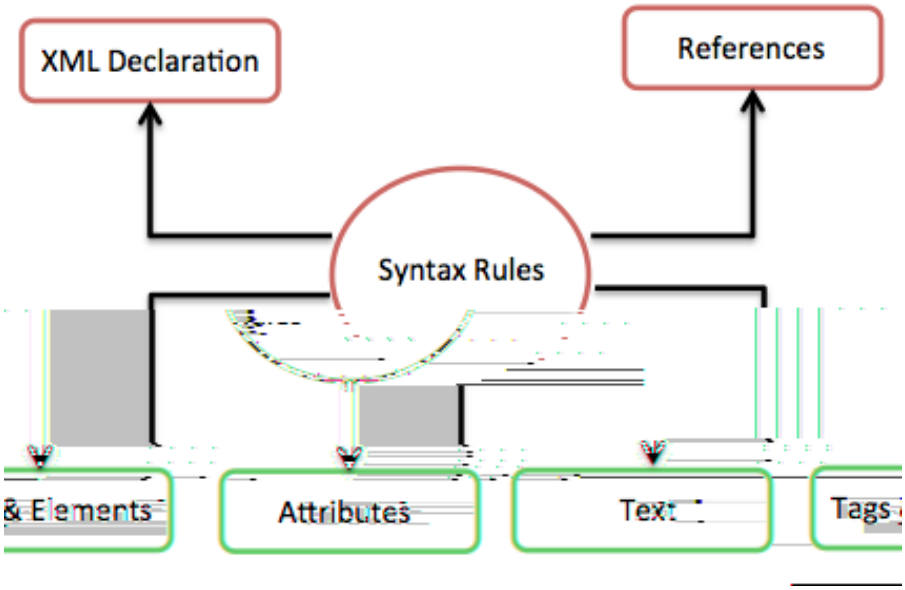
This chapter takes you through the simple syntax rules to write an XML document. Following is a complete XML document:

```
<?xml version="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

You can notice there are two kinds of information in the above example:

- The markup, like *<contact-info>* and
- The text, or the character data, *Tutorials Point* and *(040) 123-4567*.

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



Let us see each component of the above diagram in detail:

XML Declaration

The XML document can optionally have an XML declaration. It is written as below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

Syntax Rules for XML declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets < > as shown below:

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint
<contact-info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint</company>
<contact-info>
```

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
   <x>...</x>
   <y>...</y>
</root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>** is different from **<Contact-Info>**.

Attributes

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example:

```
<a href="http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

Here, *href* is the attribute name and *http://www.tutorialspoint.com/* is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice:

```
<a b="x" c="y" b="z">....</a>
```

Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax:

```
<a b=x>....</a>
```

In the above syntax, the attribute value is not defined in quotation marks.

XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol **"&"**, which is a reserved character and end with the symbol **";"**. XML has two types of references:

Entity References: An entity reference contains a name between the start and the end delimiters. For example **&** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

Character References: These contain references, such as **A**, contains a hash mark (“#”) followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

XML Text

The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.

To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.

Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.

Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below:

not allowed character	replacement-entity	character description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML Documents

3

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document example

A simple document is given in the following example:

```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

The following image depicts the parts of XML document.

Document Prolog Section

The **document prolog** comes at the top of the document, before the root element. This section contains:

- XML declaration
- Document type declaration

You can learn more about XML declaration in chapter XML Declaration.

Document Elements Section

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

You can learn more about XML elements in chapter XML Elements.

XML Declaration

4

This chapter covers XML declaration in detail. XML declaration contains details that prepare an XML processor to parse the XML document. It is optional, but when it is used, it must appear in first line of the XML document.

Syntax

Following syntax shows XML declaration:

```
<?xml
   version="version_number"
   encoding="encoding_declaration"
   standalone="standalone_status"
?>
```

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote. Following table shows the above syntax in detail:

Parameter	Parameter_value	Parameter_description
Version	1.0	Specifies the version of the XML standard used.

Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	*yes* or *no*.	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to *no*. Setting it to *yes* tells the processor there are no external declarations required for parsing the document.

Rules

An XML declaration should abide with the following rules:

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is:*version, encoding and standalone.*
- Either single or double quotes may be used.

- The XML declaration has no closing tag i.e. </?xml>

XML Declaration Examples

Following are few examples of XML declarations:

XML declaration with no parameters:

```
<?xml >
```

XML declaration with version definition:

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

XML declaration with all parameters defined:

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

XML declaration with all parameters defined in single quotes:

```
<?xml version='1.0' encoding='iso-8859-1' standalone='no' ?>
```

XML Tags

5

Let us learn about one of the most important part of XML, the *XML tags*. XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

We can broadly categorize XML tags as follows:

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. An example of start-tag is:

```
<address>
```

End Tag

Every element that has a start tag should end with an end-tag. An example of end-tag is:

```
</address>
```

Note that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called *content*. An element which has no content is termed as **empty**. An **empty** element can be represented in two ways as below:

(1) A start-tag immediately followed by an end-tag as shown below:

```
<hr></hr>
```

(2) A complete empty-element tag is as shown below:

```
<hr />
```

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags:

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

```
<address>This is wrong syntax</Address>
```

Following code shows a correct way, where we use the same case to name the start and the end tag.

```
<address>This is correct syntax</address>
```

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example:

```
<outer_element>

   <internal_element>

      This tag is closed before the outer_element

   </internal_element>

</outer_element>
```

XML Elements

6

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax

Following is the syntax to write an XML element:

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where

- **element-name** is the name of the element. The *name* its case in the start and end tags must match.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:

  ```
  name = "value"
  ```

  The *name* is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax:

```
<name attribute1 attribute2.../>
```

Example of an XML document using various XML element:

```
<?xml version="1.0"?>
<contact-info>
   <address category="residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   <address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements:

- An element *name* can contain any alphanumeric characters. The only punctuation marks allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

XML Attributes

7

This chapter describes about the XML attributes. Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a *name-value* pair.

Syntax

An XML attribute has following syntax:

```
<element-name attribute1 attribute2 >
....content..
< /element-name>
```

where *attribute1* and *attribute2* has the following form:

```
name = "value"
```

The *value* has to be in double (" ") or single (' ') quotes. Here, *attribute1* and *attribute2* are unique attribute labels.

Attributes are used to add a unique label to an element, place the label in a category, add a Boolean flag, or otherwise associate it with some string of data.

Following example demonstrates the use of attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE garden [
```

```
   <!ELEMENT garden (plants)*>
   <!ELEMENT plants (#PCDATA)>
   <!ATTLIST plants category CDATA #REQUIRED>
]>
<garden>
   <plants category="flowers" />
   <plants category="shrubs">
   </plants>
</garden>
```

Attributes are used to distinguish among elements of the same name. When you do not want to create a new element for every situation. Hence, use of an attribute can add a little more detail in differentiating two or more similar elements.

In the above example we have categorized the plants by including attribute *category* and assigning different values to each of the elements. Hence we have two categories of *plants*, one *flowers* and other *color*. Hence we have two plant elements with different attributes.

You can also observe that we have declared this attribute at the beginning of the XML.

Attribute Types

Following table lists the type of attributes:

Attribute Type	Description
StringType	It takes any literal string as a value. CDATA is a StringType. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.

TokenizedType	This is more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized. The TokenizedType attributes are given as: **ID:** It is used to specify the element as unique. **IDREF:** It is used to reference an ID that has been named for another element. **IDREFS:** It is used to reference all IDs of an element. **ENTITY:** It indicates that the attribute will represent an external entity in the document. **ENTITIES:** It indicates that the attribute will represent external entities in the document. **NMTOKEN:** It is similar to CDATA with restrictions on what data can be part of the attribute. **NMTOKENS:** It is similar to CDATA with restrictions on what data can be part of the attribute.
EnumeratedType	This has a list of predefined values in its declaration. out of which, it must assign one value. There are two types of enumerated attribute: **NotationType:** It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document.

	Enumeration: Enumeration allows you to define a specific list of values that the attribute value must match.

Element Attribute Rules

Following are the rules that need to be followed for attributes:

- An attribute name must not appear more than once in the same start-tag or empty-element tag.
- An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.
- Attribute values must not contain direct or indirect entity references to external entities.
- The replacement text of any entity referred to directly or indirectly in an attribute value must not contain either less than sign <

XML Comments

8

This chapter explains how comments work in XML documents. XML comments are similar to HTML comments. The comments are added as notes or lines for understanding the purpose of an XML code.

Comments can be used to include related links, information and terms. They are visible only in the source code; not in the XML code. Comments may appear anywhere in XML code.

Syntax

XML comment has following syntax:

```
<!-------Your comment----->
```

A comment starts with <!-- and ends with -->. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

Example

Following example demonstrates the use of comments in XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!---Students grades are uploaded by months---->
<class_list>
   <student>
```

```
      <name>Tanmay</name>
      <grade>A</grade>
   </student>
</class_list>
```

Any text between <!--

XML Character Entities

9

This chapter describes the XML Character Entities. Before we understand the Character Entities, let us first understand what an XML entity is.

As put by W3 Consortium the definition of entity is as follows:

> *"The document entity serves as the root of the entity tree and a starting-point for an XML processor."*

This means, entities are the placeholders in XML. These can be declared in the document prolog or in a DTD. There are different types of entities and this chapter will discuss Character Entity.

Both, the HTML and the XML, have some symbols reserved for their use, which cannot be used as content in XML code. For example, < and > signs are used for opening and closing XML tags. To display these special characters, the character entities are used.

There are few special characters or symbols which are not available to be typed directly from keyboard. Character Entities can be used to display those symbols/special characters also.

Types of Character Entities

There are three types of character entities:

- Predefined Character Entities

- Numbered Character Entities
- Named Character Entities

Predefined Character Entities

They are introduced to avoid the ambiguity while using some symbols. For example, an ambiguity is observed when less than (<) or greater than (>) symbol is used with the angle tag (<>). Character entities are basically used to delimit tags in XML. Following is a list of pre-defined character entities from XML specification. These can be used to express characters without ambiguity.

- Ampersand: &
- Single quote: '
- Greater than: >
- Less than: <
- Double quote: "

Numeric Character Entities

The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format. As there are thousands of numeric references available, these are a bit hard to remember. Numeric reference refers to the character by its number in the Unicode character set.

General syntax for decimal numeric reference is:

```
&# decimal number ;
```

General syntax for hexadecimal numeric reference is:

```
&#x Hexadecimal number ;
```

The following table lists some predefined character entities with their numeric values:

Entity name	Character	Decimal reference	Hexadecimal reference
quot	"	"	"
amp	&	&	&
apos	'	'	'
lt	<	<	<
gt	>	>	>

Named Character Entity

As it is hard to remember the numeric characters, the most preferred type of character entity is the named character entity. Here, each entity is identified with a name.

For example:

- 'Acute' represents capital Á character with acute accent.
- 'ugrave' represents the small ù with grave accent.

XML CDATA Sections

10

This chapter discusses the XML CDATA section. The term CDATA means, Character Data. CDATA are defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

The predefined entities such as <, >, and & require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Syntax

Following is the syntax for CDATA section:

```
<![CDATA[
   characters with markup
]]>
```

The above syntax is composed of three sections:

- **CDATA Start section -** CDATA begins with the nine-character delimiter **<![CDATA[**
- **CDATA End section -** CDATA section ends with **]]>** delimiter.

- **CData section -** Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

```
<script>
<![CDATA[
   <message> Welcome to TutorialsPoint </message>
]] >
</script >
```

In the above syntax, everything between <message> and </message> is treated as character data and not as markup.

CDATA Rules

The given rules are required to be followed for XML CDATA:

- CDATA cannot contain the string "]]>" anywhere in the XML document.
- Nesting is not allowed in CDATA section.

XML Whitespaces

11

This chapter discusses white space handling in XML documents. Whitespace is a collection of spaces, tabs, and newlines. They are generally used to make a document more readable.

XML document contain two types of white spaces **(a)** Significant Whitespace and**(b)** Insignificant Whitespace. Both are explained below with examples.

Significant Whitespace

A significant Whitespace occurs within the element which contain text and markup present together. For example:

```
<name>TanmayPatil</name>
```

and

```
<name>Tanmay Patil</name>
```

The above two elements are different because of the space between **Tanmay** and **Patil**. Any program reading this element in an XML file is obliged to maintain the distinction.

Insignificant Whitespace

Insignificant whitespace means the space where only element content is allowed. For example:

```
<address.category="residence">
```

or

```
<address....category="..residence">
```

The above two examples are same. Here, the space is represented by dots (**.**). In the above example, the space between *address* and *category* is insignificant.

A special attribute named **xml:space** may be attached to an element. This indicates that whitespace should not be removed for that element by the application. You can set this attribute to **default** or **preserve** as shown in the example below:

```
<!ATTLIST address  xml:space (default|preserve) 'preserve'>
```

Where:

- The value **default** signals that the default whitespace processing modes of an application are acceptable for this element;
- The value **preserve** indicates the application to preserve all the whitespaces.

XML Processing

12

This chapter describes the Processing Instructions (PIs). As defined by the XML 1.0 Recommendation,

> *"Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application."*

Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Syntax

Following is the syntax of PI:

```
<?target instructions?>
```

Where:

- **target** - identifies the application to which the instruction is directed.
- **instruction** - it is a character that describes the information for the application to process.

A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string ?> is encountered.

Example

PIs are rarely used. They are mostly used to link XML document to a style sheet. Following is an example:

```
<?xml-stylesheet href="tutorialspointstyle.css" type="text/css"?>
```

Here, the *target* is an xml-stylesheet. *href="tutorialspointstyle.css"* and *type="text/css"*are *data* or *instructions* that the target application will use at the time of processing the given XML document.

In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown; the first attribute states that the type of the transform is XSL and the second attribute points to its location.

Processing Instructions Rules

A PI can contain any data except the combination ?>, which is interpreted as the closing delimiter. Here are two examples of valid PIs:

```
<?welcome to pg=10 of tutorials point?>


<?welcome?>
```

XML Encoding

13

Encoding is the process of converting unicode characters into their equivalent binary representation. When the XML processor reads an XML document, it encodes the document depending on the type of encoding. Hence, we need to specify the type of encoding in the XML declaration.

Encoding Types

There are mainly two types of encoding:

- UTF-8
- UTF-16

UTF stands for *UCS Transformation Format*, and UCS itself means Universal Character Set. The number 8 or 16 refers to the number of bits used to represent a character. They are either 8(one byte) or 16(two bytes). For the documents without encoding information, UTF-8 is set by default.

Syntax

Encoding type is included in the prolog section of the XML document. The syntax for UTF-8 encoding is as below:

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

Syntax for UTF-16 encoding:

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
```

Example

Following example shows declaration of encoding:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

In the above example, **encoding="UTF-8"**, specifies that 8-bits are used to represent the characters. To represent 16-bit characters, **UTF-16** encoding can be used.

The XML files encoded with UTF-8 tend to be smaller in size than those encoded with UTF-16 format.

XML Validation

13

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are:

- Well-formed XML document
- Valid XML document

Well-formed XML document

An XML document is said to be **well-formed** if it adheres to the following rules:

- Non DTD XML files must use the predefined character entities for **amp(&)**,**apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Example

Example of well-formed XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address
[
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Above example is said to be well-formed as:

- It defines the type of document. Here, the document type is **element** type.
- It includes a root element named as **address**.
- Each of the child elements among name, company and phone is enclosed in its self-explanatory tag.
- Order of the tags is maintained.

Valid XML document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter XML - DTDs.

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Begin syntax for DTD is as follows

```
<!DOCTYPE element DTD identifier
```

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
```

```
</address>
```

Let us go through the above code:

Start Declaration- Begin the XML declaration with following statement

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

DTD- Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.

- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
```

```
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

SYSTEM IDENTIFIERS

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

PUBLIC IDENTIFIERS

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
            <xs:element name="phone" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
            <xs:element name="phone" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
           <xs:element name="company" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="AddressType" />
           <xs:element name="phone1" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Address2">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="AddressType" />
           <xs:element name="phone2" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
```

```
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:

```
<xs:attribute name="x" type="y"/>
```

An XML document is always descriptive. The tree structure is often referred to as XML Tree and plays an important role to describe any XML document easily.

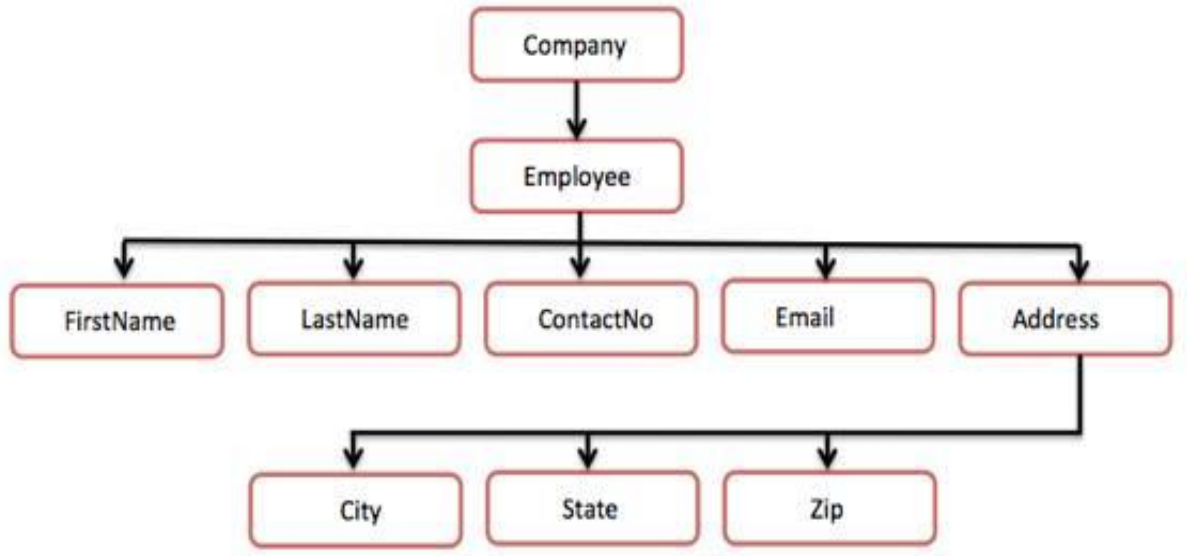
The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root. The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes.

Example

Following example demonstrates simple XML tree structure:

```
<?xml version="1.0"?>
<Company>
  <Employee>
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
      <Email>tanmaypatil@xyz.com</Email>
      <Address>
           <City>Bangalore</City>
           <State>Karnataka</State>
           <Zip>560212</Zip>
      </Address>
  </Employee>
</Company>
```

Following tree structure represents the above XML document:



In the above diagram, there is a root element named as <company>. Inside that, there is one more element <Employee>. Inside the employee element, there are five branches named <FirstName>, <LastName>, <ContactNo>, <Email>, and <Address>. Inside the <Address> element, there are three sub-branches, named <City> <State> and <Zip>.

The Document Object Model (DOM) is the foundation of XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a way of describing those nodes and the relationships between them.

A DOM Document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be *tree based*.

The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

Example

The following example (sample.htm) parses an XML document ("address.xml") into an XML DOM object and then extracts some information from it with JavaScript:

```
<!DOCTYPE html>
<html>
   <body>
      <h1>TutorialsPoint DOM example </h1>
      <div>
         <b>Name:</b> <span id="name"></span><br>
         <b>Company:</b> <span id="company"></span><br>
         <b>Phone:</b> <span id="phone"></span>
      </div>
```

```
      <script>
         if (window.XMLHttpRequest)
         {// code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
         }
         else
         {// code for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/xml/address.xml",false);
         xmlhttp.send();
         xmlDoc=xmlhttp.responseXML;

         document.getElementById("name").innerHTML=
         xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
         document.getElementById("company").innerHTML=
         xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
         document.getElementById("phone").innerHTML=
         xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
      </script>
   </body>
</html>
```

Contents of **address.xml** are as below:

```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Now let us keep these two files **sample.htm** and **address.xml** in the same directory **/xml** and execute the **sample.htm** file by opening it in any browser. This should produce an output as shown below:

Here, you can see how each of the child nodes is extracted to display their values.

A Namespace is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to group. The Namespace is identified by Uniform Resource Identifier (URI).

Namespace Declaration

A Namspace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below:

```
<element xmlns:name="URL">
```

Syntax

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.tutorialspoint.com/profile">
   <cont:name>Tanmay Patil</cont:name>
   <cont:company>TutorialsPoint</cont:company>
   <cont:phone>(011) 123-4567</cont:phone>
```

```
</cont:contact>
```

Here, the Namespace prefix is **cont**, and the Namespace identifier (URI) as*www.tutorialspoint.com/profile*. This means, the element names and attribute names with the **cont** prefix (including the contact element), all belong to the*www.tutorialspoint.com/profile* namespace.

XML Database is used to store the huge amount of information in the XML format. As the use of XML is increasing in every field, it is required to have the secured place to store the XML documents. The data stored in the database can be queried using **XQuery**, serialized, and exported into desired format.

XML Database Types

There are two major types of XML databases:

- XML- enabled
- Native XML (NXD)

XML- Enabled Database

XML enabled database is nothing but the extension provided for the conversion of XML document. This is relational database, where data are stored in tables consisting of rows and columns. The tables contain set of records, which in turn consist of fields.

Native XML Database

Native XML database is based on the container rather than table format. It can store large amount of XML document and data. Native XML database is queried by the **XPath**-expressions. Native XML database has advantage over the XML-enabled database. It is highly capable to store, query and maintain the XML document than XML-enabled database.

Example

Following example demonstrates XML database:

```
<?xml version="1.0"?>
<contact-info>
```

```
   <contact1>
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </contact1>
   <contact2>
      <name>Manisha Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 789-4567</phone>
   </contact2>
</contact-info>
```

Here, a table of contacts is created that holds the records of contacts (contact1 and contact2), which in turn consists of three entities - *name, company* and *phone*.

XML Viewers

21

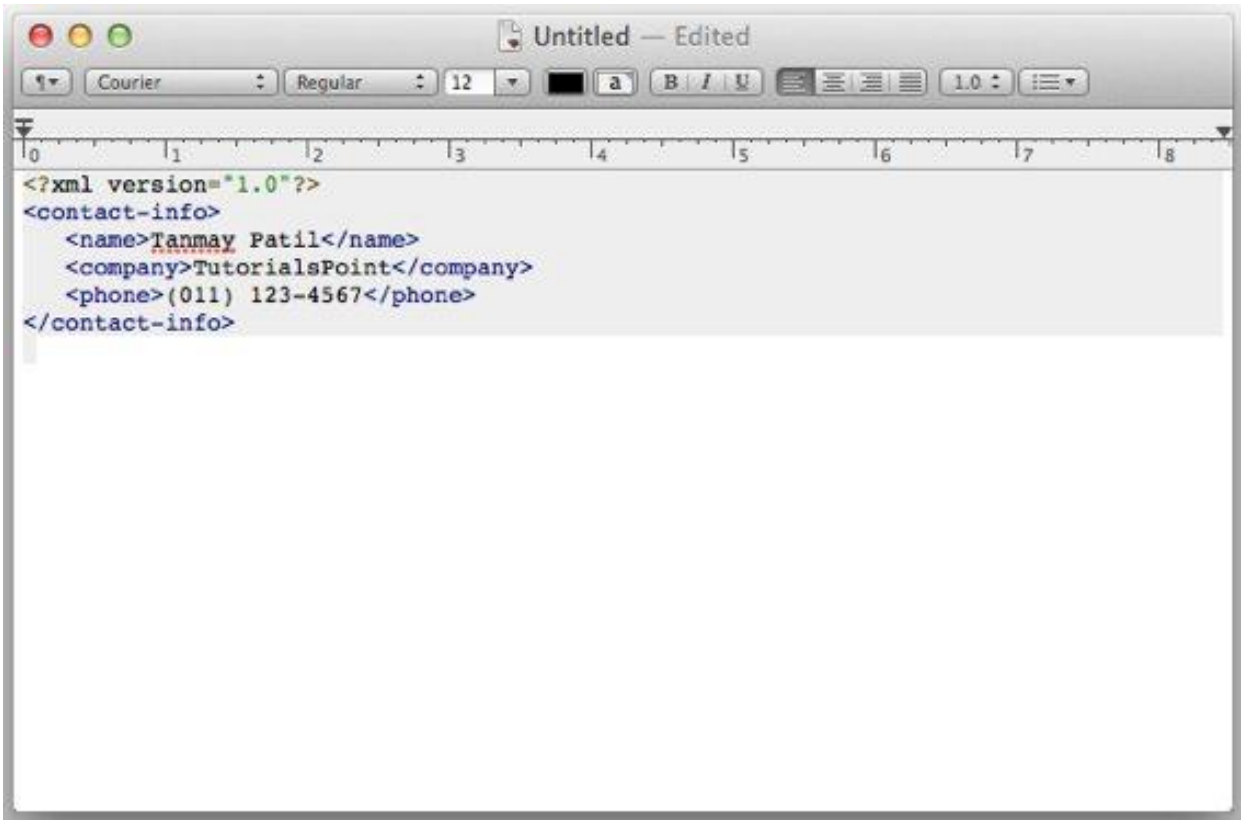
This chapter describes various methods to view an XML document. An XML document can be viewed using a simple text editor or any browser. Most of the major browsers supports XML. XML files can be opened in browser by just double clicking on the XML document (if it is a local file) or by typing the URL path in the address bar (if the file is located on server), in the same way as we open other files in the browser. XML files are saved with a ".xml" extension.

Let us explore various methods by which we can view an XML file. Following example (sample.xml) is used to view in all the sections of this chapter.

```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Text Editors

Any simple text editor such as Notepad, Textpad or TextEdit can be used to create or view an XML document as shown below:



Firefox Browser

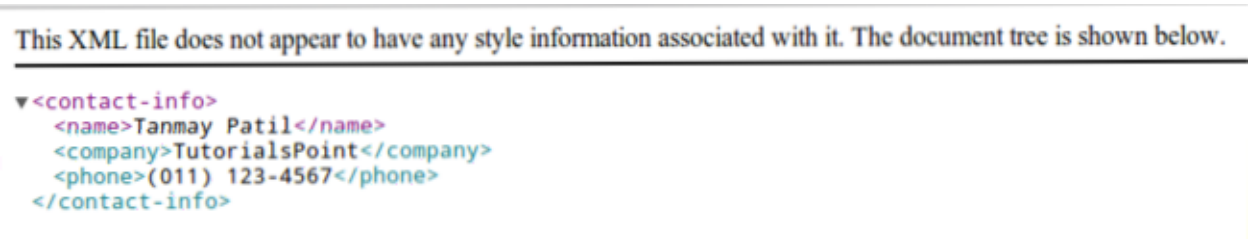
Open the above XML code in chrome by double clicking on the file, the XML code displays coding with colour, which makes the code readable. It shows plus(+) or minus (-) sign at the left side in the XML element. When we click on the minus sign(-), the code hides and by clicking on plus(+) sign the code lines get expanded. The output in Firefox is as shown below:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <contact-info>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
  </contact-info>
```

Chrome Browser

Open the above XML code in a chrome browser. The code gets displayed as shown below:

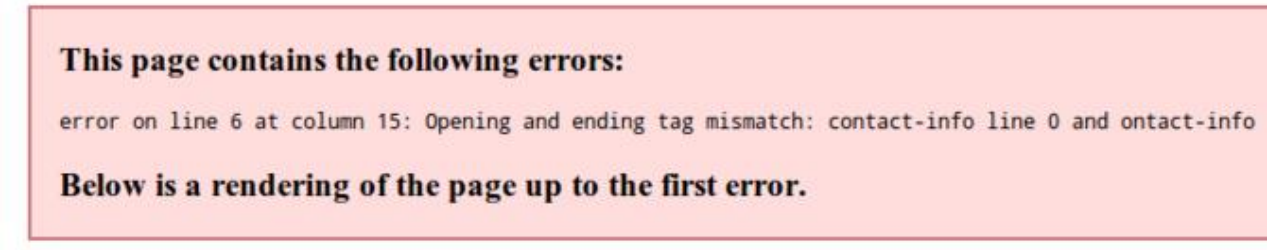


Errors in XML Document

If your XML code has some tags missing then a message is displayed in the browser. Let us try to open the following XML file in chrome:

```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</ontact-info>
```

In the code above, the start and end tags are not matching (refer the *contact_info* tag), hence the an error message is displayed by the browser as shown below:



Tanmay Patil TutorialsPoint (011) 123-4567

XML Editors

22

XML Editor is a markup language editor. The XML documents can be edited or created using existing editors such as *Notepad, Wordpad* or any similar text editor. You can also find a professional XML editor online or for downloading, which has more powerful editing features such as:

- It automatically closes the tags that are left open.
- It strictly checks syntax.
- It highlights XML syntax with colour for increased readability.
- It helps you to write a valid XML code.
- It provides automatic verification of XML documents against DTDs and Schemas.

Open Source XML Editors

There are some open source XML editors given below:

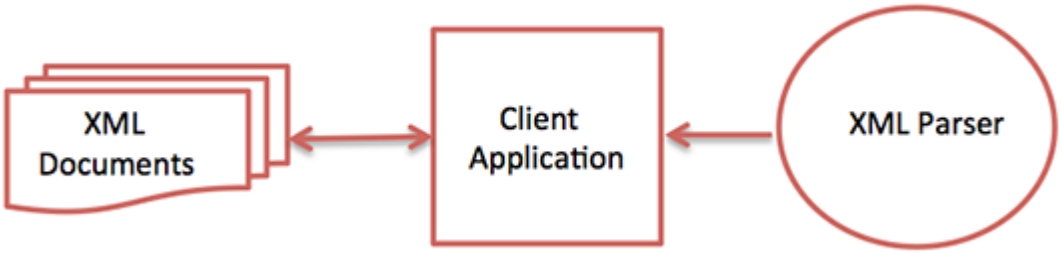
- **Xerlin**: Xerlin is an open source XML editor for the Java 2 platform released under an Apache license. It is a Java based XML modelling application, for creating and editing XML files easily.
- **CAM - Content Assembly Mechanism**: CAM XML Editor tool with XML+JSON+SQL Open-XDX sponsored by Oracle.

XML Parsers

23

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document:



The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below:

MSXML (Microsoft Core XML Services) : This is a standard set of XML tools from Microsoft that includes a parser.

- **System.Xml.XmlDocument :** This class is part of .NET library, which contains a number of different classes related to working with XML.

- **Java built-in parser :** The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon :** Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces :** Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

XML Processors

24

When a software program reads an XML document and takes actions accordingly, this is called *processing* the XML. Any program that can read and process XML documents is known as an *XML processor*. An XML processor reads an XML file and turns it into in-memory structures that the rest of the program can access.

The most fundamental XML processor reads an XML documents and converts it into an internal representation for other programs or subroutines to use. This is called a *parser*, and it is an important component of every XML processing program.

Processor involves processing the instructions that can be studied in the chapter XML - Processing Instruction.

Types

XML processors are classified as **validating** or **non-validating** types, depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue with normal processing.

A few validating parsers are: xml4c (IBM, in C++), xml4j (IBM, in Java), MSXML (Microsoft, in Java), TclXML (TCL), xmlproc (Python), XML::Parser (Perl), Java Project X (Sun, in Java).

A few non-validating parsers are: OpenXML (Java), Lark (Java), xp (Java), AElfred (Java), expat (C), XParse (JavaScript), xmllib (Python).