6 EXTENSIBLE MARKUP LANGUAGE (XML)

KEY OBJECTIVES

After completing this chapter readers will be able to—

- understand the importance of XML
- learn the basic building blocks of XML documents
- get an idea about well-formed and valid XML documents
- get an idea about how to display XML documents
- understand how name clashes are avoided using namespaces

6.1 COMMON USAGE

XML is the acronym of e**X**tensible **M**arkup **L**anguage. The XML standard was developed by the **W**orld **W**ide **W**eb **C**onsortium (W3C) in the late 1990's and since then, it has received a great hype. XML is an application profile or restricted form of the **S**tandard **G**eneralized **M**arkup **L**anguage (SGML). The primary purpose of this standard was to provide a way to store self-describing data easily. Self-describing data are those that describe both their structure and their content.

XML is used to describe structured data or information. The data are intended to be used by machines or people. HTML documents describe how data should appear on the browser's screen; they carry no information about the data. XML documents, on the other hand, describe the meaning of data. XML documents may also refer to presentation information.

XML is used as a primary means to manipulate and transfer structured data over the web. The content and structure of XML documents are accessed by a software module, called *XML processor*.

Similar to HTML documents, data are marked up by tags in XML documents. HTML supports a predefined fixed set of tags. Authors of HTML documents can use only those tags. XML allows us to define new tags and use them in XML documents to satisfy application requirement. Since more such new tags may be defined, XML is said to be extensible.

6.2 ROLE OF XML

One of the major difficulties in developing documents such as word processor documents, spreadsheets, and HTML documents is that they mix document content with structure and formatting. This makes managing content and designing difficult.

An XML document separates the contents from their presentation. It does not specify how the contents should be displayed. It only specifies the contents with their logical structure. The formatting task is imposed on an external style sheet. This helps authors of the XML documents to develop contents and find a logical relation among them without bothering about the formatting information. Since formatting is done on an external style sheet, the look and feel may be changed very easily by choosing different style sheets.

XML also promotes easy data sharing among applications. Since applications hold data in different structures, mapping data from one to another is a difficult task. XML can help us in this regard. Each data structure is mapped to an agreed-upon XML structure. This XML structure may then be used by other applications. So, each application has to deal with only two structures: its own and the XML structure. This way, applications can share data with others even if they are stored in a different format.

Each XML file, if written with a little care, can describe itself. So, far we have not given any example of XML documents. In spite of this, you can understand the meaning of the following XML document:

```
<?xml version="1.0"?>
<email>
  <to>parama@it.jusl.ac.in</to>
  <from>u_roy@it.jusl.ac.in</from>
  <subject>Reminder</subject>
  <body>Next BoS meeting will held on 10 June at 3:00 P.M.</body>
</email>
```

It is not difficult to understand that this XML document describes an email message. The message here is from `u_roy@it.jusl.ac.in` to `parama@it.jusl.ac.in` to remind the latter regarding the BoS meeting.

An XML document consists of the following parts:

- Prolog
- Body

6.3 PROLOG

This part of the XML document may contain the following parts:

- XML declaration
- Optional processing instruction
- Comments
- Document Type Declaration

6.3.1 XML Declaration

Every XML document should start with a one-line XML declaration. It is not mandatory, but W3C recommends it. The declaration describes the document itself. Here is a sample XML declaration:

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

The XML declaration is a processing instruction that tells the processing agent that the document is an XML document. The mandatory `version` attribute indicates the version used in this document. The current version is 1.0. The declaration may use two optional attributes: `encoding` and `standalone`.

`encoding`
It specifies the type of encoding scheme used in the document.

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

In this example, "UTF-8" (UTF stands for Unicode Transformation Format) is used which has the same character set as ASCII. XML parsers must support 8-bit and 16-bit Unicode encoding.

`standalone`
This optional attribute indicates whether the document can be processed as a standalone document or is dependent on other documents. If "yes" is specified, the XML document must not contain any external **D**ocument **T**ype **D**eclaration (DTD). The value "no" leaves the issue open, i.e., the XML document may or may not refer to external DTDs. If nothing is specified "no" is assumed.

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

Note that the attribute `standalone` is required if a DTD is used. Most often, XML documents use schema instead of DTD and hence the attribute `standalone` is not required.

6.3.2 Processing Instruction

Processing instructions start with `<?` and end with `?>`. They allow XML documents to contain special instructions that are used to pass parameters to the application. These parameters instruct the application about how to interpret the XML document. XML parsers do not take care of processing instructions, instead, they pass parameters to the underlying application. Processing instructions are not a textual part of the XML document. The XML declaration discussed in Section 6.3.1 is a processing instruction. Consider the following processing instruction:

```
<?xml-stylesheet href="simple.xsl" type="text/xsl"?>
```

This processing instruction states that the XML document should be transformed using the style sheet `simple.xsl`.

6.3.3 Comments

Like HTML, comments may appear anywhere in the XML documents. An XML comment starts with <!-- and ends with -->. Everything within these character sequences will be ignored by the parsers and will not be parsed. They are not part of the document's textual portion. Comments are used to explain the purpose of critical tags in an XML document. They take the following form:

```
<!--comment text-->
```

The following points should be remembered while using comments:

- Do not use the double hyphen '--' within comments as it might confuse the XML processor.
- Never place a comment inside an entity declaration.
- Never place a comment within a tag. This would result in poorly-formed XML.
- Never place a comment before the XML declaration. An XML declaration must always be the first line in any XML document.

6.3.4 Document Type Declaration

Although XML allows us to create new tags, it is probably not meaningful to use completely new tags that too in a completely arbitrary order. XML documents, to have a meaning, must have a logical structure created using a set of related tags.

The Document Type Declaration (or `DOCTYPE` declaration) is used to specify the logical structure of the XML document. The structure is specified by imposing constraint on what tags can be used and where. A parser reads this section and checks whether the XML document has been written according to the rules specified.

A Document Type Declaration may contain the following:

- Name of the root element
- Reference to an external DTD
- Element declaration
- Entity declaration

Detailed information about DTD may be found in Chapter 7.

6.4 BODY

This portion of the XML document contains textual data marked up by tags. It must have one element, called the *document element* or *root element*, which defines the content in the XML document. In an HTML document, the root element is `<html>`. Consider the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<greeting>Hello World!</greeting>
```

In this document, the name of the root element is `<greeting>`, which contains the text "Hello World".

The root element must be the top-level element in the document hierarchy and there can be one and only one root element. The following XML document is not valid as it has two top-level elements, `<greeting>` and `<message>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<greeting>Hello World!</greeting>
<message>World is beautiful!</message>
```

The root element contains other elements which, in turn, contain other elements and so on.

```
<?xml version="1.0" encoding="UTF-8"?>
<contact>
   <person>
      <name>B. S. Roy</name>
      <number>9674141547</number>
   </person>
   <person>
      <name>G. Mahapatra</name>
      <number>919441804070</number>
   </person>
</contact>
```

This XML document has the root element `<contact>` which has two `<person>` elements. Each `<person>` element has two elements, `<name>` and `<number>`.

6.5 ELEMENTS

An XML element consists of a starting tag, an ending tag, and its contents and attributes. The content may be simple text, or other elements, or both. Each element contains different types of data that are stored in the XML document.

The XML tags are very much similar to that of HTML tags. A tag begins with the less than character ('<') and ends with the greater than ('>') character. It takes the form <tag-name>. Every tag must have its corresponding ending tag. The ending tag looks exactly like the starting tag except that it includes a slash ('/') character before the tag name. All information in this element must be contained in the starting and ending tags. Following is an example of an XML element:

```
<greeting>Hello World!</greeting>
```

Here, `<greeting>` is the starting tag and `</greeting>` is the ending tag. The starting tag, the ending tag and everything between these two form an element. This element has only text content "Hello World!".

6.5.1 Anatomy of an Element

An element consists of an opening tag, a closing tag, and contents.

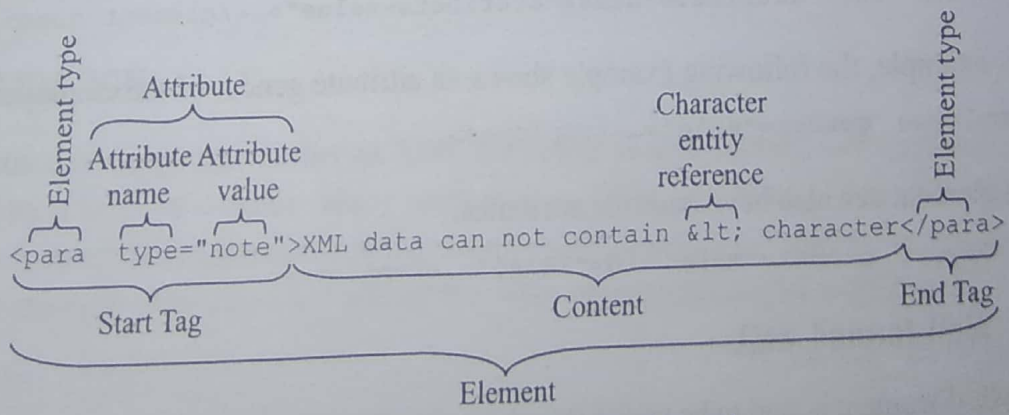

Figure 6.1 Structure of an element

Figure 6.1 shows the structure of an element. The name of the element is `para`. The opening tag is written within angular brackets. The closing tag is written exactly the same way, except that a slash (/) is added before the element name. Attributes are declared within the opening tag. For example, the para element has only one attribute `type` whose value is `note`. The attribute value must be quoted. An element can have an arbitrary number of attributes. The element content can be simply text, content, another element, or both. For the para element, the content is text. The text content may contain entities that refer to some piece of text to be substituted while processing. An entity starts with '&' and ends with ';' characters. XML specification defines five entities as shown in Table 6.1. XML document writers can also define their own entities in the DTD or XML declaration. Such declaration is discussed in Chapters 7 and 8.

6.5.2 Naming Rules

The following rules must be obeyed while selecting an element name:

- Names can only contain letters, digits, and some other special characters.
- Names cannot start with a number or punctuation marks.
- Names must not contain the string "xml", "XML", or "Xml".
- Names cannot contain white space(s).

6.5.3 Empty Elements

Empty elements are those that do not have any content. However, they can have attributes. According to the well-formedness constraint, every XML element must have a closing tag. So, a closing tag must be specified even for an empty element. Following is an example of an empty element:

```
<line width="100"></line>
```

XML specification provides a shorthand notation for empty elements as follows:

```
<line width="100"/>
```

6.6 ATTRIBUTES

Attributes are used to describe elements or to provide more information about elements. They appear in the starting tag of the element. The syntax for specifying an attribute in an element is:

```
<element-name attribute-name="attribute-value">…</element-name>
```

For example, the following example shows an attribute gender of the employee element.

```
<employee gender="male">…</employee>
```

An element can also have multiple attributes:

```
<employee gender="male" id="12345">…</employee>
```

6.6.1 Well-formed XML

An XML document is said to be well-formed if it contains text and tags that conform with the basic XML well-formedness constraints. Well-formed XML documents are not created by a rigid structural

definition but by their use. It frees authors from the fixed nature of XML and allows their imagination to prevail over restraint. Authors can extend existing documents by creating new elements that fit their applications. The only thing they have to remember is the well-formedness constraint.

The following rules must be followed by the XML documents to be well-formed:

An XML document must have one and exactly one root element.

Consider the following well-formed XML document. It has one root element `<contact>`, which contains one `<person>` element which, in turn, contains two elements, `<name>` and `<number>`.

```
<contact>
   <person>
      <name>B. S. Roy</name>
      <number>9674141547</person>
   </person>
</contact>
```

The following document is not well-formed as it has two top-level elements, `<contact>` and `<phonebook>`.

```
<contact>
   <person>
      <name>B. S. Roy</name>
      <number>9674141547</person>
   </person>
</contact>
<phonebook>
      <name>G. Mahapatra</name>
      <number>919441804070</number>
</phonebook>
```

All tags must be closed.

Every element must have a closing tag corresponding to its opening tag. The following document fragment is not well-formed as there is no closing tag for the opening tag `<name>`:

```
<!--Incorrect-->
<name>B. S. Roy
```

However, the following code is well-formed because the opening tag `<name>` has its corresponding closing tag `</name>`.

```
<!--Correct-->
<name>Mozart</name>
```

A tag must be closed even if it does not have any content (empty element). In HTML, some tags such as `<img>`, `<br>`, `<hr>` do not require any closing tag. However, in XML closing tags are always needed. So, the following syntax is incorrect in XML:

```
<br>
```

Its correct representation is

```
<br></br>
```

Alternatively, XML has an abbreviated version for it as follows:

```
<br/>
```

All tags must be properly nested.

Elements must not overlap. An ending tag must have the same name as the most recent unmatched tag. The following code is not well-formed as `</b>` appears when the most recent (closest) starting tag is `<i>`.

```
<b><i>This is incorrect nesting</b></i>
```

The following code shows the correct nesting:

```
<b><i>This is correct nesting</i></b>
```

XML tags are case-sensitive

Unlike HTML, XML element names are case-sensitive. So, `Message` and `message` refer to two different names. So, the following code is not well-formed as there is no ending tag corresponding to the starting tag `<Message>`.

```
<Message>This is incorrect</message>
```

The following two elements are well-formed:

```
<Message>This is correct</Message>
<message>This is also correct</message>
```

Attributes must always be quoted

The value of an attribute must be quoted by double inverted comma. In HTML, for most of the cases it works even if no quotation is used. But, it is mandatory for an XML document to be well-formed.

```
<speed unit="rpm">7200</speed>
```

The following syntax is incorrect because the value `rpm` is specified without any quotation marks:

```
<speed unit=rpm>7200</speed>
```

Certain characters are reserved for processing

Certain characters cannot be used in the XML document directly. For example, '<', '>', and "" cannot be used. So, the following syntax is incorrect as it confuses the XML parsers:

```
<!--Incorrect-->
<condition>if salary < 1000 then</condition>
```

XML provides entities for these special characters and may be used. XML parsers will substitute each entity by its actual value. An entity starts with an '&' character and ends with a ';'. The following code is correct.

```
<!--Correct-->
<condition>if salary &lt; 1000 then</condition>
```

6.6.2 Predefined Entities

In the W3C XML specification, five entities are defined, each of which represents a special character that cannot be used in the XML document directly. All XML processors must recognize those entities, whether they are declared or not. XML also allows us to define entities of arbitrary size in the DTD or schema.

Table 6.1 shows the predefined entities.

Table 6.1 Predefined entities

Entity Name	Entity number	Description	Character
&lt;	&#60;	less than	<
&gt;	&#62;	greater than	>
&amp;	&#38;	ampersand	&
&quot;	&#34;	quotation mark	"
&apos;	&#39;	apostrophe	'

6.6.3 Valid XML

Well-formed XML documents obey only basic well-formedness constraints. So, valid XML documents are those that

- are well-formed
- comply with rules specified in the DTD or schema.

The basic difference between a well-formed and a valid XML document is that the former is defined without any DTD or schema and the latter requires a DTD or schema. A DTD or schema specifies a set of rules, which the valid XML documents must follow. The rules usually specify the name and content of the element that can occur in the valid documents. XML parsers check whether the XML documents are developed using these rules.

Authoring, processing, storing, and displaying valid XML documents are easier as they follow a predefined structure. Data stored in the XML documents can be verified against the underlying DTD or schema. Use of style sheets is easy as they can be created using DTD or schema as opposed to a relatively unknown set of markups used in well-formed documents. Valid XML documents may be shared among a set of DTD or schema-aware applications.

6.7 VALIDATION

It is a method of checking whether an XML document is well-formed and conforms to the rules specified by a DTD or schema. Many tools are available to validate XML documents against DTD or schema. UNIX/Linux provides one such application called xmllint for this purpose. The following example shows how to validate the XML document sample.xml using this command:

```
xmllint --valid sample.xml
```

Java, JavaScript, and C/C++ also provide interfaces to validate XML documents. A detailed description about XML validation can be found in Chapter 9.

6.7.1 Elements or Attributes

Attributes are usually used to provide information that is not an integral part of the XML document. Attributes may always be converted to embedded elements to provide the same information. For example, the example depicted in Section 6.6 could be written as

```
<employee>
  <gender>male</gender>
  <id>12345</id>
  …
</employee>
```

There is no rigid rule that decides when to use elements and when to use attributes. However, it is recommended not to use attributes as far as possible due to the following reasons:

- Too many attributes reduce the readability of an XML document.
- Attributes cannot contain multiple values, but elements can.
- Attributes are not easily extendable for future changes.
- Attributes cannot represent logical structure, but elements together with their child elements can.
- Attributes are difficult to access by the parsers.
- Attribute values are not easy to check against DTD.

6.8 DISPLAYING XML

XML documents do not carry information about how to display the data. New tags can be added in the XML document. Web browsers do not have any idea about the tags used in the XML file. So, if you open an XML file in a browser, the entire content (data and tags) is displayed in a tree-like structure.

There are many ways to display data stored in an XML document. The two common methods are

- CSS (Cascading Style Sheets)
- XSL (eXtensible Stylesheet Language)

The use of CSS is exactly the same as HTML and was discussed in Chapter 5. XSL was specially designed for XML. Consider the following XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="books.xsl"?>
<bookstore>
  <book category="literature">
    <title lang="beng">Sanchoita</title>
    <author>Rabindranath Tagore</author>
    <year>2009</year>
```

```
      <price>200.00</price>
   </book>
   <book category="literature">
      <title lang="en">Gitanjali</title>
      <author>Rabindranath Tagore</author>
      <year>2008</year>
      <price>29.00</price>
   </book>
   <book category="WEB">
      <title lang="en">Essential XML</title>
      <author>Don Box</author>
      <year>2000</year>
      <price>150</price>
   </book>
</bookstore>
```

To link an XML document to an XSL style sheet, the following processing instruction

```
<?xml-stylesheet type="text/xsl" href="books.xsl"?>
```

The style sheet `books.xsl` looks like this.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method='html' version='1.0' encoding='UTF-8' indent=
   <xsl:template match="/">
      <html>
         <body>
            My Book Collection:
            <table border="1">
               <tr bgcolor="#9acd32">
                  <th>Title</th><th>Author</th><th>Year</th><th>Price
               </tr>
               <xsl:for-each select="bookstore/book">
               <tr>
                  <xsl:for-each select="./*">
                     <td><xsl:value-of select="."/></td>
                  </xsl:for-each>
               </tr>
               </xsl:for-each>
            </table>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

The syntax of XSL is discussed in Chapter 11. This time you simply write such a sty link it with the XML document. Now, if you open this document in Google Chrome, it lo shown in Figure 6.2.

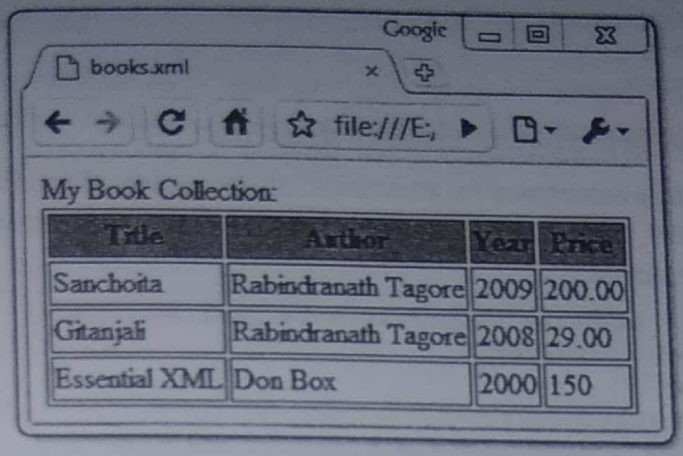

Figure 6.2 Displaying XML document using CSS

6.9 NAMESPACE

XML was developed to be used by many applications. If many applications want to communicate using XML documents, a potential problem may occur.

6.9.1 The Problem

In an XML document, element names and attribute names are selected by developers. According to the XML 1.0 specification, element and attribute names are unstructured flat strings. So, name conflicts may occur when merging XML documents from different developers to get a final XML document. Consider the following XML document (client.xml) which represents an HTML table of client-side technologies:

```
<table>
   <tr><td>JavaScript</td><td>VBScript</td></tr>
</table>
```

This XML document has the root element `table`. The following XML document (server.xml) carries information about a table of server-side technologies.

```
<table>
   <row><col>JSP</col><col>ASP</col></row>
</table>
```

This XML document also contains the root element `table`. Let us now merge these two XML documents to obtain a single XML document as follows:

```
<technology>
   <table>
      <tr><td>JavaScript</td><td>VBScript</td></tr>
   </table>
   <table>
      <row><col>JSP</col><col>ASP</col></row>
   </table>
</technology>
```

This new XML document has two `<table>` elements which have different content and meaning. So, if we query a parser to find the table containing server-side technologies, it fails. One simple, but not-too-flexible way to resolve this problem is to embed them in two different elements as follows:

```
<technology>
   <client>
      <table>
         <tr><td>JavaScript</td><td>VBScript</td></tr>
      </table>
   </client>
   <server>
      <table>
         <row><col>JSP</col><col>ASP</col></row>
      </table>
   </server>
</technology>
```

Now, the first and second table elements can be uniquely referred by referring to their parent element names, which are unique. Needless to say, this mechanism needs not only extra elements to be inserted, but also knowledge about the XML documents to be merged.

6.9.2 Solution

XML namespace provides a simple, straightforward, but elegant way to distinguish between element names in the XML document, no matter where they come from. XML namespace suggests that we use a prefix with every element. So, client.xml can now be written like this:

```
<c:table>
   <c:tr><c:td>JavaScript</c:td><c:td>VBScript</c:td></c:tr>
</c:table>
```

Similarly, we can write server.xml.

```
<s:table>
   <s:row><s:col>JSP</s:col><s:col>ASP</s:col></s:row>
</s:table>
```

Who guarantees that the prefixes used by different developer will be unique? One good idea is to use Uniform Resource Identifier (URI). URIs are unique all over the world. Usually we use Uniform Resource Locator (URL) to choose a unique name. The developers will then use their own unique URL as the prefix to every element they use. But, what happens if we use a long URL repeatedly in our XML documents? Will it not create a mess? XML namespace has the answer: instead of using an entire URL as the prefix for all elements, use a small prefix and make an association between this prefix and the URL. It means prefixes are just shorthand placeholders of URLs.

6.9.3 Binding

How to make this association between a prefix and a URL? This association is done in the starting tag using the reserved XML attribute `xmlns`. It takes the following form:

```
xmlns:prefix="URI"
```

Let us now rewrite the file client.xml.

```
<c:table xmlns:c="http://it.site.jusl.ac/client">
   <c:tr><c:td>JavaScript</c:td><c:td>VBScript</c:td></c:tr>
</c:table>
```

This declaration states that the prefix `c` is associated with a namespace whose value is `http://it.site.jusl.ac/client`. The prefix does not have any meaning, its only purpose is to point to the namespace name. The namespace declaration does not suggest nor require that the URL be used to access the associated resource. For example, the URL `http://it.site.jusl.ac/client` does not really represent any resource. The URLs are used only to make element names unique. Using a URL instead of a flat, simple string reduces the chance of different developers using duplicate names.

The actual name (W3C calls it expanded name or qualified name) is then obtained by using this prefix and a local name separated by ":". So, the name of the table element has the expanded name `c:table`. Essentially, it maps to

```
<{http://it.site.jusl.ac/client}table>
```

Note that the `xmlns:c` attribute is removed during this mapping. Not only the table element but also all the descendant elements are mapped as follows:

```
<{http://it.site.jusl.ac/client}table>
   <{http://it.site.jusl.ac/client}tr>
   <{http://it.site.jusl.ac/client}td>JavaScript</{http://it.site.jusl.ac/
client}td>
      <{http://it.site.jusl.ac/client}td>VBScript</{http://it.site.jusl.ac/
client}td>
   </{http://it.site.jusl.ac/client}tr>
</{http://it.site.jusl.ac/client}table>
```

Now, every element has a unique name. A prefix is needed in both starting tag as well as ending tag. The prefix simply indicates which namespace the elements belong to. The same prefix may be used to all of its descendants to indicate that they belong to the same namespace. This way, name clashes can be avoided using a combination of universally managed URI namespace with the vocabulary's local name.

The file client.xml can use a separate namespace as follows:

```
<s:table xmlns:s="http://it.site.jusl.ac/server">
   <s:row><s:col>JSP</s:col><s:col>ASP</s:col></s:row>
</s:table>
```

The two files can then be safely merged as follows:

```
<technology>
   <c:table xmlns:c="http://it.site.jusl.ac/client">
      <c:tr><c:td>JavaScript</c:td><c:td>VBScript</c:td></c:tr>
   </c:table>
   <s:table xmlns:s="http://it.site.jusl.ac/server">
```

```
      <s:row><s:col>JSP</s:col><s:col>ASP</s:col></s:row>
   </s:table>
</technology>
```

A namespace, in practice, is identified by a URL.

When defining a namespace for an element, all descendant elements inherit the same namespace and the associated prefix may be used. So, it is a common practice to define all namespaces within the root element.

```
<technology xmlns:c="http://it.site.jusl.ac/client" xmlns:s="http://
it.site.jusl.ac/server">
   <c:table>
      <c:tr><c:td>JavaScript</c:td><c:td>VBScript</c:td></c:tr>
   </c:table>
   <s:table>
      <s:row><s:col>JSP</s:col><s:col>ASP</s:col></s:row>
   </s:table>
</technology>
```

However, namespaces declared explicitly in an element overrides the namespace obtained from its parent.

6.9.4 Namespace Rules

The `xmlns` attribute identifies the namespace and makes association between a prefix and the created namespace. Many prefixes may be associated with one namespace. The following example illustrates this:

```
<small:root xmlns:small="http://it.site.jusl.ac" xmlns:capital="http://
it.site.jusl.ac">
   <small:a>
      <small:b/>
   </small:a>
   <capital:A>
      <capital:B/>
   </capital:A>
</small:root>
```

In this example, elements are from the same namespace but use different prefixes. Similarly, elements may use the same prefix, but belong to different namespaces.

```
<root>
   <it:a xmlns:it="http://it.site.jusl.ac/small" >
      <it:b/>
   </it:a>
   <it:A xmlns:it="http://it.site.jusl.ac/capital">
      <it:B/>
   </it:A>
</root>
```

The elements `<it:a>` and `<it:b>` belong to the namespace `http://it.site.jusl.ac/small`, but `<it:A>` and `<it:B>` belong to the namespace `http://it.site.jusl.ac/capital`. Note that namespace defined in an element is only valid within the element and its descendant elements.

6.9.5 Default Namespace

Namespaces may not have their associated prefixes. These are called default namespaces. In such cases, a blank prefix is assumed for the element and all of its descendants.

```
<root xmlns="http://it.site.jusl.ac">
   <a>
      <b/>
   </a>
   <A>
      <B/>
   </A>
</root>
```

In this example, all the elements belong to the default namespace `http://it.site.jusl.ac`. Even if a default namespace is used for an element, its children can still define their own namespaces.

```
<root xmlns="http://it.site.jusl.ac">
   <small:a xmlns:small="http://it.site.jusl.ac/small">
      <small:b/>
   </small:a>
   <capital:A xmlns:capital="http://it.site.jusl.ac/capital">
      <capital:B/>
   </capital:A>
</root>
```

Even for the default namespace, a value of the attribute `xmlns` must be specified. Otherwise, the namespace remains undeclared.

```
<root xmlns="http://it.site.jusl.ac">
   <a xmlns="">
      <b/>
   </a>
   <A xmlns="http://it.site.jusl.ac/capital">
      <B/>
   </A>
</root>
```

In this example, the default namespace for the element `<a>` remains undeclared. However, the default namespace for the element `<A>` is declared correctly.

Attributes may be assigned to namespaces also.

```
<small:root xmlns:small="http://it.site.jusl.ac/small" xmlns:capital="http:/
/it.site.jusl.ac/capital">
   <small:a small:id="aaa">
      <small:b/>
   </small:a>
   <capital:A capital:ID="AAA">
      <capital:B/>
   </capital:A>
</small:root>
```

Attributes may belong to a namespace different from that of their embedding element.

```
<small:root xmlns:small="http://it.site.jusl.ac/small" xmlns:capital="http:/
/it.site.jusl.ac/capital">
   <small:a capital:ID="aaa">
      <small:b/>
   </small:a>
   <capital:A small:id="AAA">
      <capital:B/>
   </capital:A>
</small:root>
```

In this example, the `ID` attribute belongs to a namespace different from that of its embedding element. Similarly, the `id` attribute belongs to a namespace different from that of its embedding element.

Attributes without any prefix do not belong to any namespace.

```
<small:root xmlns:small="http://it.site.jusl.ac/small" xmlns:capital="http:/
/it.site.jusl.ac/capital">
   <small:a ID="aaa">
      <small:b/>
   </small:a>
   <capital:A id="AAA">
      <capital:B/>
   </capital:A>
</small:root>
```

In this example, attributes `ID` and `id` do not belong to any namespace. The attributes do not belong to the default namespace even if a default namespace is defined in the embedded element.

```
<small:root xmlns:small="http://it.site.jusl.ac/small" xmlns:capital="http:/
/it.site.jusl.ac/capital">
   <small:a>
      <b xmlns:capital="http://it.site.jusl.ac" id="bbb"/>
   </small:a>
   <capital:A>
      <capital:B/>
   </capital:A>
</small:root>
```

In this example, the `id` attribute of the `<b>` element does not belong to the default namespace even if the default namespace is defined for `<b>`.