1. **Introduction to XML and XHTML 3 hours**
2. **XML, DTD, XSTL, XHTML**

<https://www.w3schools.com/xml>

**What is XML?**

You might think of XML as the Zen of the coding world. Its purpose isn’t to do; it’s simply to be. While XML doesn’t actually do anything, it may be as fundamental to the web as HTML. XML stands for Extensible Markup Language. 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.

**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 −**

1. XML can work behind the scene to simplify the creation of HTML documents for large web sites.
2. XML can be used to exchange the information between organizations and systems.
3. XML can be used for offloading and reloading of databases.
4. XML can be used to store and arrange the data, which can customize your data handling needs.
5. XML can easily be merged with style sheets to create almost any desired output.
6. 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 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.

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

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

**XML Does Not DO Anything**

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Tove from Jani, stored as XML:

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

**The XML above is quite self-descriptive:**

It has sender information.

It has receiver information

It has a heading

It has a message body.

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

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

Note

To: Tove

From: Jani

Reminder

Don't forget me this weekend!

**The Difference Between XML and HTML**

XML and HTML were designed with different goals:

* XML was designed to carry data - with focus on what data is
* HTML was designed to display data - with focus on how data looks
* XML tags are not predefined like HTML tags are

**XML Does Not Use Predefined Tags**

The XML language has no predefined tags.

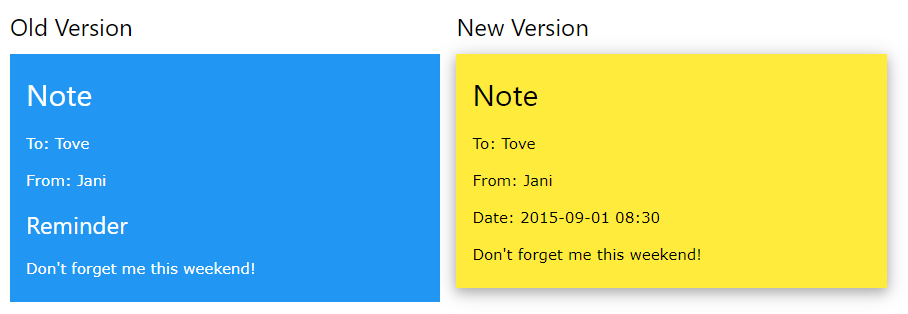
* The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
* HTML works with predefined tags like <p>, <h1>, <table>, etc.
* With XML, the author must define both the tags and the document structure.

**XML is Extensible**

* Most XML applications will work as expected even if new data is added (or removed).
* Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).
* Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed, older version of the application can still work:

<note>  
  <date>2015-09-01</date>  
  <hour>08:30</hour>  
  <to>Tove</to>  
  <from>Jani</from>  
  <body>Don't forget me this weekend!</body>  
</note>



**XML Simplifies Things**

* It simplifies data sharing
* It simplifies data transport
* It simplifies platform changes
* It simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

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

**XML is a W3C Recommendation**

XML became a W3C Recommendation as early as in February 1998.

**XML, or Extensible Markup Language**, stores data in a form where it can easily be retrieved and shared – even by incompatible applications. Here is a common scenario: Data is stored as XML, retrieved by Java, and displayed in HTML. The underlying code remains the same (meaning that a programmer doesn’t have to sit there all day making changes), and the screen doesn’t refresh constantly, annoying the end user. But when there’s a need for new data, it’s there in its current form.

XML is commonly used for documents that are in multiple languages or where there are multiple variants that serve the needs of different users. XML can, for example, be used to differentiate learning content for visually impaired learners. If you’re using it for this purpose, you don’t have to create a separate document for each use. You simply create rules that will determine how the data is handled by different applications.

**"While XML doesn’t actually do anything, it may be as fundamental to the web as HTML."**

Later Ajax or some other scripting language will “read” the XML and determine how to display data for the audience at hand. One other potential advantage to using XML is that some processing can be moved client-side as opposed to server-side.

* XML is more customizable than HTML.
* In HTML, you use tags that are pre-determined and have been defined in the language standard. With XML, you get to define your own tags.
* XML is designed in a way that allows individual industries to use their own vocabulary and create their own document structures.

In a sense, you’re creating your own language. It’s relatively easy for someone later to look at your code and make sense of it. (If you don’t specify how it’s supposed to be read, though, you’ll find that it’s your code/ mark-up displaying there in the browser.)

XML is used for data and content management in many industries. Industries have created their own standards for XML documents. Some examples are SCORM (used in e-learning) and HL7 (used in healthcare).

A number of new languages have been based off XML. They are designed to meet the very specific needs of particular industries. XML can be an efficient way of storing video and other multimedia. This led to the development of SMIL, a language that is based in XML and used for streaming. XHTML, as the name implies, has features of XML and HTML.

**History of XML**

Some languages trace their roots to individuals trying to solve small scale or localized problems. XML was put together by a team, working to develop a product with wide-scale application. The problem they were tackling? Making the internet more usable.

You don’t usually use XML in isolation – you use it in concert with other computer languages.

The team didn’t work from scratch. SGML provided most of the foundation. Development commenced in 1996, and a working copy was released within five months. XML 1.0 was approved by the W3 in 1998. Version 1.1 followed in 2004.

**Learning XML**

You don’t usually use XML in isolation – you use it in concert with other computer languages. You generally begin study after you’ve got a foundation in basic computer languages like HTML and Java or CSS. Some XML tutorials, like the one from W3Schools, even specify what languages you will need to know as a foundation.

**XML Basic**

<?xml version = "1.0"?>

<contact-info>

<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

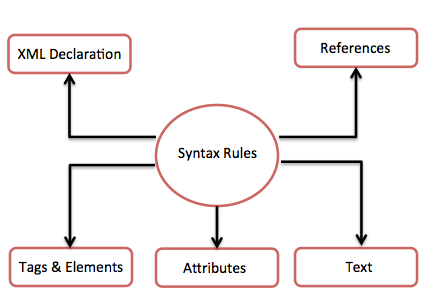
</contact-info>

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

Markup, like <contact-info>

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 follows –

<?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**

1. The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
2. If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
3. The XML declaration strictly needs be the first statement in the XML document.
4. 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. The names of XML-elements are enclosed in 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.

The Following example shows incorrect nested tags −

<?xml version = "1.0"?>

<contact-info>

<company>TutorialsPoint

<contact-info>

</company>

The 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>**

## 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".

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

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 shown 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

**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 this chapter − [XML Declaration](https://www.tutorialspoint.com/xml/xml_declaration.htm)

## 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.

[https://www.tutorialspoint.com//xhtml/what\_is\_xhtml.htm](https://www.tutorialspoint.com/xhtml/what_is_xhtml.htm)

What is XHTML?

XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage. It is the next step in the evolution of the internet. The XHTML 1.0 is the first document type in the XHTML family.

XHTML is almost identical to HTML 4.01 with only few differences. This is a cleaner and stricter version of HTML 4.01. If you already know HTML, then you need to give little attention to learn this latest version of HTML.

XHTML was developed by World Wide Web Consortium (W3C) to help web developers make the transition from HTML to XML. By migrating to XHTML today, web developers can enter the XML world with all of its benefits, while still remaining confident in the backward and future compatibility of the content.

**Why to Use XHTML?**

Developers who migrate their content to XHTML 1.0 get the following benefits −

1. XHTML documents are XML conforming as they are readily viewed, edited, and validated with standard XML tools.
2. XHTML documents can be written to operate better than they did before in existing browsers as well as in new browsers.
3. XHTML documents can utilize applications such as scripts and applets that rely upon either the HTML Document Object Model or the XML Document Object Model.
4. XHTML gives you a more consistent, well-structured format so that your webpages can be easily parsed and processed by present and future web browsers.
5. You can easily maintain, edit, convert and format your document in the long run.
6. Since XHTML is an official standard of the W3C, your website becomes more compatible with many browsers and it is rendered more accurately.
7. XHTML combines strength of HTML and XML. Also, XHTML pages can be rendered by all XML enabled browsers.
8. XHTML defines quality standard for your webpages and if you follow that, then your web pages are counted as quality web pages. The W3C certifies those pages with their quality stamp.

Web developers and web browser designers are constantly discovering new ways to express their ideas through new markup languages. In XML, it is relatively easy to introduce new elements or additional element attributes. The XHTML family is designed to accommodate these extensions through XHTML modules and techniques for developing new XHTML-conforming modules. These modules permit the combination of existing and new features at the time of developing content and designing new user agents.

**Basic Understanding**

Before we proceed further, let us have a quick view on what are HTML, XML, and SGML.

**What is SGML?**

This is **S**tandard **G**eneralized **M**arkup **L**anguage (SGML) application conforming to International Standard ISO 8879. HTML is widely regarded as the standard publishing language of the World Wide Web. This is a language for describing markup languages, particularly those used in electronic document exchange, document management, and document publishing. HTML is an example of a language defined in SGML.

XHTML syntax is very similar to HTML syntax and almost all the valid HTML elements are valid in XHTML as well. But when you write an XHTML document, you need to pay a bit extra attention to make your HTML document compliant to XHTML.

Here are the important points to remember while writing a new XHTML document or converting existing HTML document into XHTML document −

* Write a DOCTYPE declaration at the start of the XHTML document.
* Write all XHTML tags and attributes in lower case only.
* Close all XHTML tags properly.
* Nest all the tags properly.
* Quote all the attribute values.
* Forbid Attribute minimization.
* Replace the **name** attribute with the **id** attribute.
* Deprecate the **language** attribute of the script tag.

Here is the detail explanation of the above XHTML rules −

**DOCTYPE Declaration**

All XHTML documents must have a DOCTYPE declaration at the start. There are three types of DOCTYPE declarations, which are discussed in detail in XHTML Doctypes chapter. Here is an example of using DOCTYPE −

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

**Case Sensitivity**

XHTML is case sensitive markup language. All the XHTML tags and attributes need to be written in lower case only.

<!-- This is invalid in XHTML -->

<A Href="/xhtml/xhtml\_tutorial.html">XHTML Tutorial</A>

<!-- Correct XHTML way of writing this is as follows -->

<a href="/xhtml/xhtml\_tutorial.html">XHTML Tutorial</a>

In the example, **Href** and anchor tag **A** are not in lower case, so it is incorrect.

**Closing the Tags**

Each and every XHTML tag should have an equivalent closing tag, even empty elements should also have closing tags. Here is an example showing valid and invalid ways of using tags −

<!-- This is invalid in XHTML -->

<p>This paragraph is not written according to XHTML syntax.

<!-- This is also invalid in XHTML -->

<img src="/images/xhtml.gif" >

The following syntax shows the correct way of writing above tags in XHTML. Difference is that, here we have closed both the tags properly.

<!-- This is valid in XHTML -->

<p>This paragraph is not written according to XHTML syntax.</p>

<!-- This is also valid now -->

<img src="/images/xhtml.gif" />

**Attribute Quotes**

All the values of XHTML attributes must be quoted. Otherwise, your XHTML document is assumed as an invalid document. Here is the example showing syntax −

<!-- This is invalid in XHTML -->

<img src="/images/xhtml.gif" width=250 height=50 />

<!-- Correct XHTML way of writing this is as follows -->

<img src="/images/xhtml.gif" width="250" height="50" />

**Attribute Minimization**

XHTML does not allow attribute minimization. It means you need to explicitly state the attribute and its value. The following example shows the difference −

<!-- This is invalid in XHTML -->

<option selected>

<!-- Correct XHTML way of writing this is as follows -->

<option selected="selected">

Here is a list of the minimized attributes in HTML and the way you need to write them in XHTML −

|  |  |
| --- | --- |
| **HTML Style** | **XHTML Style** |
| compact | compact="compact" |
| checked | checked="checked" |
| declare | declare="declare" |
| readonly | readonly="readonly" |
| disabled | disabled="disabled" |
| selected | selected="selected" |
| defer | defer="defer" |
| ismap | ismap="ismap" |
| nohref | nohref="nohref" |
| noshade | noshade="noshade" |
| nowrap | nowrap="nowrap" |
| multiple | multiple="multiple" |
| noresize | noresize="noresize" |

**The *id* Attribute**

The id attribute replaces the name attribute. Instead of using name = "name", XHTML prefers to use id = "id". The following example shows how −

<!-- This is invalid in XHTML -->

<img src="/images/xhtml.gif" name="xhtml\_logo" />

<!-- Correct XHTML way of writing this is as follows -->

<img src="/images/xhtml.gif" id="xhtml\_logo" />

**The *language* Attribute**

The language attribute of the script tag is deprecated. The following example shows this difference −

<!-- This is invalid in XHTML -->

<script language="JavaScript" type="text/JavaScript">

document.write("Hello XHTML!");

</script>

<!-- Correct XHTML way of writing this is as follows -->

<script type="text/JavaScript">

document.write("Hello XHTML!");

</script>

**Nested Tags**

You must nest all the XHTML tags properly. Otherwise your document is assumed as an incorrect XHTML document. The following example shows the syntax −

<!-- This is invalid in XHTML -->

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

<!-- Correct XHTML way of writing this is as follows -->

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

**Element Prohibitions**

The following elements are not allowed to have any other element inside them. This prohibition applies to all depths of nesting. Means, it includes all the descending elements.

|  |  |
| --- | --- |
| **Element** | **Prohibition** |
| <a> | Must not contain other <a> elements. |
| <pre> | Must not contain the <img>, <object>, <big>, <small>, <sub>, or <sup> elements. |
| <button> | Must not contain the <input>, <select>, <textarea>, <label>, <button>, <form>, <fieldset>, <iframe> or <isindex> elements. |
| <label> | Must not contain other <label> elements. |
| <form> | Must not contain other <form> elements. |

**A Minimal XHTML Document**

The following example shows you a minimum content of an XHTML 1.0 document −

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/TR/xhtml1" xml:lang="en" lang="en">

<head>

<title>Every document must have a title</title>

</head>

<body>

...your content goes here...

</body>

</html>

**XHTML VS HTML**

Due to the fact that XHTML is an XML application, certain practices that were perfectly legal in SGML-based HTML 4 must be changed. You already have seen XHTML syntax in previous chapter, so differences between XHTML and HTML are very obvious. Following is the comparison between XHTML and HTML.

**XHTML Documents Must be Well-Formed**

Well-formedness is a new concept introduced by XML. Essentially, this means all the elements must have closing tags and you must nest them properly.

**CORRECT: Nested Elements**

<p>Here is an emphasized <em>paragraph</em>.</p>

**INCORRECT: Overlapping Elements**

<p>Here is an emphasized <em>paragraph.</p></em>

**Elements and Attributes Must be in Lower Case**

XHTML documents must use lower case for all HTML elements and attribute names. This difference is necessary because XHTML document is assumed to be an XML document and XML is case-sensitive. For example, <li> and <LI> are different tags.

**End Tags are Required for all Elements**

In HTML, certain elements are permitted to omit the end tag. But XML does not allow end tags to be omitted.

**CORRECT: Terminated Elements**

<p>Here is a paragraph.</p><p>here is another paragraph.</p>

<br><hr/>

**INCORRECT: Unterminated Elements**

<p>Here is a paragraph.<p>here is another paragraph.

<br><hr>

**Attribute Values Must Always be Quoted**

All attribute values including numeric values, must be quoted.

**CORRECT: Quoted Attribute Values**

<td rowspan="3">

**INCORRECT: Unquoted Attribute Values**

<td rowspan=3>

**Attribute Minimization**

XML does not support attribute minimization. Attribute-value pairs must be written in full. Attribute names such as compact and checked cannot occur in elements without their value being specified.

**CORRECT: Non Minimized Attributes**

<dl compact="compact">

**INCORRECT: Minimized Attributes**

<dl compact>

**Whitespace Handling in Attribute Values**

1. When a browser processes attributes, it does the following −
2. Strips leading and trailing whitespace.
3. Maps sequences of one or more white space characters (including line breaks) to a single inter-word space.

**Script and Style Elements**

In XHTML, the script and style elements should not have “<” and “&” characters directly, if they exist; then they are treated as the start of markup. The entities such as “<” and “&” are recognized as entity references by the XML processor for displaying “<” and “&” characters respectively.

Wrapping the content of the script or style element within a CDATA marked section avoids the expansion of these entities.

<script type="text/JavaScript">

<![CDATA[

... unescaped VB or Java Script here... ...

]]>

</script>

An alternative is to use external script and style documents.

**The Elements with *id* and *name* Attributes**

XHTML recommends the replacement of *name* attribute with *id* attribute. Note that in XHTML 1.0, the *name* attribute of these elements is formally deprecated, and it will be removed in a subsequent versions of XHTML.

**Attributes with Pre-defined Value Sets**

HTML and XHTML both have some attributes that have pre-defined and limited sets of values. For example, ***type*** attribute of the ***input*** element. In HTML and XML, these are called **enumerated attributes**. Under HTML 4, the interpretation of these values was case-insensitive, so a value of **TEXT** was equivalent to a value of **text**.

Under XHTML, the interpretation of these values is case-sensitive so all of these values are defined in lower-case.

**Entity References as Hex Values**

HTML and XML both permit references to characters by using hexadecimal value. In HTML these references could be made using either **&#Xnn;** or **&#xnn;** and they are valid but in XHTML documents, you must use the lower-case version only such as **&#xnn;**.

**The <html> Element is a Must**

All XHTML elements must be nested within the <html> root element. All other elements can have sub elements which must be in pairs and correctly nested within their parent element. The basic document structure is −

<!DOCTYPE html....>

<html>

<head> ... </head>

<body> ... </body>

</html>

**DTD, XSTL**

**DTD** <https://www.w3schools.com/xml/xml_dtd_intro.asp>

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.

**What is a DTD?**

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

**Why Use a DTD?**

With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.

## Syntax

Basic syntax of a DTD is as follows −

<!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*.

**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 an 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 the XML declaration with the 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** 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**.

**XSTL**

Extensible style language (XSL) is a style sheet language that is used for transforming and presenting XML documents. XSL is formalized as a specification World Wide Web Consortium (W3C).

The XSL specification is composed of three parts:

* XML Based Language: This is used for transforming XML documents, which is known as XSL transformation (XSLT)
* XML Based Language (2): This specifies the visual format of XML document, which is known as XSLformatting objects (XSL-FO).
* Non-XML Based Language: This addresses specific parts of XML document, known as XML Path Language (XPath).

For example, if an XML page contains a table describing company employees, XML describes this data, while XSL is used by the Web browser to render the data. Data color, fonts and other attributes are stored within the code for XSL, rather than XML.

XSL is critical to organizing data in a presentable, understandable format that can be easily classified (i.e., color in cell headers). XSL contains timing data that can be shown or hidden by developers according to a predetermined schedule. XSL may also contain template-descriptive data that can be reused with more than one XML page.

XSL is often viewed as an extension of the Document Style Semantics and Specification Language (DSSSL) with CSS1 standards.