

**Naval Surface Warfare Center
Carderock Division
Bethesda, Maryland 20817-5700**

Department Technical Memorandum

7 June 2002

Guidelines for Converting an SGML DTD to an XML DTD

by

Mr. Donald Gignac

**Technology Implementation Support Team
Technical Information Systems Department
Total Ship Systems Directorate**

**Distribution Authorized to U.S. Government
Agencies and their Contractors Only;
Other Requests for this Document Shall be
Referred to NSWC CD, Code 205**

Guidelines for Converting an SGML DTD to an XML DTD

The following summary of the differences between XML (Extensible Markup Language) and SGML (Standard Generalized Markup Language) is based largely on material taken from James Clark's W3C note "Comparison of SGML and XML" (<http://www.w3.org/TR/NOTE-sgml-xml>). Additional material has been added to clarify points of SGML syntax that may not be familiar to CALS SGML users. Emphasis has been placed on those areas where the CALS SGML usage is affected.

For information on SGML syntax, see the annotated ISO 8879 specification in Charles Goldfarb's "The SGML Handbook" (Clarendon Press, 1990).

For information on the CALS implementation of SGML, see MIL-PRF-28001C "Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text" (May 2, 1997) and MIL-HDBK-28001 "Application of MIL-PRF-28001 Using Standard Generalized Markup Language (SGML)" (June 30, 1995). Both are available at "<http://navycals/28001/sgmlstds.html>".

For more information on XML see the latest XML specification at the W3C Web site "<http://www.w3.org/TR/REC-xml>" and Bob DuCharme's book "XML The Annotated Specification" (Prentice-Hall, 1999).

The "Web SGML Adaptations Annex" to ISO 8879 is "Appendix K (normative): Web SGML Adaptations" (<http://www.ornl.gov/sgml/sc34/document/0029.htm>). It enables the definition of XML as an SGML application, i.e., a "pure" subset of SGML. When we refer to SGML below, we mean the "traditional" pre-Annex K SGML that is, SGML applications that do not make use of the extensions to SGML defined in Annex K. XML differs from SGML in the following respects.

1. XML does NOT enable the following optional features of SGML.

1.1 DATATAG: This is an obscure and little-used feature of SGML that is NOT used in CALS SGML.

1.2 OMITTAG: This feature is USED in CALS SGML, and its non-availability in XML could cause a problem for some CALS applications. Instances that have been tagged using the tag minimization afforded by OMITTAG will have to have the omitted tags inserted when they are converted to XML.

1.3 RANK: This is an obscure and little-used feature of SGML that is NOT used in CALS SGML.

1.4 LINK (SIMPLE, IMPLICIT and EXPLICIT): This is a complicated and little-used feature of SGML that is NOT used in CALS SGML.

1.5 CONCUR: This is an obscure and little-used feature of SGML that is NOT used in CALS SGML.

1.6 SUBDOC: This feature is USED in CALS SGML. The SGML Declaration in MIL-PRF-28001C now allows use of the SUBDOC feature with a level of 2. CALS SUBDOC entities will have to be converted into external NDATA (non-SGML) entities where the data content notation is SGML or XML. The names of these entities will have to be values of ENTITY or ENTITIES type attributes.

1.7 FORMAL: This feature is USED in CALS SGML. XML PUBLIC identifiers do NOT have to conform to the "FORMAL" syntax defined in [79] "formal public identifier" through [90] "public text display version" of clause 10.2 "Formal Public Identifier" of ISO 8879. Instead the far more lenient syntax of [74]

Guidelines for Converting an SGML DTD to an XML DTD

"public identifier" in clause 10.1.6 "External Identifier" of ISO 8879 applies. For example,

```
PUBLIC "XML DTD"
```

is an acceptable XML PUBLIC identifier. This will not cause a problem for CALS PUBLIC identifiers. However, all XML PUBLIC identifiers will have to be followed by system literals, for example

```
PUBLIC "XML DTD" "C:\xml\xml.dtd"
```

2. XML has a unique syntax that differs from the reference concrete syntax of SGML as follows.

2.1 No short reference delimiters are declared in the SGML Declaration for XML. Moreover, the SHORTREF and USEMAP declarations used to define short references in SGML are not included in the XML declarations defined in the XML specification. As a result short references cannot occur in XML documents.

2.2 The PIC (processing instruction close) delimiter is "?>". The terminating ">" of all SGML processing instructions will have to be changed to "?>". Moreover, the string "?>" can not be used for any purpose in an XML processing instruction other than to terminate it.

2.3 All capacities and most of the quantities have been assigned the value of "99999999". For all practical purposes they are unlimited. However, the "NORMSEP" quantity is still two points. The "BSEQLen", "DTAGLen", and "DTEMPLen" quantities are not used in XML.

2.4 In the SGML Declarations for XML, "NAMECASE GENERAL" is "NO". That is, all alphabetic name characters used in element type and attribute names, reserved names, and name tokens are case sensitive in XML. For example, this means that "nosuch", "Nosuch", and "NOSUCH" refer to three different element type names, attribute names, or name tokens. Moreover, all reserved names such as ELEMENT, ATTLIST, IMPLIED, REQUIRED, ENTITY, CDATA, etc. must always be upper case in XML. For example, the ATTLIST declaration

```
<!attlist nosuch att cdata #implied>
```

contains three XML syntax errors with regard to reserved name use. In SGML, only entity names are case sensitive.

2.5 The underscore "_" and colon ":" are name characters in XML. However, the use of ":" in XML names is not recommended at present since it is "reserved for experimentation with name spaces". See the note in clause 2.3 "Common Syntactic Concepts" in the XML specification.

2.6 XML names should not begin with the eight strings "XML", "xml", "XmL", "XmL", "xMl", "xML", "XML", "xml" since this class of names is "reserved for standardization". See clause 2.3 "Common Syntactic Comments" of the XML specification.

2.7 The following short tagging practices enabled by the SHORTTAG feature of SGML are not allowed in XML.

Unclosed start-tags: Short tagging allows the elimination of the tag close ">" for start-tags under certain conditions, e.g.,

Guidelines for Converting an SGML DTD to an XML DTD

`<item<paratext>`

Unclosed end-tags: Short tagging allows the elimination of the tag close ">" for end-tags under certain conditions, e.g.,

`</paratext</item>`

Empty start-tags: Short tagging allows the substitution of "<>" for a start-tag under certain conditions, e.g.,

`<topic>QWERTY</topic><>ASDFG</topic>`

Empty end-tags: Short tagging allows the substitution of "</>" for an end-tag under certain conditions, e.g.,

`<paratext>QWERTY</>`

Undelimited attribute values: Short tagging allows the undelimited specification of an attribute value when the attribute value consists only of name characters, e.g.,

`<chapter version=1.1 state=open loc="C:\chaps\chap1.txt">`

The "loc" attribute value must be delimited because ":" and "\" are not CALS SGML characters.

Omission of attribute name: When an attribute's values are defined in terms of a name token group, e.g.,

`<!ATTLIST valve status (open | closed) #IMPLIED`

short tagging allows that attribute to be specified in terms of the value only, e.g.,

`<valve open>`

2.8 In XML, there are two alternate forms for element markup with no content. An element can have no content: (1) if the element type is declared EMPTY, or (2) the element type has optional content that is not present.

The first form is the empty-element tag which is merely the element type's start-tag terminated with ">" instead of ">". For example, the XML empty-element tag for the EMPTY element type "graphic" is

`<graphic boardno="fig.cgm"/>`

The second form consists of placing the element type's end-tag after the element type's start-tag, for example

`<graphic boardno="fig.cgm"></graphic>`

However, for compatibility with older SGML systems, it is recommended that: (1) the empty-element tag be used for EMPTY element types; and (2) the empty-element tag be used only for EMPTY element types. See clause 3.1 "Start-Tags, End-Tags, and Empty-Element Tags" and "for operability" in clause 1.2 "Terminology" in the XML specification.

Guidelines for Converting an SGML DTD to an XML DTD

3. The following restrictions pertain to XML and not SGML.

3.1 The following XML restrictions pertain to entity references.

3.1.1 In XML, general entity references are always terminated with ";". SGML allows the ";" in a general entity reference to be omitted if the entity name is followed by anything other than a name character.

3.1.2 In XML, external data entities may not be referenced in content. This prohibits the referencing of an entity such as

```
<!ENTITY statement1 SYSTEM "C:\states\state1.txt" CDATA CALS>
```

in parsed character data (#PCDATA content). However, EMPTY element types with ENTITY type attributes could replace such entity references. With regard to this example, first define the "statement" element type

```
<!ELEMENT statement EMPTY>
<!ATTLIST statement type ENTITY #REQUIRED>
```

in the DTD. Then provide for its occurrence in the appropriate content models. Finally, replace all "&statement1;" entity references in content with the tag

```
<statement type="statement1">
```

3.1.3 In XML, an element in the replacement text of a general entity reference in content can not be split. This means that if a start-tag occurs in the replacement text of a general entity, the corresponding end-tag must be there too. This does not allow things like

```
<!ENTITY stag "<text>">
<!ENTITY etag "</text>">
<!ENTITY statement "&stag;DO NOT DO THIS IN XML!&etag;">
. . . &statement; . . .
```

in XML though they are allowed in SGML.

3.1.4 In XML, an external entity may not be referenced in an attribute value specification. This prohibits the referencing of an entity such as

```
<!ENTITY value1 SYSTEM "C:\vals\val1.txt">
```

in an attribute value specification such as

```
<tag worth="&value1;">
```

3.1.5 In XML, a parameter entity reference in the internal (DOCTYPE) subset can provide only entire declarations. This means that the replacement text of a parameter entity declared and referenced in the internal (DOCTYPE) subset must consist of zero or more complete declarations. This prohibits referencing parameter entities such as

```
<!ENTITY % attdef "nosuch CDATA #IMPLIED">
```

in the internal (DOCTYPE) subset as follows

Guidelines for Converting an SGML DTD to an XML DTD

```
<!ATTLIST nosuch %attdef;>
```

However, this restriction does not apply to declarations in the external subset (the invoked DTD) where the above parameter entity reference would be legitimate.

3.2 The following XML restrictions apply to character references.

3.2.1 In XML, character references are always terminated with ";". SGML allows the ";" in a character reference to be omitted if the entity name is followed by anything other than a name character.

3.2.2 In XML, the standard named character references of SGML ("&#TAB;", "&#SPACE;", "&#RE;", and "&#RS;") are not defined. Also user-defined named character references are not allowed.

3.2.3 In XML, numeric character references to non-SGML characters are not allowed.

3.2.4 In XML, hexadecimal character numbers are allowed in character references, but they must be prefixed by "x", i.e., "ä".

3.3 The following restrictions apply to XML and SGML entity declarations.

3.3.1 In XML, #DEFAULT entities are not defined. In SGML, the replacement text of an SGML #DEFAULT entity will be substituted for entity reference of an undeclared general entity. For example, the #DEFAULT entity

```
<!ENTITY #DEFAULT "UNDEFINED ENTITY!">
```

will provide the "UNDEFINED ENTITY!" text whenever an undeclared entity is referenced in an instance. Any #DEFAULT entities in CALS DTDs will have to be removed when they are rewritten in XML.

3.3.2 In XML, external SDATA entities are not defined. In SGML, an external SDATA entity is an entity: (1) whose replacement text is externally stored, (2) whose replacement text will not be parsed but passed to the application software for processing in some special way, and (3) whose ENTITY declaration contains the keyword "SDATA" for specific character data. These could be transformed into external NDATA entities with appropriate data content notation specifications.

3.3.3 In XML, external CDATA entities are not defined. In SGML, an external CDATA entity is an entity: (1) whose replacement text is externally stored, (2) whose replacement text will be considered character data for parsing and processing by the application software, and (3) whose ENTITY declaration contains the keyword "CDATA" for character data. These could be transformed into external NDATA entities with appropriate data content notation specifications.

3.3.4 In XML, internal SDATA entities are not defined. In SGML, an internal SDATA entity is an entity: (1) whose replacement text is provided in the ENTITY declaration, (2) whose replacement text will not be parsed but passed to the application software for processing in some special way, and (3) whose ENTITY declaration contains the keyword "SDATA" for specific character data.

Guidelines for Converting an SGML DTD to an XML DTD

CALS DTDs use the ISO SDATA character entities to provide special characters in instances. They will have to be replaced with the corresponding entities whose replacement texts are the character references for the special characters or by the character references directly.

3.3.5 In XML, internal CDATA entities are not defined. In SGML, an internal CDATA entity is an entity: (1) whose replacement text is provided in the ENTITY declaration, (2) whose replacement text will be considered character data for parsing and processing by the application software, and (3) whose ENTITY declaration contains the keyword "CDATA" for character data.

3.3.6 In XML, PI entities are not defined. In SGML, PI entities are internal entities whose ENTITY declaration contains the keyword PI and whose replacement text is used to generate a processing instruction. For example, the PI entity

```
<!ENTITY tp PI "turn the page">
```

will generate the processing instruction

```
<? turn the page>
```

when the entity reference "&tp;" occurs in SGML instances.

3.3.7 In XML, bracketed text entities are not defined. In SGML, bracketed text entities are entities whose ENTITY declarations contain the keywords STARTTAG, ENDTAG, MS, or MD and whose references produce tags or marked sections in accordance with their parameter literals. For example, the SGML bracketed text entity "tempstart"

```
<!ENTITY tempstart STARTTAG "temp date='990402' version='1.2' ">
```

will produce the "temp" start-tag

```
<temp date='990402' version='1.2' >
```

when the entity reference "&tempstart;" occurs in instances.

3.3.8 In XML, external identifiers must always contain a system literal. In particular this means that a system literal, i.e., a path name must always follow a PUBLIC identifier.

3.3.9 In XML, attributes cannot be specified for an entity. This really refers to specifying attributes for a data content notation specified on an SGML external NDATA ENTITY declaration such as

```
<!ENTITY fig1 SYSTEM "C:\figs\fig1.cgm" NDATA cgm [type="special"]>
```

Since attributes can not be defined for an entity's data content notation in XML, no values can be specified on an XML ENTITY declaration.

3.3.10 Well-formedness is required for the replacement texts of parsed entities. The distinction between parsed and unparsed XML entities is the following. A parsed entity is an entity whose contents (replacement text) becomes an integral part of the document when the entity is referenced.

An unparsed entity is one whose contents may or may not be XML markup, and that has an associated data content notation specified for the processing of its

Guidelines for Converting an SGML DTD to an XML DTD

content. Unparsed entities can not be referenced. The only way that they can be cited is to provide their names as values of ENTITY or ENTITIES attribute values. XML does not restrict the contents of unparsed entities.

The following well-formed constraints on parsed entities should not cause any significant problems in converting CALS compliant DTDs and instances to XML.

3.3.10.1 Parsed entities can not reference themselves directly or indirectly.

3.3.10.2 An internal general parsed entity is well-formed if its replacement text consists of zero or more element markups, character data, general entity references, character references, CD (character data) sections, processing instructions, comments, or a combination thereof. It must also satisfy all the applicable well-formedness constraints (WFCs) of the XML specification.

3.3.10.3 An external parsed general entity is well-formed if its replacement text consists of an optional XML text declaration followed by zero or more element markups, character data, general entity references, character references, CD (character data) sections, processing instructions, comments, or a combination thereof. It must also satisfy all the applicable well-formedness constraints (WFCs) of the XML specification. Moreover, though it is not required, external parsed entities should begin with a text declaration. See clause 4.3.1 "The Text Declaration".

3.3.10.4 Parameter entities can be referenced only in XML DTDs.

3.3.10.5 An internal parameter entity is "well-formed by definition" (clause 4.3.2 "Well-formed Parsed Entities" of the XML specification). [72] "EntityDecl", [74] "PEDef", and [9] "EntityValue" in the XML specification define the content of an internal parameter entity to consist of zero or more character general entity references, character references, parameter entity references, character data not containing the "%" or "&" symbols, or a combination thereof.

3.3.10.6 An external parameter entity is well-formed if its replacement text consists of an optional XML text declaration followed by zero or more markup declarations, conditional sections, parameter entity reference, or a combination thereof. It must also satisfy all the well-formedness constraints (WFCs) of the XML specification.

3.3.10.7 An ampersand in a parameter literal of a parsed entity must be part of a syntactically valid entity reference or numeric character reference. The "&" character can occur in a parameter literal (the delimited entity definition in an ENTITY declaration) only as part of a general entity reference or character reference. It can not occur by itself as data in the parameter literal. If the "&" character is required as data in the replacement text of a general entity, then it must be escaped by "&#" or a character reference.

3.3.10.8 The same also holds true for the "%" character in the parameter literal of a parsed entity. It must be part of a parameter entity reference. If the "%" character is required as data in the replacement text of a general entity, then a general entity reference or a character reference must be used to escape it.

3.3.11 In XML, the five "lt" ("<"), "gt" (">"), "amp" ("&"), "quot" (''), and "apos" ('') character entities are predefined in XML and may be referenced

Guidelines for Converting an SGML DTD to an XML DTD

without declaring them in a DTD or DOCTYPE (inner) declaration subset. This is not the case in SGML.

3.4 The following restrictions apply to XML and SGML attribute definition list ("ATTLIST") declarations.

3.4.1 In XML, attribute list declarations for name groups are not allowed. For example, SGML ATTLIST declarations such as

```
<!ATTLIST (up | down) dir CDATA #IMPLIED>
```

must be rewritten as separate declarations

```
<!ATTLIST up dir CDATA #IMPLIED>
```

```
<!ATTLIST down dir CDATA #IMPLIED>
```

3.4.2 In XML, attributes cannot be defined for a data content notation. SGML allows attributes to be defined for data content notations. For example

```
<!ATTLIST #NOTATION cgm type (regular | special) #IMPLIED>
```

It then follows that in SGML values may be specified for these attributes when such a notation is specified on an NDATA ENTITY declaration. For example

```
<!ENTITY fig1 SYSTEM "C:\figs\fig1.cgm" NDATA cgm [type="special"]>
```

3.4.3 In XML, current attributes are not defined. SGML current ("#CURRENT") attributes can "remember" an attribute value for an attribute common to several element types.

3.4.4 In XML, content reference attributes are not defined. In SGML, a content reference attribute is an attribute with the default value of "#CONREF" specified for a non-EMPTY element type. If this attribute is specified on the element type's start-tag; then the element type is considered to have been declared EMPTY, and its markup consists solely of its start-tag. When this attribute is not specified on the element type's start-tag, the element content must be in accordance with the element type's content model or declared value.

The following is suggested as an XML alternative for an element type with a #CONREF attribute. Make the element type's content model optional by placing a "?" or "*" as appropriate after it and replace the #CONREF attribute with an attribute similar to

```
empty_content (yes) #IMPLIED
```

3.4.5 In XML, the SGML NUTOKEN or NUTOKENS declared values for attributes are not defined. Change them to NMTOKEN or NMTOKENS respectively or CDATA. However, since NMTOKEN and NMTOKENS are more restrictive than CDATA, they will tend to enforce the intent of the original NUTOKEN or NUTOKENS attribute values.

3.4.6 In XML, the SGML NUMBER or NUMBERS declared values for attributes are not defined. Change them to NMTOKEN or NMTOKENS respectively or CDATA. However, since NMTOKEN and NMTOKENS are more restrictive than CDATA, they will tend to enforce the intent of the original NUMBER or NUMBERS attribute values.

Guidelines for Converting an SGML DTD to an XML DTD

3.4.7 In XML, the SGML NAME or NAMES declared values for attributes are not defined. Change them to NMTOKEN or NMTOKENS respectively or CDATA. However, since NMTOKEN and NMTOKENS are more restrictive than CDATA, they will tend to enforce the intent of the original NAME or NAMES attribute values.

3.4.8 In XML, the "or" connector "|" must be used in XML name token groups. SGML allows any or all of the connectors ("|", ",", or "&") to be used in separating the name tokens in a name token group defining attribute value options. For example,

```
<!ATTLIST nosuch dir (up & down , right | left) #IMPLIED>
```

is legitimate in SGML, but only

```
<!ATTLIST nosuch dir (up | down | right | left) #IMPLIED>
```

is acceptable in XML.

3.4.9 In XML, an attribute's default value consisting solely of name characters must always be quoted. In SGML, when the default value in an attribute definition in an ATTLIST declaration consists solely of name characters, the delimiters may be omitted. Otherwise it must be delimited. For example, the default value "eoss.dtd" in

```
fileloc CDATA eoss.dtd
```

does not have to be delimited since "." is an SGML name character, but the default value "C:\eoss\eoss.dtd" in

```
fileloc CDATA "C:\eoss\eoss.dtd"
```

must be delimited because ":" and "\" are not CALS SGML name characters.

3.4.10 In XML, the definition of an element type may have more than one attribute definition list, i.e., ATTLIST declaration. For example

```
<!ATTLIST nosuch type CDATA #IMPLIED>
```

```
<!ATTLIST nosuch version NMTOKEN #REQUIRED>
```

is acceptable in XML, but not in SGML.

3.4.11 In XML, an attribute definition may have repeated attribute tokens, i.e., a name token can occur more than once in the name token groups defining the ranges of attribute values. For example

```
<!ATTLIST nosuch available (yes | no) #IMPLIED  
backordered (yes | no) #IMPLIED>
```

is acceptable in XML, but not in SGML.

3.4.12 In XML, an attribute definition list can be empty, i.e., the ATTLIST declaration defines no attributes for an element type. For example

```
<!ATTLIST nosuch>
```

is acceptable in XML, but not in SGML.

Guidelines for Converting an SGML DTD to an XML DTD

3.5 The following XML restrictions pertain to element type (ELEMENT) declarations.

3.5.1 In XML, element type declarations for name groups are not allowed. For example, SGML ELEMENT declarations such as

```
<!ELEMENT (up | down) - - (#PCDATA)>
```

must be rewritten as separate ELEMENT declarations

```
<!ELEMENT up (#PCDATA)>
```

```
<!ELEMENT down (#PCDATA)>
```

Remember that XML does not have tag minimization parameters.

3.5.2 In XML, "ranked" element types are not allowed. In SGML, "ranked" element types are allowed even when the RANK feature is "NO". The use of "ranked" element types is a largely unsupported aspect by SGML application software.

3.5.3 In XML, omitted tag minimization parameters in element type declarations are not allowed. This means that all SGML omitted tag minimization parameters of "- O", "- o", "- -", "O O", or "o o" will have to be removed from the ELEMENT declarations.

3.5.4 In XML, RCDATA declared content is not allowed. In SGML, RCDATA (replaceable character data) content consists of character data in which general entity references and character references are recognized and resolved. SGML RCDATA content is terminated by the first "</" delimiter encountered. SGML RCDATA declared content will have to be replaced with the "(#PCDATA)" content model.

3.5.5 In XML, CDATA declared content is not allowed. In SGML, CDATA (character data) content consists only of character data which will be terminated by the first "</" delimiter encountered. SGML CDATA declared content will have to be replaced with the "(#PCDATA)" content model.

3.5.6 In XML, the "and" connector "&" can not be used in content models. The SGML "(a & b)" model group is equivalent to the SGML and XML "((a, b)|(b, a))" content model. Removing "&" from some complex SGML content models could be a significant problem.

3.5.7 In XML, all mixed content models (i.e., those containing the "#PCDATA" content token) must be of the form

```
(#PCDATA | a | b | c | . . . )*
```

This could involve some work in converting complex SGML content models in CALS DTDs, particularly those defined in terms of parameter entity references.

3.5.8 In XML, inclusions are not allowed in content models. The SGML included element types

```
+(a | b | c . . . )
```

Guidelines for Converting an SGML DTD to an XML DTD

will have to be made options of the XML content model. While removing the inclusion from an SGML content model is trivial, the content model's model group will have to be rewritten to allow the previously included element types as model group options so as to preserve the intent of the SGML content model. This may require significant analysis of the element type's intended use.

3.5.9 In XML, exclusions are not allowed in content models. If the SGML excluded element types

```
-(a | b | c . . . )
```

can not be eliminated as options of the XML content model, tagging conventions will have to be used to disallow the undesired markup.

3.6 The following XML restrictions apply to comments.

3.6.1 The "embedded" comments of SGML are not allowed in XML declarations. An XML comment is a declaration of the form

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

where the text can not contain "--". An SGML comment merely consists of

```
-- text --
```

where the text does not contain "--". Such SGML comments can be "embedded" in declarations wherever the syntax permits. The "-- 4 digits --" in

```
<!ATTLIST date year CDATA -- 4 digits -- #REQUIRED>
```

is an example of an "embedded" SGML comment. All such embedded SGML comments must be removed or converted to XML comments.

```
<!-- 4 digits -->  
<!ATTLIST date year CDATA #REQUIRED>
```

3.6.2 The "empty" comment declaration of SGML

```
<!>
```

is not defined in XML.

3.6.3 An XML comment declaration cannot contain more than one SGML comment. SGML allows comment declarations containing more than one comment such as

```
<!-- text1 -- -- text2 -- -- text3 -->
```

All XML comment declarations must be of the form

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

where the text contains no intervening "--". The text of the comment may extend over several lines.

3.6.4 An XML comment declaration must be terminated with "-->". SGML allows comments of the form

Guidelines for Converting an SGML DTD to an XML DTD

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

and similar forms where the terminating ">" can be separated from the last "--" by spaces, carriage returns, TABs, or a combination thereof.

3.7 The following XML restrictions pertain to processing instructions.

3.7.1 In XML, all processing instructions (PIs) must start with the name of the PI target.

3.7.2 A PI target must not be one of the eight strings "XML", "xml", "XmL", "Xml", "xMl", "xML", "XMl", "xmL" since they are "reserved for standardization". See clause 2.6 "Processing Instructions" of the XML specification.

3.8 The following XML restrictions pertain to marked sections.

3.8.1 In XML, marked section declarations can not use the TEMP status keyword. In SGML, the

```
<![TEMP[ . . . ]]>
```

marked section merely identifies the material in its declaration subset as temporary material that may be removed later.

3.8.2 In XML, marked section declarations can not use the RCDATA status keyword. In SGML, the

```
<![RCDATA[ . . . ]]>
```

marked section causes the material in its declaration subset to be treated as replaceable character data (character data in which general entity references and character references are recognized and resolved) for parsing purposes.

3.8.3 In XML, INCLUDE or IGNORE marked sections are not allowed in the document instance. In SGML and XML, the

```
<![INCLUDE[ . . . ]]>
```

marked section causes the material in its declaration subset to be considered as being present for parsing purposes. In SGML and XML, the

```
<![IGNORE[ . . . ]]>
```

marked section causes the material in its declaration subset to be considered as not being present for parsing purposes. These two constructs are not allowed in XML document instances though they are allowed in SGML document instances.

3.8.4 In XML, a status keyword specification must always be provided in an XML marked section declaration. In SGML, the

```
<![ [ . . . ] ]>
```

marked section is considered to be the same as

```
<![INCLUDE[ . . . ]]>
```

Guidelines for Converting an SGML DTD to an XML DTD

3.8.5 In XML, a status keyword specification must not contain more than one status keyword in an XML marked section declaration. In SGML, marked sections can have multiple status keywords such as in

```
<![INCLUDE RCDATA TEMP[ . . . ]]>
```

The material in such a declaration subset is treated with respect to the "IGNORE "CDATA RCDATA INCLUDE" priority assigned to interpretation of multiple status keywords in SGML.

3.8.6 In XML, marked sections with the status keywords of INCLUDE, IGNORE, and CDATA are not allowed in the internal (DOCTYPE) subset. This is also true in SGML.

3.8.7 XML marked sections are allowed only in the external subset. Parameter entity references may be used to supply the keyword.

3.9 The following miscellaneous XML restrictions apply.

3.9.1 Though it is not required, the XML Declaration

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

should be placed before the DOCTYPE declaration. Encoding and "standalone" information may also be specified in the form of attribute values on the XML Declaration. See clause 4.3.3 "Encoding Information" and clause 2.9 "Standalone Document Declaration" of the XML specification.

3.9.2 The XML specification defines only one concrete syntax and no optional features for XML. Since there is one SGML Declaration for XML, there is no need for the SGML Declaration for XML to be present when parsing in an XML environment. However, it would be required to define XML when parsing in an SGML environment.

3.9.3 In XML element content, the "<" character can not occur other than as a start-tag open delimiter, i.e., part of a start-tag. If "<" is required as a data character, it must be provided by a general entity reference or character reference. For example, if "3 < 5" occurs in text, the "<" must be escaped by "<" or a character reference. For consistency, ">" should also be used to provide the ">" character as data though this is not required.

3.9.4 In XML element content, the "&" character can not occur other than as part of a general entity reference or a character reference. If "&" is required as a data character, it must be provided by a general entity reference or character reference. For example, if "R&D" occurs in text, the "&" must be escaped by "&" or a character reference.

3.9.5 In XML, a parameter separator required by the formal syntax must always be present and cannot be omitted when it is adjacent to a delimiter. In SGML, such parameter separators may be omitted when the parser would not be confused by their absence. Consider the following examples.

For example, omitting the space between "--" and "(" in the following SGML declaration

```
<!ELEMENT nosuch - -(#PCDATA)>
```

Guidelines for Converting an SGML DTD to an XML DTD

is acceptable. However, omitting the space between "nosuch" and "--" in

```
<!ELEMENT nosuch- - (#PCDATA)>
```

is not acceptable in SGML as the first dash would be considered part of the element type name (since "-" is an SGML name character). As a result the SGML omitted tag minimization parameter would be incomplete. In this case omitting the parameter separator would result in an SGM syntax error.

However, in XML all required parameter separators must be present. For example

```
<!ELEMENT nosuch(#PCDATA)>
```

is a syntax error since the element type name and its content specification must be separated. On the other hand

```
<!ELEMENT nosuch (#PCDATA)>
```

is not a syntax error since the separation of the content specification and the terminating ">" is optional. See [45] "elementdecl" in clause 3.2 "Element Type Declarations" in the XML specification.