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.

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 instruct the 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 Declaration

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

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

- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

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.

XML 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, **&amp;** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

- **Character References:** These contain references, such as **&#65;**, 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".

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
<	&lt;	less than
>	&gt;	greater than
&	&amp;	ampersand
'	&apos;	apostrophe
"	&quot;	quotation mark

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 used, it must appear in the first line of the XML document.

Syntax

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

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.

- 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 order of placing the parameters is important. The correct order is: *version, encoding and standalone.*
- The XML declaration has no closing tag, i.e. **</?xml>**

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 follows:

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

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

A complete empty-element tag is as shown below:

```
<hr />
```

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 Rules

Following rules are required to be followed for XML elements:

- An element name can contain any alphanumeric characters. The only punctuation mark 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.

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 a 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: • **Notation Type:** 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 a less than sign (**<**).

Syntax

XML comment has the following syntax:

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

XML Comments Rules

Following rules should be followed for XML comments:

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments must not appear within attribute values.
- Comments cannot be nested inside the other comments.

Types of Character Entities

There are three types of character entities:

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

Predefined Character Entities

- Ampersand: **&amp;**
- Single quote: **&apos;**
- Greater than: **&gt;**
- Less than: **&lt;**
- Double quote: **&quot;**

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:

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

Numeric Character Entities

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	"	&#34;	&#x22;
amp	&	&#38;	&#x26;
apos	'	&#39;	&#x27;
lt	<	&#60;	&#x3C;
gt	>	&#62;	&#x3E;

Insignificant Whitespace

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

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

or

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

The above 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 following example:

```
<!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.

In this chapter, we will discuss **XML CDATA section**. The term CDATA means, Character Data. CDATA is defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

The predefined entities such as **&lt;**, **&gt;**, and **&amp;** 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 an 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.

12. XML – Processing

Syntax

Following is the syntax of PI:

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

Where,

- **target** - Identifies the application to which the instruction is directed.

- **instruction** - A character that describes the information for the application to process.

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"?>
```

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?>
```

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.

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.

15. 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.

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

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **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**.

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>
```

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.

The content of the DTD file **address.dtd** is 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 is written as follows:

```
<!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.

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

```
<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"/>
```

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

Namespace Declaration

A Namespace 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.