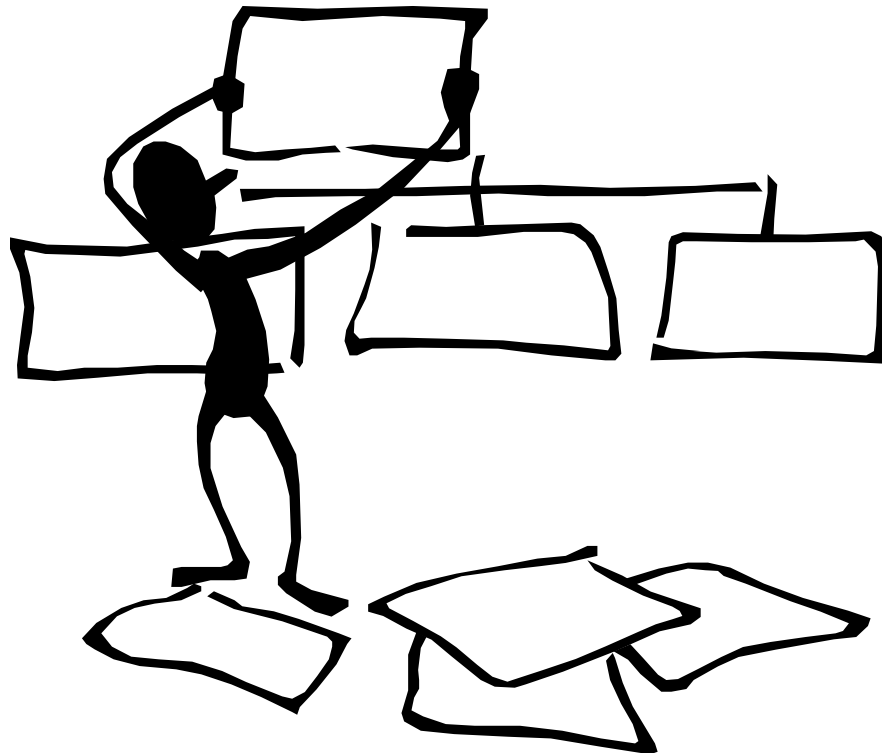


XML 1.0 and Namespaces



XML Structure and Syntax

From:
Skonnard, Gudgin,
"XML Quick Reference"
ISBN 0-201-74095-8

XML 1.0 and Namespaces

- ◆ XML 1.0 and Namespaces in XML provide a tag-based syntax for structuring data and applying markups to documents.
- ◆ Documents that conform to XML 1.0 and Namespaces in XML specifications may be made up of a variety of syntactic constructs such as elements, namespace declarations, attributes, processing instructions, comments, and text.

Elements

```
<tagname></tagname>  
<tagname/>  
<tagname>children</tagname>
```

- ◆ Elements typically make up the majority of the content of an XML document.
- ◆ Every XML document has exactly one top-level element, known as the *document element*.
- ◆ Elements have a name and may also have children. These children may themselves be elements or may be processing instructions, comments, CDATA sections, or characters. The children of an element are ordered.

- ◆ Elements may also be annotated with attributes. The attributes of an element are unordered.
- ◆ An element may also have namespace declarations associated with it. The namespace declarations of an element are unordered.
- ◆ Elements are serialized as a pair of tags: an open tag and a close tag. The syntax for an open tag is the less-than character (<) immediately followed by the name of the element, also known as the *tagname*, followed by the greater-than character (>).
- ◆ The syntax for a close tag is the character sequence </ immediately followed by the tagname, followed by the greater-than character (>).

- ◆ The children of an element are serialized between the open and close tags of their parent.
- ◆ In cases when an element has no children, the element is said to be *empty*. A shorthand syntax may be used for empty elements consisting of the less-than character (<) immediately followed by the tagname, followed by the character sequence />.
- ◆ XML does not define any element names; rather, it allows the designer of an XML document to choose what names will be used. Element names in XML are case sensitive and must begin with a letter or an underscore (_).
- ◆ The initial character may be followed by any number of letters, digits, periods (.), hyphens (-), under-scores, or colons (:).

- ◆ Because colons are used as part of the syntax for namespaces in XML, they should not be used except as described by that specification.
- ◆ Element names that begin with the character sequence xml, or any recapitalization thereof, are reserved by the XML specification for future use.

Elements Examples

- ◆ *An element with children*
`<Person>`
 `<name>Martin</name>`
 `<age>33</age>`
`</Person>`
- ◆ *An empty element*
`<Paid></Paid>`
- ◆ *Empty element shorthand*
`<Paid/>`

Namespaces

```
<prefix:localname xmlns:prefix='namespace URI'>  
  </prefix:localname>  
<prefix:localname xmlns:prefix='namespace URI' />  
<prefix:localname xmlns:prefix='namespace URI'>  
  children  
</prefix:localname>
```

- ◆ Because XML allows designers to choose their own tag names, it is possible that two or more designers may choose the same tag names for some or all of their elements.
- ◆ XML namespaces provide a way to distinguish deterministically between XML elements that have the same local name but are, in fact, from different vocabularies.

- ◆ This is done by associating an element with a namespace. A namespace acts as a scope for all elements associated with it.
- ◆ Namespaces themselves also have names. A namespace name is a uniform resource identifier (URI).
- ◆ The namespace name and the local name of the element together form a globally unique name known as a *qualified name*.

- ◆ Namespace declarations appear inside an element start tag and are used to map a namespace name to another, typically shorter, string known as a *namespace prefix*.
- ◆ The syntax for a namespace declaration is `xmlns:prefix='URI'`. It is also possible to map a namespace name to no prefix using a default namespace declaration. The syntax for a default namespace declaration is `xmlns='URI'`.
- ◆ The URI may appear in single quotes (‘) or double quotes (“).
- ◆ Only one default namespace declaration may appear on an element.
- ◆ Any number of nondefault namespace declarations may appear on an element, provided they all have different prefix parts.
- ◆ It is legal, although not particularly useful, to map the same URI to more than one prefix.

- ◆ All namespace declarations have a scope—that is, a set of elements to which they may apply.
- ◆ A namespace declaration is in scope for the element on which it is declared and all of that element’s descendants.
- ◆ The in-scope mapping of a given prefix to a namespace name can be overridden by providing a new mapping for that prefix on a descendant element.
- ◆ The in-scope default namespace can be overridden by providing a new default namespace declaration on a descendant element.

- ◆ The names of all elements in a document that conforms to the Namespaces in the XML specification are QNames.
- ◆ Syntactically, all QNames have a local name and an optional prefix. Both the local name and the prefix are NCNames.
- ◆ An NCName is a name without a colon in it.
- ◆ The syntax for an element with a prefix is the prefix, followed by a colon, followed by the local name.
- ◆ Elements not in any namespace are known as *unqualified elements*.
- ◆ The namespace name of unqualified elements is the empty string "".

- ◆ *Qualified and unqualified elements*

```
<pre:Person xmlns:pre='urn:example-org:People'>
  <name>Martin</name>
  <age>33</age>
</pre:Person>
```

- ◆ An element with a local name of Person and a prefix of pre that is mapped to the namespace name urn:example-org:People .
- ◆ The element has children with local names of name and age . Both of these child elements are unqualified; that is, they are not in any namespace.

- ◆ *Qualified and unqualified elements using a default namespace declaration*

```
<Person xmlns='urn:example-org:People'>  
  <name xmlns="">Martin</name>  
  <age xmlns="">33</age>  
</Person>
```

- ◆ An element with a local name of Person and no prefix. The element is in the namespace urn:example-org:People by virtue of an in-scope default namespace declaration for that URI.
- ◆ The element has children with local names of name and age . Both of these child elements are unqualified; that is, they are not in any namespace.
- ◆ This example is equivalent to the previous example.

- ◆ *Qualified elements*

```
<pre:Person xmlns:pre='urn:example-org:People'>  
  <pre:name>Martin</pre:name>  
  <pre:age>33</pre:age>  
</pre:Person>
```

- ◆ An element with a local name of Person and a prefix of pre that is mapped to the namespace URI urn:example-org:People .
- ◆ The element has children with local names of name and age . Both of these child elements also have a prefix of pre and are in the urn:example-org:People namespace.

- ◆ *Qualified elements using a default namespace declaration*

```
<Person xmlns='urn:example-org:People'>
  <name>Martin</name>
  <age>33</age>
</Person>
```

- ◆ An element with a local name of Person and no prefix. The element is in the namespace urn:example-org:People by virtue of an in-scope default namespace declaration for that URI.
- ◆ The element has children with local names of name and age . Both of these child elements are also in the urn:example-org:People namespace.
- ◆ This example is equivalent to the previous example.

Attributes

name='value'

name="value"

- ◆ Elements can be annotated with attributes. Attributes can be used to encode actual data or to provide metadata about an element—that is, provide extra information about the content of the element on which they appear.
- ◆ The attributes for a given element are serialized inside the start tag for that element.
- ◆ Attributes appear as name/value pairs separated by an equal sign (=).
- ◆ Attribute names have the same construction rules as element names. Attribute values are textual in nature and must appear either in single quotes or double quotes.

- ◆ An element may have any number of attributes, but they must all have different names.
- ◆ **Examples**
- ◆ *Data attributes*

```
<Person name='Martin' age='33'/>
```

- ◆ A person represented using attributes rather than child elements
- ◆ *Metadata attributes*

```
<age base='16' units='years'>20</age>  
<age base="10" units="years">32</age>
```

- ◆ Some elements with metadata attributes

Attributes and Namespaces

prefix:localname='value'
prefix:localname="value"

- ◆ Attribute names are QNames. The namespace of an attribute with a given prefix is the namespace specified by the in-scope namespace declaration for that prefix.
- ◆ It is an error if no such namespace declaration is in scope.
- ◆ Unprefixed attributes are not in any namespace even if a default namespace declaration is in scope.

Attributes and Namespaces

(cont)

- ◆ **Examples**

- ◆ *Qualified attributes*

```
<Person xmlns='urn:example-org:People'  
        xmlns:b='urn:example-org:People:base'  
        xmlns:u='urn:example-org:units'>  
  <name>Martin</name>  
  <age b:base='10'u:units='years'>33</age>  
</Person>
```

- ◆ An attribute with a local name of base in the namespace urn:example-org:People:base and an attribute with a local name of units in the namespace urn:example-org:units

- ◆ *Unqualified attributes*

```
<Person xmlns='urn:example-org:People'>  
  <name>Martin</name>  
  <age base='10'units='years'>33</age>  
</Person>
```

- ◆ Attributes that are in no namespace, even though a default namespace declaration is in scope

Processing Instructions

<?target data?>

- ◆ Processing instructions are used to provide information to the application processing an XML document. Such information may include instructions on how to process the document, how to display the document, and so forth.
- ◆ Processing instructions can appear as children of elements. They can also appear as top-level constructs (children of the document) either before or after the document element.

Processing Instructions

(cont)

- ◆ Processing instructions are composed of two parts: the target or name of the processing instruction and the data or information. The syntax takes the form `<?target data?>`.
- ◆ The target follows the same construction rules as for element and attribute names.
- ◆ Apart from the termination character sequence (`?>`), all markup is ignored in processing instruction content.
- ◆ Processing instructions defined by organizations other than the World Wide Web Consortium (W3C) may not have targets that begin with the character sequence `xml` or any recapitalization thereof.

Processing Instructions

(cont)

- ◆ Namespace declarations do not apply to processing instructions. Thus, creating targets that are guaranteed to be unique is problematic.

- ◆ **Example**

- ◆ *Processing instructions*

<?display table-view?>

<?sort alpha-ascending?>

<?textinfo whitespace is allowed ?>

<?elementnames <fred>,<bert>,<harry>?>

- ◆ Various processing instructions

Comments

`<!--comment text -->`

- ◆ XML supports comments that are used to provide information to humans about the actual XML content. They are not used to encode actual data.
- ◆ Comments can appear as children of elements. They can also appear as top-level constructs (children of the document) either before or after the document element.
- ◆ Comments begin with the character sequence `<!--` and end with the character sequence `-->`.

- ◆ The text of the comment is serialized between the start and the end sequences.
- ◆ The character sequence -- may not appear inside a comment.
- ◆ Other markup characters such as less than, greater than, and ampersand (&), may appear inside comments but are not treated as markup. Thus, entity references that appear inside comments are not expanded.

- ◆ **Examples**

- ◆ *Legal comments*

<!--This is a comment about how to open (<![CDATA []and
close

([]>)CDATA sections -->

<!--I really like having elements called <fred>in my
markup languages -->

<!--Comments can contain all sorts of character literals
including &, <, >, ' and ". -->

<!--If entities are used inside comments (< for example)
they are not expanded.-->

- ◆ Some syntactically legal comments

◆ *Illegal comments*

<!--Comments cannot contain the --
character sequence -->

<!--Comments cannot end with a hyphen --->

<!--Comments cannot <!--be nested -->--->

◆ Some syntactically illegal comments

Whitespace

- ◆ Whitespace characters in XML are space, tab, carriage return, and line feed characters.
- ◆ XML requires that whitespace be used to separate attributes and namespace declarations from each other and from the element tagname.
- ◆ Whitespace is also required between the target and data portion of a processing instruction and between the text portion of a comment and the closing comment character sequence (-->) if that text ends with a hyphen (-).
- ◆ XML allows whitespace inside element content, attribute values, processing instruction data, and comment text.
- ◆ Whitespace is also allowed between an attribute name and the equal character and between the equal character and the attribute value. The same is true for namespace declarations.

- ◆ Whitespace is allowed between the tag-name of an open or close tag and the ending character sequence for that tag.
- ◆ Whitespace is not allowed between the opening less-than character and the element tagname or between the prefix, colon, and local name of an element or attribute.
- ◆ Nor is it allowed between the start processing instruction character sequence `<?` and the target.

Whitespace

(cont)

- ◆ **Examples**
- ◆ *Legal use of whitespace*

```
<pre:Vehicle xmlns:pre='urn:example-org:Transport' type='car' >
  <seats> 4 </seats>
  <colour> White </colour>
  <engine>
    <petrol />
    <capacity units='cc' >1598</capacity>
  </engine >
</pre:Vehicle >
```

- ◆ Whitespace used in various places in an XML document: between the tagname, namespace declaration, attribute, and closing greater-than character on the top-level element start tag, between each element, in the character content of the seats and colour elements, between the tagname and the />sequence of the petrol element, between the tagname and the closing greater-than character of the end tag for the engine element and the top-level element.

- ◆ *Illegal use of whitespace*

```
<pre :Vehicle xmlns: pre='urn:example-org:Transport'  
type='car'>  
    < seats>4</ seats>  
</pre:Vehicle>
```

- ◆ Whitespace used incorrectly in various places in an XML document: between pre and :Vehicle in the start tag of the top-level element, between xmlns:and pre of the namespace declaration of the top-level element, between the opening less-than character and seats in the start tag of the child element, and between </and seats in the end tag of the child element.

Prohibited Character Literals

<
&
>
'
"

- ◆ Certain characters cause problems when used as element content or inside attribute values.
- ◆ Specifically, the less-than character cannot appear either as a child of an element or inside an attribute value because it is interpreted as the start of an element.
- ◆ The same restrictions apply to the ampersand because it is used to indicate the start of an entity reference.
- ◆ If the less-than or ampersand characters need to be encoded as element children or inside an attribute value, then a character entity must be used.

Prohibited Character Literals

(cont)

- ◆ Entities begin with an ampersand and end with a semicolon (;). Between the two, the name of the entity appears.
- ◆ The entity for the less-than character is `<`; the entity for the ampersand is `&` .
- ◆ The apostrophe (') and quote characters (") may also need to be encoded as entities when used in attribute values. If the delimiter for the attribute value is the apostrophe, then the quote character is legal but the apostrophe character is not, because it would signal the end of the attribute value.
- ◆ If an apostrophe is needed, the character entity `'` must be used. Similarly, if a quote character is needed in an attribute value that is delimited by quotes, then the character entity `"` must be used.

Prohibited Character Literals

(cont)

- ◆ A fifth character reference is also provided for the greater-than character. Although strictly speaking such characters seldom need to be “escaped,” many people prefer to “escape” them for consistency with the less-than character.

- ◆ **Examples**

- ◆ *Built-in entity in element content*

```
<IceCream>  
  <name>Cherry Garcia</name>  
  <manufacturer>Ben & Jerry</manufacturer>  
</IceCream>
```

- ◆ Use of the built-in entity & inside element content.

Prohibited Character Literals

(cont)

- ◆ *Built-in entity in attribute content*

```
<sayhello word='&apos;Hi&apos;'/>
```

- ◆ Use of the built-in entity `'` inside attribute content.

CDATA Sections

`<![CDATA [text content possibly containing literal << or & characters]]>`

- ◆ CDATA sections can be used to “block escape” literal text when replacing prohibited characters with entity references is undesirable.
- ◆ CDATA sections can appear inside element content and allow `<` and `&` character literals to appear.
- ◆ A CDATA section begins with the character sequence `<![CDATA [` and ends with the character sequence `]]>`.
- ◆ Between the two character sequences, an XML processor ignores all markup characters such as `<`, `>`, and `&`. The only markup an XML processor recognizes inside a CDATA section is the closing character sequence `]]>`.

- ◆ The character sequence that ends a CDATA section `]]>` must not appear inside the element content. Instead, the closing greater-than character must be escaped using the appropriate entity `>`.
- ◆ CDATA sections cannot be nested.

XML Declaration

```
<?xml version='1.0' encoding='character encoding'  
standalone='yes|no'?>
```

- ◆ XML documents can contain an XML declaration that if present, must be the first construct in the document.
- ◆ An XML declaration is made up of as many as three name/value pairs, syntactically identical to attributes.
- ◆ The three attributes are a mandatory version attribute and optional encoding and standalone attributes.
- ◆ The order of these attributes within an XML declaration is fixed.
- ◆ The XML declaration begins with the character sequence <?xml and ends with the character sequence ?>.

XML Declaration

(cont)

- ◆ Note that although this syntax is identical to that for processing instructions, the XML declaration is not considered to be a processing instruction.
- ◆ All XML declarations have a version attribute with a value that must be 1.0 .
- ◆ The character encoding used for the document content can be specified through the encoding attribute.
- ◆ XML documents are inherently Unicode, even when stored in a non-Unicode character encoding.
- ◆ The XML recommendation defines several possible values for the encoding attribute.

XML Declaration

(cont)

- ◆ For example, UTF-8, UTF-16, ISO-10646-UCS-2, and ISO-10646-UCS-4 all refer to Unicode/ISO-10646 encodings, whereas ISO-8859-1 and ISO-8859-2 refer to 8-bit Latin character encodings.
- ◆ Encodings for other character sets including Chinese, Japanese, and Korean characters are also supported. It is recommended that encodings be referred to using the encoding names registered with the Internet Assigned Numbers Authority (IANA).
- ◆ All XML processors are required to be able to process documents encoded using UTF-8 or UTF-16, with or without an XML declaration. The encoding of UTF-8- and UTF-16-encoded documents is detected using the Unicode byte-order-mark.
- ◆ The XML declaration is mandatory if the encoding of the document is anything other than UTF-8 or UTF-16. In practice, this means that documents encoded using US-ASCII can also omit the XML declaration because US-ASCII overlaps entirely with UTF-8.

XML Declaration

(cont)

- ◆ Only one encoding can be used for an entire XML document. It is not possible to “redefine” the encoding part of the way through. If data in different encodings needs to be represented, then external entities should be used.
- ◆ If an XML document can be read with no reference to external sources, it is said to be a *stand-alone document*. Such documents can be annotated with a standalone attribute with a value of yes in the XML declaration.
- ◆ If an XML document requires external sources to be resolved to parse correctly and/or to construct the entire data tree (for example, a document with references to external general entities), then it is not a stand-alone document.
- ◆ Such documents may be marked `standalone='no'`, but because this is the default, such an annotation rarely appears in XML documents.

XML Declaration

(cont)

- ◆ **Example**

- ◆ *XML declarations*

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

```
<?xml version='1.0' encoding='US-ASCII' ?>
```

```
<?xml version='1.0' encoding='US-ASCII'  
    standalone='yes' ?>
```

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

```
<?xml version='1.0' encoding='UTF-16' ?>
```

```
<?xml version='1.0' encoding='ISO-10646-UCS-2' ?>
```

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
```

```
<?xml version='1.0' encoding='Shift-JIS' ?>
```

Character Reference

&#DecimalUnicodeValue;
&#xHexadecimalUnicodeValue;

- ◆ Many character encodings cannot natively represent the full range of ISO-10646 characters.
- ◆ When an XML document contains characters that cannot be represented natively in the chosen encoding, then these nonrepresentable characters must be written as character references.
- ◆ Character references begin with the character sequence &# followed by the ISO-10646 value of the character to be written in either decimal or hexadecimal form.
- ◆ If the character value is represented in hexadecimal form, then it must be preceded by an x .

Character Reference

(cont)

- ◆ Character references end with ;.
- ◆ Character references can only be used for attribute and element content.
- ◆ Non representable characters appearing as part of element or attribute names or as part of processing instructions or comments cannot be written using character references; rather, a more suitable encoding must be used instead.

Character Reference

(cont)

- ◆ **Example**

- ◆ *Character references*

```
<?xml version='1.0' encoding='US-ASCII' ?>
<Personne occupation='&#xe9;tudiant'>
  <nom>Martin</nom>
  <langue>Fran&#231;ais</langue>
</Personne>
```

- ◆ Character references appearing in element and attribute content

Well-Formed XML

- ◆ All XML must be well formed.
- ◆ A well-formed XML document is one in which, in addition to all the constructs being syntactically correct, there is exactly one top-level element, all open tags have a corresponding close tag or use the empty element shorthand syntax, and all tags are correctly nested (that is, close tags do not overlap).
- ◆ In addition, all the attributes of an element must have different names.
- ◆ If attributes are namespace qualified then the combination of namespace name and local name must be different.
- ◆ Similarly, all the namespace declarations of an element must be for different prefixes. All namespace prefixes used must have a corresponding namespace declaration that is in scope.

Well-Formed XML

(cont)

- ◆ **Examples**

- ◆ *Well-formed XML*

```
<?xml version='1.0' encoding='UTF-8' ?>
<p:Person xmlns:p='urn:example-org:People'>
  <name>Martin</name>
  <!--Young and spritely -->
  <age>33</age>
  <height units='inches'>64</height>
</p:Person>
```

- ◆ A well-formed XML document

Well-Formed XML

(cont)

- ◆ *XML that is not well formed*

```
<?xml version='1.0' encoding='UTF-8' ?>
<p:Person>
  <name>Martin</name>
  <age value='33'>A young <b><i>and</b></i>spritely
    person</age>
  <height units='inches' units='in'>64</height>
  <weight xmlns:x1='urn:example-org:People'
    xmlns:x2='urn:example-org:People'
    x1:units='stone' x2:units='shekels'>10</weight>
</p:Person>
<p:Person/>
```

- ◆ An XML document that is not well formed because it has two top-level elements, the `` and `<i>` tags inside the age element overlap, the height element has duplicate unqualified attribute names, the weight element has duplicate qualified attribute names, and the namespace prefix `p` is not in scope.