

# XML - QUICK GUIDE

[http://www.tutorialspoint.com/xml/xml\\_quick\\_guide.htm](http://www.tutorialspoint.com/xml/xml_quick_guide.htm)

Copyright © tutorialspoint.com

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

XML tags identify the data and used to store and organize data, rather than specifying how to display it like HTML tags are used to display the data. XML is not going to replace HTML in the near but 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:

1. **XML is extensible** : XML essentially allows you to create your own language, or tags, that suits your application.
2. **XML separates data from presentation** : XML allows you to store content with regard to how it will be presented.
3. **XML is a public standard** : XML was developed by an organization called the World Wide Web Consortium *W3C* and available as an open standard.

## XML Usage

A short list of XML's 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 of information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange data in a way that is customizable for your needs.
- XML can easily be mixed with stylesheets to create almost any output desired.
- Virtually any type of data can be expressed as an XML document.

## What is Markup?

XML is not itself a markup language rather it is a set of rules for building markup languages. 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 is an example of how XML markup looks when embedded in a piece of text:

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

This snippet includes the markup symbols , or you can call them tags and they are `<message>...</message>` and `<text>... </text>`. The tags `<message>` and `</message>` mark the start and end points of the whole XML fragment. The tags `<text>` and `</text>` surround the text Hello, world!.

## Is XML a Programming Language?

A programming language is a vocabulary and syntax for instructing a computer to perform specific tasks. XML doesn't qualify as a programming language because it doesn't instruct a computer to do anything, as such. It's usually stored in a simple text file and is processed by special software that's capable of interpreting XML.

# SYNTAX

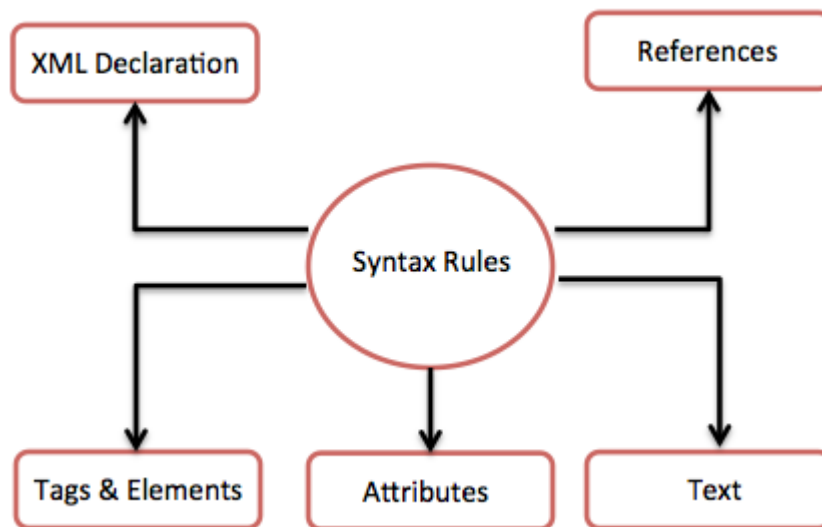
This chapter take you through the simple syntax rules to be followed to write a XML document. Following is a complete *butverysimple* XML document:

```
<?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:

- markup, like `<contact-info>` and
- text *alsoknownascharacterdata*, like `TutorialsPoint` and `011 123-4567`.

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



Let us see each of the above points in detail in the following sections.

## XML Declaration

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

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

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

## Syntax Rules for XML declaration

- The XML declaration is case sensitive i.e it may not begin with "`<?XML`" or any other variant.
- If you specify the XML declaration, then it must be the first statement in the XML document.
- An HTTP protocol can override the encoding value that you put in the XML declaration.

## Tags and Elements

A xml file is structured by several xml-elements, also called xml-nodes or xml-tags. The xml-

element name is enclosed by < > brackets as shown below:

```
<element>
```

## Syntax Rules for Tags and Elements

**1 Element Syntax :** Each xml-element needs to be closed as shown below. Either with start and end elements:

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

or for simple-cases just this way:

```
<element/>
```

**2 Nesting of elements :** A xml-element can contain additional children xml-elements inside it. But the elements may not overlap i.e an end tag must always have the same name as the most recent unmatched start tag.

Following example shows wrong 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>
```

**3 Root element :** A XML document can have only one root element. For example following is not a well-formed XML document, because both the x and y elements occur at the top level:

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

The following example shows a well formed XML document:

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

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

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

## Attributes

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

1 - Attribute names in XML *unlikeHTML* are case sensitive i.e *HREF* and *href* refer to two different XML attributes.

2 - Same attribute cannot have two values. The following example is not well-formed because the *b* attribute is specified twice:

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

3 - An attribute name should not be defined in quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates WRONG xml format:


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

Here 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 character "&" *which is specially reserved* and end with the character ";". XML has two types of references:

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

**2 Character References :** These contain references, like &#65;, contains a hash mark  followed by a number. The number always refers to the Unicode code for a single character. In this case 65 refers to letter "A".

## XML Text

1 - 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.

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

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

4 - Some characters are reserved by the xml syntax itself. Hence they can't be used directly. To use them some replacement-entities can be used which are listed below:

not allowed character	replacement-entity	character description
<	&lt;	less than
>	&gt;	greater than
&	&amp;	ampersand
'	&apos;	apostrophe
"	&quot;	quotation mark

## DOCUMENTS

An XML *document* is a basic unit of XML information, composed of elements and other markup in an orderly package. An XML *document* can contain any data, say for example, database of numbers, or some abstract structure representing a molecule or equation.

### Example

A simple document would look like as in the following example:

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

As can be seen the following image depicts the XML parts.



## Document Prolog Section

The **document prolog** comes at the top of the document, before the root element. This section can further be divided into:

- XML declaration
- Document type declaration

We will learn more about XML declaration while studying [XML Declaration](#).

## Document Elements Section

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

We will learn more about XML elements while studying [XML Elements](#).

# DECLARATION

This chapter will discuss about the XML *declaration*. XML declaration contains details that prepare an XML processor for working with a document. It is optional in use but, when used, it must appear in first line of the XML document.

## Syntax

Following is the syntax to write the 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 is the detail of the above syntax:

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 document. UTF-8 is the default encoding used.
Standalone	It's value is either yes or no.	It informs the parser whether the document relies on information from an external source, such as external document type definition <i>DTD</i> , for its content. The default value is <i>no</i> ; setting it to <i>yes</i> 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 first line in the XML document.
- If the XML declaration is included, it must contain the version number attribute.
- Parameter names and values are case-sensitive.
- The names are always lowercase.
- The order of placing the parameters is important. The correct order is: *version*, *encoding*, *standalone*
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e `</?xml>`

## Examples

Following are few examples of XML declarations:

Example of XML declaration with no parameters defined.

```
<?xml >
```

Example of XML declaration with version defined.

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

Example of XML declaration with all parameters defined.

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

Example of XML declaration with all parameters defined in single quotes.

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

## TAGS

Now we are going to learn about one of the most important part of XML, which is *XML tags*. XML

tags form the base of an XML. They define the boundaries of an element in the XML. They can also be used to insert comments, declare settings for parsing the environment and insert special instructions.

We can 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 using an end-tag. An example of end-tag is:

```
</address>
```

Note that the end tags include a solidus "/" before the element's name.

## Empty Tag

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

1 Start-tag immediately followed by an end-tag as shown below:

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

2 Its a complete empty-element tag 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 for 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 observed in the word "element" which is not allowed and gives error.

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

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

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

### Rule 2

XML tags must be closed in order i.e 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>
```

## ELEMENTS

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

Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags 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* in the start and end tag 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 character data. An attribute will have following syntax:

```
name = "value"
```

*name* is followed by an = sign and a string *value* inside quotes " " or ' '.

### Empty Element

An empty element `element` with no content will have following syntax:

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

Following is a simple example of XML document making use of 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 are the rules that need to be followed for XML elements:

- An element *name* can contain any alphanumeric characters. The only punctuation allowed in names are the hyphen -, under-score \_ and period . .
- Names are case sensitive. For example Address, address, ADDRESS are different names.
- Element start and end tag should be identical.
- An element which is a container can contain text or elements as seen in the above example.

## ATTRIBUTES

This chapter discusses about the XML attributes. Attributes are part of the XML elements. An



element can have any number of unique attributes. Attribute gives more information about the XML element or more precisely it defines a property of the element. 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* will have the following form:

```
name = "value"
```

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

*Attributes* can be used to add a unique label to an element, place it in a category, add a Boolean flag, or otherwise associate some short string of data. Following example demonstrates the use of attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<garden>
  <plants category="flowers">
    <rose>ROSE</rose>
    <lotus>LOTUS</lotus>
  </plants>
  <plants category="color">
    <red>RED</red>
    <pink>PINK</pink>
  </plants>
</garden>
```

Attributes usually are used to distinguish between elements of the same name say when you don't want to create a new element for every situation, so an attribute can add a little more detail in differentiating two or more similar types of 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.

## Attribute Types

Following table lists the type of attributes:

Attribute Type	Description
StringType	May take any literal string as a value. CDATA is a StringType. CDATA is character data. This means that any string of non-markup characters is legal as part of the attribute.
TokenizedType	These are more constrained type. The validity constraints noted in the grammar are applied after the attribute value has been normalized. There are following kinds of TokenizedType attribute types: <ul style="list-style-type: none"><li>• <b>ID</b> : This is used if you want to specify your element as unique.</li><li>• <b>IDREF</b> : This is used to reference an ID that has been named for another element.</li></ul>

- **IDREFS** : This is used to reference all IDs of an element.
- **ENTITY** : This indicates that the attribute will represent an external entity in the document itself.
- **ENTITIES** : This indicates that the attribute will represent an external entities in the document.
- **NMTOKEN** : This is similar to CDATA with even more restrictions on what data can be part of the attribute.
- **NMTOKENS** : This is similar to CDATA with even more restrictions on what data can be part of the attribute.

### EnumeratedType

These have a list of allowed values in their declaration. they must take one of those values. There are two kinds of enumerated attribute types

- **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.
- The attribute must have been declared; the value must be of the type declared for it.
- 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 <

## COMMENTS

This chapter explains how comments works in XML documents. XML comments are similar to HTML comments. Comments are added as notes or lines for understanding purpose of a XML code.

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

### Syntax

XML comment, has the following syntax:

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

Comments starts with <!--and ends with-->. You can add textual notes as a comment between the characters. Never nest one comment with other comment.

### 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 <!-- and --> section is considered as an comment and you can include anything you want.

## XML Comments Rules

Following are the rules that need to be followed for XML comments:

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments cannot appear before an Element tag.
- Comments must not appear within attribute values.
- Comments cannot be nested one inside the another.

## CHARACTER ENTITIES

This chapter discusses the XML **Character Entities**. But before we understand **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.*

Which means entities are 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 on character entity.

**Character Entities** are used to name some of the entities which are symbolic representation of information i.e characters that are difficult or impossible to type can be substituted by Character Entities.

## Types of Character Entities

There are three types of character entities:

- Predefined Character Entities
- Numbered Character Entities
- Named Character Entities

## Predefined Character Entity

These are introduced to avoid the ambiguity while using some of symbols. For example ambiguity observed when less than < or greater than > symbol is used with the angle tag<>. These 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 safely.

- ampersand: &amp;
- Single quote: &apos;
- Greater than: &gt;
- Less than: &lt;

- Double quote: &quot;

## Numeric Character Reference

To refer the character entity numeric reference can be used. Numeric reference can either be in decimal or hexadecimal numbers. As there are thousands of numeric references present, these are bit harder 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 ;
```

Following table list some of the predefined character entities with their numeric values:

Entity name	Character	Decimal reference	Hexadecimal reference
quot	"	&#34;	&#x22;
amp	&	&#38;	&#x26;
apos	'	&#39;	&#x27;
lt	<	&#60;	&#x3C;
gt	>	&#62;	&#x3E;

## Named Character Entity

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

For example:

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

## CDATA SECTIONS

This chapter discusses the XML CDATA section. CDATA means Character Data. CDATA are used to escape block of text that does not parsed by the parser and are otherwise recognized as markup.

Predefined entities such as &amp;lt;, &amp;gt;, and &amp;amp; require typing and are generally hard to read in the markup. For such cases CDATA section can be used. By using CDATA section, you are commanding the parser that this 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  
]]>
```

Details of the above syntax:

- **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

Here is the example of CDATA. Here everything written inside the CDATA section will be ignored by the parser.

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

Here `<message>` and `</message>` are treated as character data and not markup.

## CDATA Rules

Following are the rules that need to be followed for XML CDATA:

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

## WHITE SPACES

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

XML document contain two types of white spaces **a** Significant Whitespace and **b** insignificant Whitespace. Both have been explained below with examples.

### Significant Whitespace

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

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

and

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

These two elements are different because of that space between **Tanmay** and **Patil**, and any program reading this element in an XML file is obliged to maintain the distinction.

### Insignificant Whitespace

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

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

or

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

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

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

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

Where:

- The value **default** signals that applications' default white-space processing modes are acceptable for this element;
- The value **preserve** indicates the intent that applications preserve all the white space.

## PROCESSING

This chapter discusses about the Processing Instructions PIs. As defined by the [XML 1.0 Recommendation](#), Processing Instruction is:

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

Processing instructions PIs can be used to pass information to applications so as to escapes most XML rules. PIs can appear anywhere in the document outside of other 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.
- **instructions** - 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 hardly used. They are known mostly to be used to link XML document to a stylesheet. Following is an example:

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

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

In this case a browser will recognize the target as saying 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 would be interpreted as the closing delimiter. Here are two examples of valid PIs:

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

```
<?welcome?>
```

## ENCODING

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

### Types

There are mainly two types of encoding present:

- UTF-8
- UTF-16

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

### Syntax

Encoding type is included in the prolog section of the XML document. Below is the syntax for UTF-8 encoding.

```
<?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 the 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-bit are used to represent the characters. To represent 16-bit characters **UTF-16** encoding can be used.

## VALIDATION

**Validation** is a process by which an XML document is validated. An XML document is said to be valid if its content matches with the elements, attributes and other piece of an 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**&, **a**single quote, **gt**>, **lt**<, **quot**double quote.

- It must follow the ordering of the tag. i.e. Inner tag must be closed before the outer tag is closed.
- Opening tags must have a closing tag or have themselves self ending. <title>...</title> or <title/>.
- Must have only one Attribute in a start tag, and has to be quoted.
- **amp&, apossingle quote, gt>, lt<, quotdouble 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 document type is **element** type.
- It includes a root element named as **address**.
- Every child elements are enclosed within the self explanatory tags. Here child elements are name, company and phone.
- Order of 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 [XML - DTDs](#).

## DTDS

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

An XML DTD can be specified internally inside the document and it can be kept in a separate document and then linked separately.

## Syntax

Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  .....
]>
```



In the above syntax

- A DTD starts with `<!DOCTYPE` delimiter.
- **element** tells the parser to parse the document from the root element specified.
- **DTD identifier** is identifier for the document type definition which may be a path to the file on the system or URL to the file on the internet. If the DTD is pointing to external path then its called as **external subset**.
- **Square bracket [ ]** encloses an optional list of entity declaration called *internal subset*.

## Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**; which means it will work on its own and does not depend on external source.

## Syntax

Following is the syntax of internal DTD:

```
<!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 with the XML declaration with the following statement

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

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

```
<!DOCTYPE address [
```

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

**DTD Body** Following the DOCTYPE declaration is the body of the DTD. This is 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 the XML document immediately follows.

## 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.
- Like 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 reference it as external DTD, *standalone* attribute in the XML declaration must be set as **no**, which means it depends on 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

Following is a simple example of 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>
```

Below is the content of the DTD file **address.dtd**:

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

## Types

You can refer to an external DTD by either using **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 document's location.

## 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*.

## SCHEMAS

XML Schema commonly known as an XML Schema Definition XSD, and it is used to describe and validate the structure and the content of XML data. XML schemas defines the elements, attributes and data type. Schema element supports the namespaces. It is similar to what a database schema describes the data that can be contained in a database.

### Syntax

You simply have to declare the following XML Schema namespace and assign a prefix such as **xs** in your XML document:

```
<"http://www.w3.org/2001/XMLSchema">
```

### Example

Following is a simple example that 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 [XML - Elements](#) chapter, 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:

**1 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" />
```

**2 Complex Type** - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain. This allows you 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>
```

As you can see in the above example, *Address* element consists of child elements. This is a container for other `<xs:element>` definitions, allowing us to build a simple hierarchy of elements in the resulting XML document.

**3 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 cases 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 in 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 name and 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 your address you only have to add them in one place.

## Attributes

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

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

# TREE STRUCTURE

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

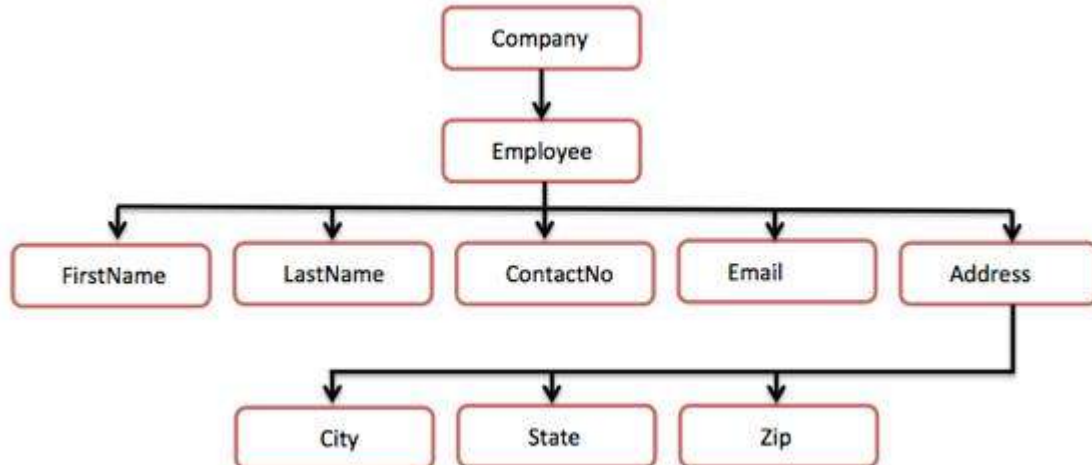
The tree structure contains root parent elements, child elements and so on. By using tree structure you can observe which branch belongs to which root and which sub-branch belongs to which branch. The search or parsing starts at the root, then move down the first branch to an element, take the first branch from there, and so on to the leaves.

## 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:



As can be seen in the above diagram, we have a root element as `<company>` and inside that we can define one more element `<Employee>`. And inside employee element there are 5 branches i.e. `<FirstName>`, `<LastName>`, `<ContactNo>`, `<Email>`, and `<Address>`. And inside the `<Address>` element we have given three sub-branches i.e. `<City>` `<State>` `<Zip>`.

## DOM

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

DOM Document is a collection of nodes, or pieces of information, organized in a hierarchy. This hierarchy allows a developer to navigate around 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 at any point on the tree 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 a JavaScript:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>TutorialsPoint DOM example </h1>
    <div>
      <b>Name:</b> <span ></span><br>
      <b>Company:</b> <span ></span><br>
      <b>Phone:</b> <span ></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's keep these two files `a sample.htm` and `b address.xml` in the same directory **/xml** and execute the **sample.htm** file by opening in any browser. This should produce an output as shown below:

# TutorialsPoint DOM example

**Name:** Tanmay Patil  
**Company:** TutorialsPoint  
**Phone:** (011) 123-4567

Here you can see how we extracted each of the child nodes and displayed their values.

## NAMESPACES

A namespace is a set of names in which all names are unique. Namespaces are a mechanism by which element and attribute names can be assigned to groups. The Namespace is identified by URIUniform Resource Locator.

## Namespace Declaration

A namespace is declared using reserved attributes. Such an attribute's name must either be **xmlns** or begin **xmlns:**.

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

- The namespace starts with the keyword **xmlns**.
- **name** is the namespace prefix.
- and **URI** is the namespace identifier.

## Example

Namespaces only affect 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, we have declared the namespace prefix as **cont**, and the namespace identifier URI as *www.tutorialspoint.com/profile*. This means that element names and attribute names with the **cont** prefix including the *contact* element all belong to the *www.tutorialspoint.com/profile* namespace.

## DATABASES

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

### XML Database Types

There are two major types of XML databases exists:

- XML- enabled
- Native XML NXD

### XML- Enabled Databases

XML enabled databases are nothing but the extensions provided for the conversion of XML document to and from. These are the relational databases, where data are stored in table consisting of rows and columns. The table contains set of records, records in turn consists of fields.

### Native XML Databases

Native XML databases are based on the container rather than table format. They can store large amount of XML document and data. Native XML databases are queried by the **XPath**-expressions.

Native XML databases have advantages over the XML-enabled databases. They are more capable to store, query and maintain the XML document than XML-enabled databases.

## Example

Following example demonstrates the 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, we have created a table of contacts which holds the records of contacts `contact1` and `contact2` which in turn consists of three *name*, *company*, *phone*.

## VIEWERS

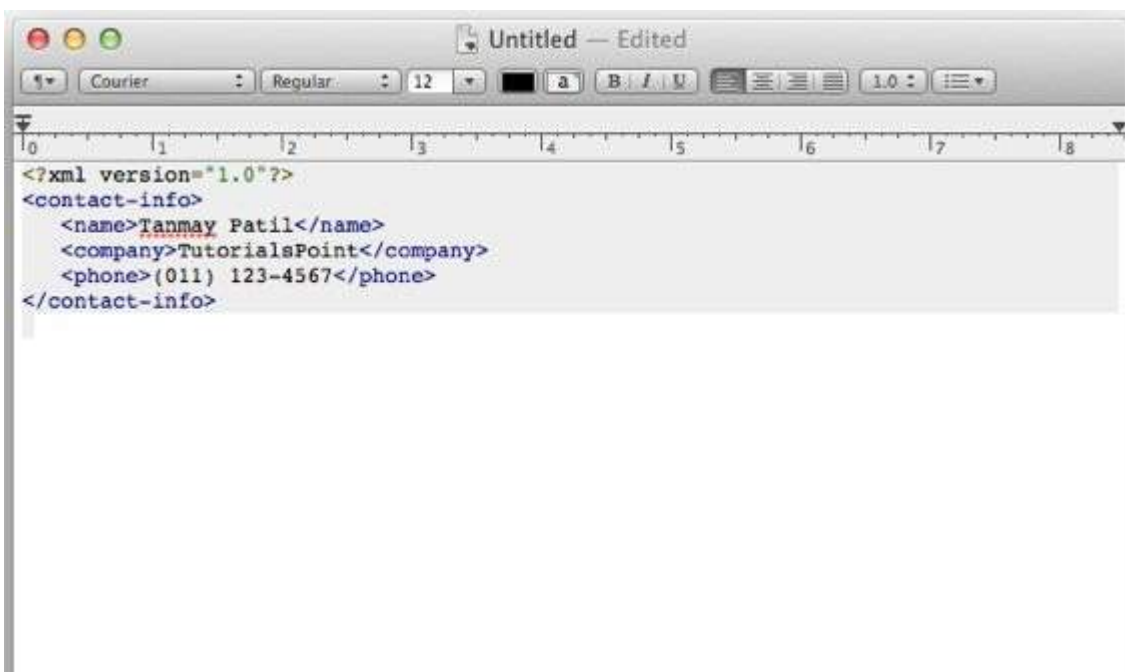
This chapter will discuss of various methods to view an XML document. An XML document can be viewed using a simple text editor or any browser. Most of the major browsers supports XML. XML files can be opened in browser by just double clicking on the XML document if its a local file or by typing the URL path in the address bar if its a file on server, as in 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/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 with color coding which makes the code read easily. 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 that displays 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:

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>
```

## Errors in XML Document

Suppose 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>
```

Here we opening and end tags are not matching, hence the an error message gets displayed on the browser as shown below:

### This page contains the following errors:

error on line 6 at column 15: Opening and ending tag mismatch: contact-info line 0 and ontact-info

Below is a rendering of the page up to the first error.

Tanmay Patil TutorialsPoint (011) 123-4567

## EDITORS

XML Editor is a markup language editor. The XML documents can be edited or created using existing editors such as *notepad*, *wordpad* or *any simple text editor*. You can also find an good professional XML editor online or download, using which you can get many advantages as

followed:

- It automatically close the tags that are been opened.
- The Editor will always assure that the syntax which are been written is always right.
- The XML syntax are highlighted with color code for proper readable.
- It helps you to write a valid XML code.
- Automatic verification of XML documents against DTD, Schemas.

## Open Source XML Editors

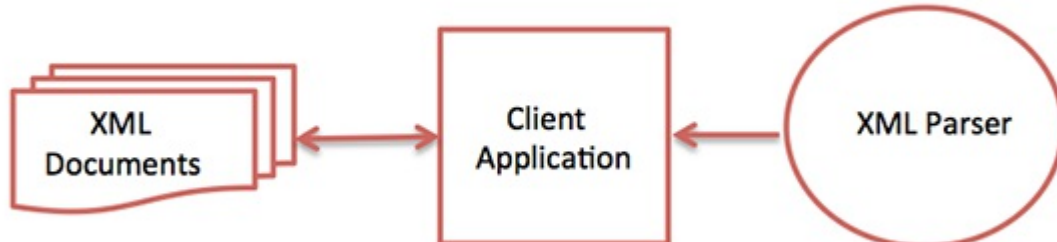
Below we have listed some open source XML editors:

- [Xerlin](#): Xerlin is an open source XML editor for the Java 2 platform released under an Apache style license. The project is a Java based XML modeling application written to make creating and editing XML files easier.
- [CAM - Content Assembly Mechanism](#): CAM XML Editor tool with XML+JSON+SQL Open-XDX sponsored by Oracle.

## PARSERS

XML parser is a software library or a package that provides methods or interfaces 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 an XML parser interacts with XML document:



This diagram shows how a parser works. Its goal 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 Microsoft's standard set of XML tools including a parser.
- **System.Xml.XmlDocument** : This class is part of Microsoft's .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's offerings contain tools for parsing, transforming, and querying XML.
- **Xerces** : Xerces is implemented in Java and is developed by the famous and open source Apache Software Foundation.

## PROCESSORS

When a software program reads an XML document and does something with it, this is called *processing* the XML. Therefore, any program that can read and that can 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 do whatever it likes with.

The most fundamental XML processor reads XML documents and converts them 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 the processing instruction that can be studied in [Processing Instruction chapter](#).

## Types

XML processors are classified as **validating** or **non-validating** 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 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*.

Examples of non-validating parsers are: OpenXML Java, Lark Java, xp Java, AElfred Java, expat C, XParse JavaScript, xml-lib Python

Loading [Mathjax]/jax/element/mml/optable/BasicLatin.js