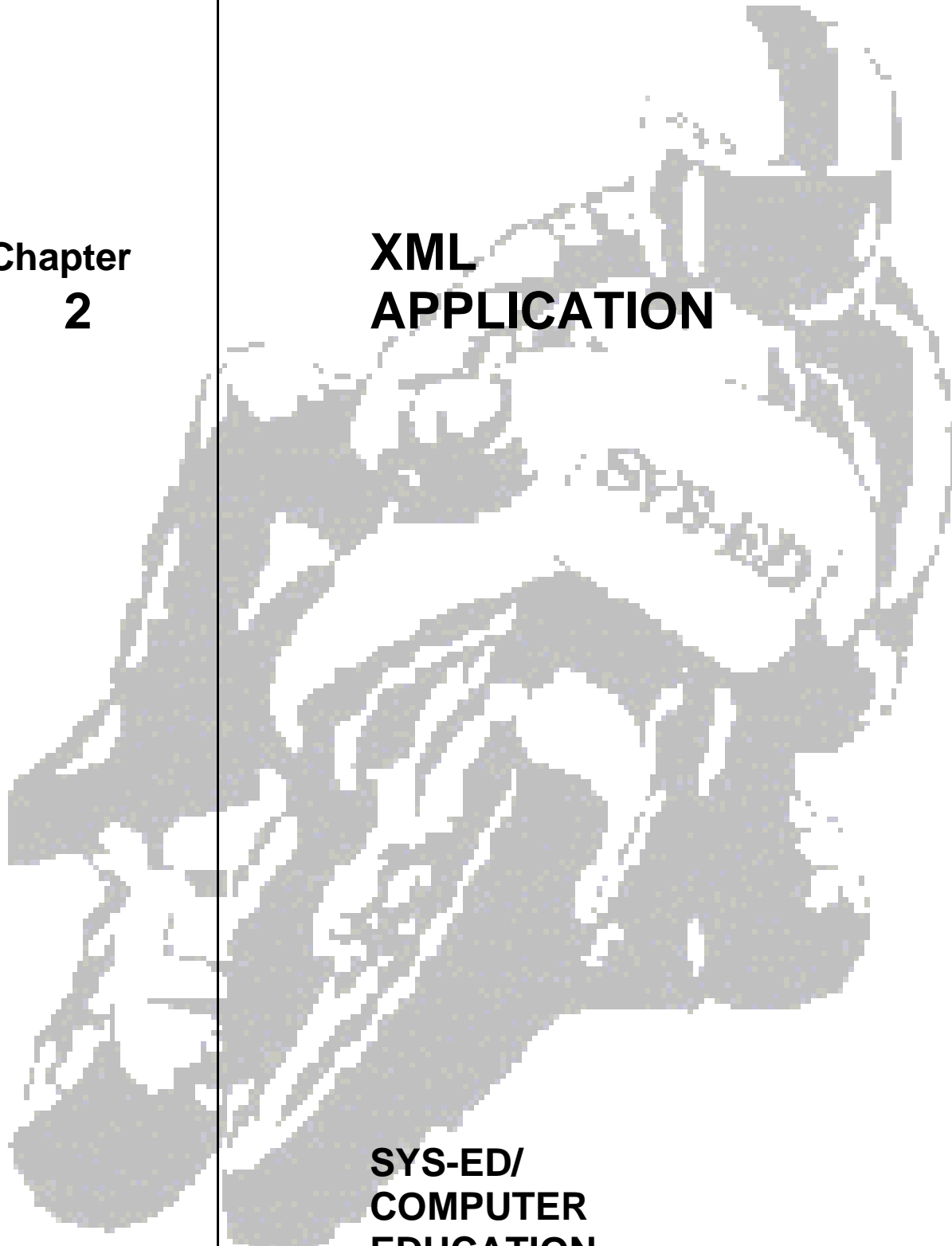


**Chapter
2**

**XML
APPLICATION**



**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

Objectives

You will learn:

- C How to create an XML document.
- C The role of the document map, prolog, and XML declarations.
- C Standalone declarations.
- C Style sheet processing instructions.
- C DOCTYPE declaration.
- C Textual content.
- C Elements.
- C Assigning meaning to XML tags.
- C Naming conventions.
- C Document: XML declaration and root element.
- C Organization of the XML data.
- C Usage of tags in XML
- C Understand document architecture.
- C Create and debug XML documents.
- C Syntax of all the different XML statements.
- C Organization of XML data.

1 XML Document Creation

XML holds data, the structure of data and classification of the information inside the document. In other words, an XML document contains information about itself.

XML achieves this through the use of custom tags and these custom tags make it superior to HTML which only has a few tags for display. However, XML does far more than simply display information.

Example:

XML document

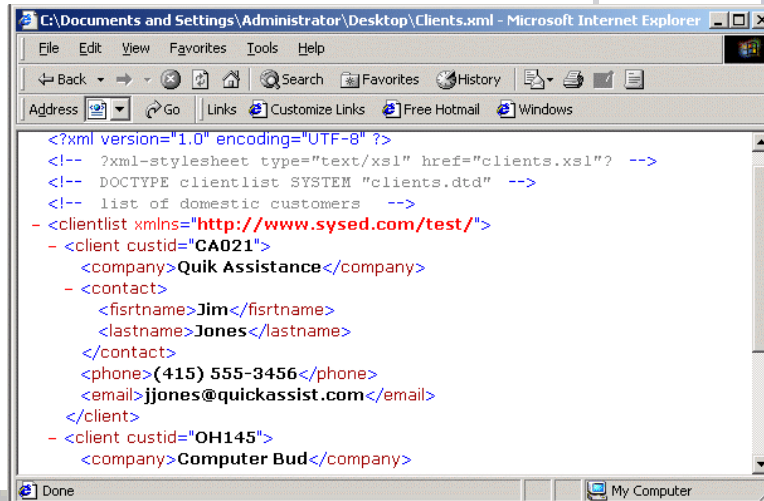
```
<clientlist xmlns="http://www.sysed.com/test">
  <client custid="CA021">
    <company>Quik Assistance</company>
    <contact>
      <firsrtname>Jim</firsrtname>
      <lastname>Jones</lastname>
    </contact>
    <phone>(415) 555-3456</phone>
    <email>jjones@quickassist.com</email>
  </client>

  <client custid="OH145">
    <company>Computer Bud</company>
    <contact>
      <firsrtname>John</firsrtname>
      <lastname>Springer</lastname>
    </contact>
    <phone>(918) 555-3432</phone>
    <email>jspringer@computerbud.com</email>
  </client>
</clientlist>
```

This is a well-formed XML document; which be typed in any text editor. XML has special terms for characterizing documents that it considers “good” depending on exactly which set of rules they satisfy.

2 XML File: Viewing in a Web Browser

The file can be opened directly in a browser that supports XML such as an Internet Explorer.



What appears on the screen will vary from browser to browser. The document on this page is reasonably well formatted and provides a syntax colored view of the document's source code. The browser must be informed about what it is expected to do with each element by using a style sheet.

3 Document Map

Prologs

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- ?xml-stylesheet type="text/xsl" href="clients.xsl"?-->
<!-- DOCTYPE clientlist SYSTEM "clients.dtd"-->
<!-- list of domestic customers -->
```

Elements

```
<clientlist xmlns="http://www.sysed.com/test">
  <client custid="CA021">
    <company>Quik Assistance</company>
    <contact>
      <firsrname>Jim</firsrname>
      <lastname>Jones</lastname>
    </contact>
    <phone>(415) 555-3456</phone>
    <email>jjones@quickassist.com</email>
  </client>
  <client custid="OH145">
    <company>Computer Bud</company>
    <contact>
      <firsrname>John</firsrname>
      <lastname>Springer</lastname>
    </contact>
    <phone>(918) 555-3432</phone>
    <email>jspringer@computerbud.com</email>
  </client>
</clientlist>
```

3.1. Prolog

The prolog refers to the information that appears before the start tag of the document or root element. It includes information that applies to the document as a whole, such as character encoding, document structure, and style sheets.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="clients.xsl"?>
<!DOCTYPE clientlist SYSTEM "clients.dtd">
<!-- list of domestic customers -->
```

Processing instructions and comments can appear in the prolog.

3.2. XML Declaration

The XML declaration typically appears as the first line in an XML document. The XML declaration is not required; however when it is used it must be the first line in the document and no other content or white space can precede it.

The XML declaration in the document map consists of the following:

| | | |
|----------------|--|--|
| version number | <?xml version="1.0"?> | This is mandatory. Although the number will change for future versions of XML, 1.0 is the current version. |
| encoding | <?xml version="1.0" encoding="UTF-8"?> | This is optional. When used, the encoding declaration must appear immediately after the version information in the XML declaration, and it must contain a value representing an existing character encoding. |
| standalone | <?xml version="1.0" encoding="UTF-8"? standalone="yes"?> | As with the encoding declaration, the standalone declaration is optional. If used, the standalone declaration must appear last in the XML declaration. |

3.3. Encoding Declaration

The encoding declaration identifies which encoding is used to represent the characters in the document.

Although XML parsers can determine automatically whether a document uses the UTF-8 or UTF-16 Unicode encoding, this declaration should be used in documents that support other encodings.

Example:

This encoding declaration is for a document that uses the ISO-8859-1 (Latin 1).

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

Case in the value specified is not considered by the encoding declaration. "ISO-8859-1" is the equivalent of "iso-8859-1".

The following is the encoding declaration for a document that uses the Japanese encoding method Shift-JIS.

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

3.4. Standalone Declaration

The standalone declaration indicates whether a document relies on information from an external source, such as external document type definition (DTD), for its content.

If the standalone declaration has a value of "yes", the parser will report an error when the document references an external DTD or external entities.

Leaving out the standalone declaration produces the same result as including a standalone declaration of "no". The XML parser will accept external resources, if there are any, without reporting an error.

4 Processing Instructions

Processing instructions can be used to pass information to applications in a way that escapes most XML rules. Processing instructions do not have to follow much internal syntax, may include markup characters without escaping them, and can appear anywhere in the document outside of other markup. They can appear in the prolog, including the document type definition - DTD, in textual content, or after the document. Their appearance is not noted by schema or DTD processors.

Processing instructions must begin with an identifier called a target, which follows rules similar to those for element and attribute names. Processing instruction targets are case-sensitive and must start with a letter or underscore. The rest of the target can contain letters, digits, hyphens, underscores, periods, and colons. Any valid XML textual characters can appear after the target.

Worldwide Web Consortium Namespaces in XML recommends that the use of colons in processing instruction names be avoided.

The following is the processing instruction syntax in the document map.

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

Processing of the contents ends immediately after the string `?>` is encountered.

5 Style Sheet Processing Instructions

The `xml-stylesheet` processing instruction must appear in the prolog, before the document or root element. Multiple processing instructions can appear, which can be useful with cascading style sheets; but most browsers use the first supported style sheet and ignore the rest.

The syntax for a style sheet processing instruction is:

```
<?xml-stylesheet type="type" href="uri"?>
```

| | |
|------|--|
| type | text/css to link to a cascading style sheet file. or text/xsl to link to an XSLT file. |
| uri | Uniform Resource Identifier (URI) of the style sheet. This URI is relative to the location of the XML document itself. |

Example:

This code is an `xml-stylesheet` processing instruction identifying a style sheet built using a cascading style sheet.

```
<?xml-stylesheet href="/style.css"
  type="text/css" title="default stylesheet"?>
```

Example:

The code is an `xml-stylesheet` processing instruction identifying a style sheet built using Extensible Stylesheet Language- XSL.

```
<?xml-stylesheet href="/style.xsl"
  type="text/xsl" title="default stylesheet"?>
```

5.1. DOCTYPE Declaration

The DOCTYPE declaration provides a space for a document to identify its root element and document type definition - DTD by reference to an external file, through direct declarations, or both.

A DOCTYPE declaration can contain:

- C The name of the document or root element.
This is required if the DOCTYPE declaration is used.
- C System and public identifiers for the DTD that can be used to validate the document structure.
If a public identifier is used, a system identifier must also be present.
- C An internal subset of DTD declarations.
The internal subset appears between square brackets ([]).

A DOCTYPE declaration is mandatory when the document is to be processed in a validating environment. In order to be valid, the DOCTYPE declaration must identify a DTD that corresponds to the document structure of the document. Nonvalidating parsers will accept documents without DOCTYPE declarations.

The simplest DOCTYPE declaration identifies only the root element of the document.

Example:

```
<!DOCTYPE rootElement>
```

Typically, documents that use the DOCTYPE declaration reference an external document containing the declarations that make up the DTD.

The following code can be used to identify the external DTD:

```
<!DOCTYPE rootElement SYSTEM "URReference">
```

The URReference points to a file containing the declarations.

```
<!DOCTYPE rootElement PUBLIC "PublicIdentifier" "URReference">
```

The PublicIdentifier provides a separate identifier that some XML parsers can use to reference the DTD in place of the URIreference. This is useful when the parser is used on a system without a network connection or where that connection would slow down processing significantly.

DOCTYPE declarations can also include declarations directly, in what is referred to as the internal subset.

When a DOCTYPE declaration includes the entire DTD directly, without reference to external files, it uses the following syntax.

```
<!DOCTYPE rootElement [  
  declarations  
>
```

When the DOCTYPE declaration includes declarations that are to be combined with external files or the external subset, it uses the following syntax.

```
<!DOCTYPE rootElement SYSTEM "URIreference"[  
  declarations  
>
```

Or

```
<!DOCTYPE rootElement PUBLIC "PublicIdentifier" "URIreference"[  
  declarations  
>
```

5.2. Comments

Content that is not intended for the XML parser, such as notes about document structure or editing, can be included in a comment. Comments begin with a `<!--` and end with a `-->`.

Example:

```
<!-- list of domestic customers -->
```

Comments can appear in the document prolog, including the document type definition - DTD; after the document; or in the textual content. Comments cannot appear within attribute values; they cannot appear inside of tags.

The parser considers the comment finished when it encounters a `-->`; it then resumes processing the document as normal XML. For this reason, the string `-->` cannot appear inside of a comment. Apart from that restriction, any legal XML characters can appear in a comment, much like a CDATA section. This makes them very useful for removing XML comment from the stream seen by the parser without removing the content from the document.

The following comments can be used to strip out markup temporarily.

```
<!-- ?xml-stylesheet type="text/xsl" href="clients.xsl"?-->  
<!-- DOCTYPE clientlist SYSTEM "clients.dtd"-->
```

5.3. Textual Content

As a result of its support for the Unicode character set, XML supports a range of characters, including letters, digits, punctuation, and symbols. Most control characters and Unicode compatibility characters are not allowed.

Since XML relies on `<`, `>`, and `&` to delimit markup, these characters should be represented using character and entity references or be confined to CDATA sections.

6 Elements

Elements form the backbone of XML documents. Elements identify named sections of information and are built using markup tags that identify the name, start, and end of the element.

Elements can also contain attribute names and values, which provide additional information about your content.

All elements must have names. Element names are case-sensitive and must start with a letter or underscore. An element name can contain letters, digits, hyphens, underscores, and periods.

Tags establish boundaries around the content, if any, of the element.

Example:

```
<contact>
  <firstname>Jim</firstname>
  <lastname>Jones</lastname>
</contact>
```

The <contact> element contains two other elements, <firstname> and <lastname>. The <firstname> element contains the text 'Jim' while the <lastname> element contains the text 'Jones'.

Empty tags are used to indicate elements that have no textual content, though they may have attributes. The HTML `img` and `br` elements are examples of empty elements. Empty tags can be used as a shortcut when there is no content between the start and end tags of a document. Empty tags look like start tags, except that they contain a slash (/) before the closing >.

6.1. Assigning Meaning to XML Tags

There are three kinds of meanings associated with Markup tags:

| | |
|-----------|--|
| Structure | <p>Structure divides documents into a tree of elements.</p> <p>Structure merely expresses the form of the document, without regard for differences between individual tags and elements.</p> |
| Semantics | <p>Semantics relates the individual elements to the real world outside of the document itself.</p> <p>Semantic meaning exists outside the document.</p> |
| Style | <p>Style specifies how an element is displayed.</p> <p>Computers are better at understanding style than semantic meaning. In XML, style meaning is applied through style sheets.</p> |

It is better to pick tags that more closely reflect the meaning of the information they contain. Some fields are working on creating industry standard tag sets. These should be used when appropriate. However, most tags will need to be made up as needed.

Examples:

| | | | |
|------------|------------|------------|----------|
| <MOLECULE> | <INTEGRAL> | <EMPLOYEE> | <SALARY> |
| <author> | <email> | <planet> | <sign> |
| <Bill> | <plus/> | <Hillary> | <plus/> |

6.2. Naming Conventions

XML element names are flexible and can contain any number of letters and digits in either upper or lowercase.

XML tags can look like any of the following:

```
<SEASON>  
<Season>  
<season>  
<season1998>  
<Season98>  
<season_98>
```

There are several thousand more variations. XML doesn't care whether all uppercase, all lowercase, mixed-case with internal capitalization, or some other convention is used.

6.3. Document: XML Declaration and Root Element

XML documents may be recognized by the XML declaration. This is a processing instruction placed at the start of all XML files that identifies the version in use. The only version currently understood is 1.0.

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

Every good XML document must have a root element. This is an element that completely contains all other elements of the document. The root element's start tag comes before all other elements' start tags, and the root element's end tag comes after all other element's end tags.

Example:

The document will look like this:

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

The XML declaration is not an element or a tag. It is a processing instruction. Therefore, it does not need to be contained inside the root element, SEASON . However, every element in this document will go in between the <SEASON> start tag and the </SEASON> end tag.

This choice of root element means that multiple seasons will not be able to be stored in a single file. In order to do this it will be necessary to define a new root element that contains seasons.

Example:

```
<?xml version="1.0" ?>
<DOCUMENT>
  <SEASON>
  </SEASON>
  <SEASON>
  </SEASON>
</DOCUMENT>
```

In order to identify which season is being talked about it. It will be necessary to give the SEASON element a YEAR child.

Example:

```
<?xml version="1.0" ?>
<SEASON>
  <YEAR>
    1998
  </YEAR>
</SEASON>
```

Indentation is used for indicating the levels of containership of the elements and the contents of the elements is good coding style, but it is not required. White space in XML is not especially significant.

Example:

```
<?xml version="1.0" ?>
<SEASON>
<YEAR>1998</YEAR>
</SEASON>
```

The document can be compressed still further, even down to a single line, but with a corresponding loss of clarity.

Example:

```
<?xml version="1.0" ?><SEASON><YEAR>1998</YEAR></SEASON>
```

This version will be much harder to read and understand.

7 Organization of the XML Data

XML is based on a containment model. Each XML element can contain text or other XML elements called its children. A few XML elements may contain both text and child elements, though in general this is bad form and should be avoided wherever possible.

Frequently there will be more than one way to organize the data. One of the advantages of XML is that it makes it fairly straightforward to write a program that reorganizes the data in a different form.

The XML model is essentially a hierarchical database, and it shares all the disadvantages and the advantages of that data model.

In XML the tags can be made up as when necessary.

8 Namespaces

Namespaces are a mechanism by which element and attribute names can be assigned to groups. They are most often used when combining different vocabularies in the same document.

It is the recommended practice that the word `xml` not be used as a namespace prefix since it is often used in reserved attributes like `xml:space`.

Namespace identifiers are, by convention, assigned to URLs. This is not a requirement. A URL is used because a namespace has to be assigned some kind of unique identifier. URLs are unique. They often contain information about the company or organization.

Many IT professionals believe that URLs are not really meant to be used as identifiers. Resources are moved around often, and URLs change. However, in the immediate future, namespace assignments are here to stay.

9 Entities

Entities are placeholders in XML. They are declared in the document prolog or in a DTD, and can be referred to many times in the document. Different types of entities have different uses.

Characters can be substituted that are difficult or impossible to type with character entities. Content that lives outside of a document with external entities can be pulled in. Rather than type the same thing in multiple times, such as boilerplate, general entities can be defined.

In the family tree of entity types, the two major branches are parameter entities and general entities. Parameter entities are used only in DTDs

An entity consists of a name and a value. When an XML parser begins to process a document, it first reads a series of declarations, some of which define entities by associating a name with a value.

The value is anything from a single character to a file of XML markup. As the parser scans the XML document, it encounters entity references, which are special markers derived from entity names. For each entity reference, the parser consults a table in memory for something with which to replace the marker. It replaces the entity reference with the appropriate replacement text or markup, then resumes parsing just before that point, so the new text is parsed too.

Any entity references inside the replacement text are also replaced; this process repeats as many times as necessary.

10 Character Entities

Entities that contain a single character are called character entities.

These fall into three major groups:

| | | |
|-------------------------------|-----------------------------|------------------------|
| Predefined character entities | Numbered character entities | Mixed-Content Entities |
|-------------------------------|-----------------------------|------------------------|

10.1. Predefined Character Entities

Some characters cannot be used in the text of an XML document because they conflict with the special markup delimiters. The XML specification provides the following predefined character entities; therefore the characters can be safely used:

| Name | Value |
|------|-------|
| amp | & |
| apos | ' |
| gt | > |
| lt | < |
| quot | " |

10.2. Numbered Character Entities

XML supports Unicode. Any Unicode character can be used in a document. A problem is how to enter a nonstandard character from a keyboard with less than 100 keys, or how to represent one in a text-only editor display. One solution is to use a numbered character entity in a reference to a character is by its number in the Unicode character set.

10.3. Mixed-Content Entities

Entity values aren't limited to a single character, of course. The more general mixed-content entities have values of unlimited length and can include markup as well as text.

These entities fall into two categories:

| | |
|----------|---|
| internal | For internal entities, the replacement text is defined in the entity declaration. |
| external | For external entities, it is located in another file. |

11 CDATA Sections

When characters are marked up frequently in text, it may be tedious to use the predefined entities `<`, `>`, and `&`. They require typing and are generally hard to read in the markup.

The CDATA section is an acronym for "character data," which means "not markup." The parser has been informed that this section of the document contains no markup and should be treated as regular text. The only thing that cannot go inside a CDATA section is the ending delimiter (`]]>`). In order that to happen a predefined entity would have to be used and written as `]]>`.

A CDATA section begins with the nine-character delimiter `<![CDATA[` (1), and it ends with the delimiter `]]>` (3). The content of the section (2) may contain markup characters (`<`, `>`, and `&`); but they are ignored by the XML processor.

12 Processing Instructions

Presentation information should be kept out of a document whenever possible. This information applies only to a specific XML processor and may be irrelevant or misleading to others.

The prescription for this kind of information is a processing instruction. It is a container for data that is targeted toward a specific XML processor.

