



# XML: a standard, extensible data format

Release: 2010-04-11

Table Of Contents

- [XML](#)
- [URIs and IRIs](#)
- [MIME Media Types](#)
- [More Features](#)
- [Use more than one XML structure in the same document with XML Namespaces](#)
- [XML 1.1](#)
- [Doctypes and DTDs](#)

XML: a standard, extensible data format - Legend Scrolls

## **XML**

A way to create flexible Markup Languages based on the experience of SGML and HTML produced XML (eXtensible Markup Language). The strict structure rules of XML include normal elements must have a start and end tag such as `<paragraph>This is element content</paragraph>` and empty elements are mainly defined as a start tag with a slash before the greater-than character as `<linebreak/>` (empty elements can be a normal element too as long as there is absolutely nothing for the element content).

Attributes are name=value pairs and the values must always be quoted with double quotes (preferred) or single quotes. Comments must only be used within its own special empty element: `<!-- An XML comment -->`. Also as most SGML documents like HTML are case-insensitive: don't care if the element or attribute names are uppercase, lowercase or mixed, XML does care about the casing: is case sensitive.

Each element cannot have more than one attribute with the same name and an entity character reference must be any of the three forms Named Entity, Hex Entity or Numbered Entity such as `&apos;`, `&#x0027;`, `&#39;`.

## **URIs and IRIs**

All the resources on the World Wide Web are identified by a *URI*: Uniform Resource Identifier. A URI can be one of many types. Three of them are: a *URL*, a *URN* and a Tag URI.

A *URL* is a Uniform Resource Locator and is the most familiar of URIs. URL provides a location as well as an identifier in space (a spatial identifier), as opposed to an identifier in time (a temporal identifier), for a resource. Such URL type URIs include `http://` addresses.

`http://www.w3.org`  
`http://example.com/folder/filename`

**Figure 1:** Examples of URL type URIs

URLs can have a Fragment part at the end represented as a hash or sharp character (`#`) followed by a unique identifier that usually points to an *ID* type attribute with the same value in markup languages like XML:

`http://example.com/folder/page.xml#thissection`

**Figure 2:** An URI with a Fragment Part

Plus URLs can have an optional query string denoted by a question mark and the variable=value pair separated by an ampersand (`&`):

`http://example.com/folder/page.php?name=me&range=2400&packed=true`  
`http://example.com/folder/page.php?`  
`name=me&range=2400&packed=true#thatsection`

XML: a standard, extensible data format - Legend Scrolls

### **Figure 3:** URI Query Strings

Although the ampersand would be `&` within XML, as:

```
http://example.com/folder/page.php?  
name=me&range=2400&packed=true  
http://example.com/folder/page.php?  
name=me&range=2400&packed=true#thatsection
```

### **Figure 4:** Ampersands in URIs in XML

The above examples are absolute URIs, but you can use relative URIs that are automatically converted to absolute by whatever environment like a web browser:

```
folder/filename  
../otherfolder/me.xml
```

### **Figure 5:** relative URIs

The web browser would take the relative URI and prefix the URI of the current document, without the document's filename, to create an absolute URI. For referring to things in another folder near the current one you use the `../` for 'up-one-level'. So the last example is referring to 'me.xml' in the 'otherfolder' folder that is beside the folder that you are in.

A *URN* is a Uniform Resource Name which provides a spatial identifier independent of it's physical or electronic location. These are restricted to some companies or organizations and include ISBNs for publications and also several organizations use URNs to access built in schemas, etc.

```
urn:isbn:0-00-000000-0  
urn:schemas-microsoft-com:office:office  
urn:oasis:names:tc:opendocument:xmlns:office:1.0
```

### **Figure 6:** Examples of URN type URIs

Tag URIs include `tag:example.com,2009:articles.entry01` or `tag:me@example.com,2009-02-10:myposts.entry-01`.

The 'tag' URI (or 'tags') provides an unique identifier not just in space but in time as well. Beginning with the word 'tag' followed by a colon followed by either a domain name that you have bought or your email address. Then a comma and a UTC date is used for the date when the domain name or email address was owned. Usually people choose the current year they are creating the `tag:`.

The date can be a four digit year or a four digit year dash two digit month or a full UTC Date such as `2009`, `2009-01` or `2009-01-01`. After the second colon (`:`) you can usually use slashes (`/`), colons and `/` or dots to separate any meaningful keywords.

*IRI* stands for Internationalized Resource Identifier and is basically a URI that supports non-Latin characters.

XML: a standard, extensible data format - Legend Scrolls

## ***MIME Media Types***

Most resources on the Internet are also categorized to a MIME Media Type such as [text/plain](#) for text files (and the default for uncategorised files), [audio/mpeg](#) for mp3 audio, [audio/m4a](#) for .m4a MPEG4 audio, [image/png](#) for Portable Network Graphics (PNG), [application/octet-stream](#) for binary files, [text/html](#) for HTML webpages, [text/css](#) for Cascade StyleSheets, [application/xml](#) for XML Documents, [application/xhtml+xml](#) for XHTML webpages, [video/mp4](#) for MPEG4 video, [application/ogg](#) for OGG audio and video and [image/svg+xml](#) for Scalable Vector Graphics (SVG).

This allows a flexible construct to create markup languages where the document author can define their own document vocabulary. This is the foundation of XML.

## ***More Features***

Other features that XML inherits from SGML are the special elements such as Processing Instructions (`<?appName attributeOrScripting ?>`) and Character Data Sections (`<![CDATA[ This is a character data section. ]]>`). Character Data Sections surround the element content to allow certain characters to be as themselves where otherwise would be interpreted as part of the markup and so XML would throw errors at you.

XML has native support for Unicode characters and all XML parsers (processors) have a built-in Document Well-Formed Validator in which it checks your XML document against the strict structure rules. Some characters to keep in mind are the quotes in attribute values and the start pointy bracket (`<`) and the ampersand (`&`) in both attribute values, element content and comments: attribute values are commonly quoted with the double quote (`"`) and so you need the named entity, `&quot;`; within the attribute value if you need the double quote in there. Otherwise the XML parser will think you are ending the attribute value before you intended and the rest of the value will cause a Document Well-formed error. But single quotes or apostrophes (`'`) can be used as themselves when double quotes surround the value. Similar for the less used values surrounded by single quotes: double quotes can be used as themselves but single quotes or apostrophes must use the named entity, `&apos;`, first introduced by XML and not available in HTML until HTML 5 (`&#x0027;` hex entity or `&#39;` numbered entity could be used by both HTML and XML as all hex and numbered entities are supported by both).

Start pointy brackets (`<`) are used as part of start and end tags for an element and part of empty elements and other special element markup so in order to use it as itself you need to use the `&lt;` named entity within attribute values, element content and comments. Ampersands are used as part of Entities such

XML: a standard, extensible data format - Legend Scrolls

as `&quot;`; so to use it as itself you need the `&amp;` named entity.

As stated earlier when element content is surrounded by a CDATA section these character rules are relaxed so you can use `<` and `&` as they are as long as you don't have `]]>` characters together within the Section as this will end the CDATA section. The named entity `&gt;` can be used instead of the literal end pointy bracket (`>`) but it does not pose any problems towards Document Well-Formedness.

The first element of an XML document is also referred to as the Document Element. An XML Document usually also has an XML Declaration but isn't required. The XML Declaration is a Processing Instruction where the appName or target application is `'xml'` and has a version, encoding and possibly a standalone attributes:

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

**Figure 7:** XML Declaration

The encoding dictates which character set the characters are going to be from. By default XML Documents use UTF-8 (8-bit Universal Character Set Transfer Format) which takes advantage of the Unicode support. This attribute is optional.

`standalone` attribute dictates if the document has any associated documents (such as StyleSheets can be used to present the document or a DTD or XML Schema is used to validate the document structure at an XML Language level beyond the Well-Formed Rules). This optional attribute has the values `'yes'` or `'no'` where `'no'` is the default.

XML also provides a few predefined attributes that you can use on any of your XML Documents:

- `xml:lang` provide a global language selector such as `xml:lang="en-GB"` or `xml:lang="en"`, etc.
- `xml:space` has the values of `'default'` (which is the default value) which removes leading and trailing spaces, tabs, newlines and other 'whitespace' characters from attribute values and element content and reduces two or more whitespace characters between words down to a single whitespace. `'preserve'` is the other value used to keep the spacing. This whitespace handling is for the XML Parser only and not for presentational purposes (CSS has the presentational version covered with the `white-space` property).
- `xml:base` takes a [URI](#) to help construct absolute URIs for relative links and other relative URIs - like the HTML `<base href="...">` element except as it is an attribute it can be used on any element and so any part of the document and any number of `xml:base` attributes can be used.
- `xml:id` is a global unique identifier type attribute.

These attributes can be used on any element or restricted by some form of

XML: a standard, extensible data format - Legend Scrolls

document validation.

A note on unique identifier (ID) type attributes such as `xml:id` or `id` from other XML languages: an element can have only one ID type attribute and the value must be unique through the whole document. This is the same for HTML's `id` attribute. ID type attribute values can only start with a letter, underscore ( `_` ) or a colon ( `:` ) and then as many letters, underscores, colons, numbers, dashes ( `-` ) and other characters.

XML has a couple of MIME Media Types that all XML based languages may use: `text/xml` and `application/xml` which is used in preference over `text/xml` these days. Most XML based languages have their own MIME Media Type following the form of `application/xmllanguage+xml`.

### ***Use more than one XML structure in the same document with XML Namespaces***

More than one XML based language may have elements or attributes of the same name. This is fine if you are isolating each XML structure into its own document. But one reason why people and companies like to use XML is to build documents with more than one XML based language within it as each XML language may describe specific things such as 2-d images, math, hyperlinks, etc. In order to uniquely identify which XML based language an element or attribute is from, most XML languages these days have their own XML Namespace.

An XML Namespace is generally a [URI](#) which is associated with an XML Prefix. The Namespaces are declared by an `xmlns:yourprefix` attribute usually within the document element but can be in any element. Then the prefix `yourprefix:` is prefixed to the element or attribute. Such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<ex:doc xmlns:ex="http://example.com/namespace" xmlns:c="http://www.c.org/charl"
xml:lang="en">
  <ex:chapter number="1" c:signed="me">
    <ex:page number="1">
      <c:emerald name="sam"/>
      <ex:title>A title</ex:title>
      <ex:text c:style="fancyEnglish">blah blah doc
        blah some text blah blah watch out for orbs blah blah
      </ex:text>
    </ex:page>
  </ex:chapter>
</ex:doc>
```

**Figure 8:** Prefixed XML Document

As you can see the main elements are bound to the `http://example.com/namespace` namespace by the `ex` prefix. The attributes that are not prefixed in an element are automatically bound to the same

XML: a standard, extensible data format - Legend Scrolls

namespace as the element. Plus the extra elements and attributes (that are on a different namespaced element) are bound to the <http://www.c.org/charl> namespace by the `c` prefix. Each XML based language has a fixed Namespace URI and most have a typical prefix but no prefix is hard wired so you can use your own.

The original XML Namespace which uses `xml` as the prefix is automatically declared by all namespace-aware XML parsers and the use of the original XML Namespace is restricted to only the formally published XML Attributes currently: `xml:lang`, `xml:space`, `xml:base` and `xml:id`. You can have a Default Namespace which has no prefix. Such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns="http://example.com/namespace" xmlns:c="http://www.c.org/charl"
xml:lang="en">
  <chapter number="1" c:signed="me">
    <page number="1">
      <c:emerald name="sam"/>
      <title>A title</title>
      <text c:style="fancyEnglish">blah blah doc
        blah some text blah blah watch out for orbs blah blah
      </text>
    </page>
  </chapter>
</doc>
```

**Figure 9:** Default Namespaced XML Document

## **XML 1.1**

The original XML specification does have a few limitations that have now been removed in XML 1.1. Rather than only supporting most of Unicode 2 in element and attribute names and processing instruction targets, version 1.1 of XML supports any Unicode based character that is not specifically forbidden allowing support for Unicode 3 to 5 and future versions. Also XML 1.1 supports [IRIs](#).

Plus support for the NEL new line character (`#x85`), mostly used on IBM's AIX Operating System, as well as the Unicode Line Separator character (`#x2028`).

## **Doctypes and DTDs**

To validate XML Documents, XML 1.0 and 1.1 provide a Document Type Definition (DTD) sub-syntax or sub-language. They are usually in separate `.dtd` files and referred to by the Doctype in the XML Document before the Document Element. DTDs are only useful with actual validation services – web browsers tend not to support DTDs or only support a tiny amount in particular circumstances as browsers are there to display documents, not to validate them. As not all UserAgents support DTDs then anything that depends on

XML: a standard, extensible data format - Legend Scrolls

DTDs, such as support for your own named character entity references can not be guaranteed.

A Doctype starts with a less-than character, an exclamation mark, the word DOCTYPE in uppercase, a space, the name of the Document Element, a space. Then either the word SYSTEM in uppercase, a space then a URL to the .dtd file within quotes.

Or the word PUBLIC in uppercase, a space, a Public ID String in quotes, a space and then the DTD URL in quotes without the word SYSTEM. The Public ID String has the form of a dash, two slashes, the name of the company or individual that invented the particular XML language, two slashes, the XML language name and version, two slashes and then a two character form of a written Human language in uppercase that the DTD would be written in; usually this would be the word EN.

Or, last one, an open square bracket, then the contents of the DTD followed by a closed square bracket. This is the Doctype Subset and allows the DTD information directly within the XML Document itself rather than referring to a separate file.

After any of those three you close the Doctype with a greater-than character.

```
<!DOCTYPE mydoc SYSTEM "http://www.example.com/dtds/mydoc.dtd">
```

**Figure 10:** Doctype with systemid.

or

```
<!DOCTYPE mydoc PUBLIC "-//DocsCorp//My Documents 1.0//EN"
"http://www.example.com/dtds/mydoc.dtd">
```

**Figure 11:** Doctype with publicid and systemid

or

```
<!DOCTYPE mydoc [
]>
```

**Figure 12:** Doctype Subset.

To define your own character entity references you use the `<!ENTITY >` DTD special element. You specify the name of the entity (without ampersand and semicolon) and then the value in quotes. The value could be a Hex Entity or a Numbered Entity or a string of normal characters.

```
<!ENTITY frac14 "&#188;">
```

**Figure 13:** DTD code to create `&frac14;` the Named Entity for ¼ from (X)HTML

```
<!ENTITY docStandardLine "This document is built with XML tools. This line will
appear in many places">
```

**Figure 14:** DTD code to create `&docStandardLine;`

Similar to character entity references can be used as variables within the DTD

## XML: a standard, extensible data format - Legend Scrolls

information such as to store attribute validation code in a DTD Entity and use it for common attributes that are used on many elements.

```
<!ENTITY % commonAttr "xml:lang NMTOKEN #IMPLIED">
```

**Figure 15:** DTD code to create %commonAttr; used only in the DTD information.

Elements are validated by the `<!ELEMENT >` DTD special element and stating the element name and the content model. Element content models will either be the word `ANY` for extremely flexible elements or the word `EMPTY` for Empty Elements. Otherwise within brackets you have a comma separated list of one or more element names of elements that are allowed within the current element that you are validating. Within the brackets you can also, or instead of, have the word `#PCDATA` to allow general text and character entity references. After the element name in the value (before the comma) you can use a question mark to indicate the child element is optional or a plus sign to indicate one or more of the child element is allowed or an asterisk (`*`) to indicate zero or more of the child element is allowed. You can also add the asterisk or plus sign or question mark after the brackets to indicate those options for the whole element content.

```
<!ELEMENT text (#PCDATA)>
<!ELEMENT c:emerald EMPTY>
<!ELEMENT page (c:emerald?, title, text+)>
```

**Figure 16:** Element `text` can have text. Element `c:emerald` is an Empty Element. Element `page` may have an optional `c:emerald` element, a single `title` element and one or more `text` elements.

Attributes use the `<!ATTLIST >` DTD special element to be validated specifying the element name that the attributes are on and a list of attribute info. The attribute info contains the attribute name, the value type and if it is optional, required or required with a fixed value.

```
<!ATTLIST chapter
  number      CDATA      #REQUIRED
  c:signed    CDATA      #IMPLIED
>
<!ATTLIST doc
  xmlns          CDATA      #FIXED
'http://example.com/namespace'
  xmlns:c       CDATA      #FIXED 'http://c.org/charl'
  %commonAttr;
>
```

**Figure 17:** Attribute `number` may have text and is required and attribute `c:signed` may also have text but is optional. Namespace declarations take text and have fixed values and the language declaration takes a name token (NMTOKEN).

Some of the value types for attributes are:

XML: a standard, extensible data format - Legend Scrolls

- **CDATA**: is generally text and character entity references;
- **NMTOKEN**: a single keyword;
- **NMTOKENS**: a space separated list of keywords;
- **ID**: a unique identifier value;
- **IDREF**: a value that matches a single ID value;
- **IDREFS**: a space separated list of values, each match an ID value;

There are no types for numbers or dates, etc. You generally would use **CDATA** for those. The limited attribute value types is one of the reasons why XML Schema and RELAX NG Schema were created, specifically XML Schema Datatypes provide a rich set of datatypes. But these schemas are beyond this article.

XML: a standard, extensible data format - Legend Scrolls

This article and others are available online and in other document formats at:

<http://www.legendscrolls.co.uk/webstandards/>

Author: Peter Davison from [Legend Scrolls](#).

Copyright ©2005-2010 Legend Scrolls and Peter Davison.  
All rights reserved.