Extensible Markup Language Version 1.0 Part 1: Syntax

W3C Working Draft 30-Jun-97

This version

`http://www.w3.org/pub/WWW/TR/WD-xml-lang-970630.html`

Previous versions

`http://www.w3.org/pub/WWW/TR/WD-xml-961114.html`
`http://www.w3.org/pub/WWW/TR/WD-xml-lang-970331.html`

Editors

Tim Bray, Textuality *(tbray@textuality.com)*
C. M. Sperberg-McQueen, University of Illinois at Chicago *(cmsmcq@uic.edu)*

Status of this document

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current W3C working drafts can be found at http://www.w3.org/pub/WWW/TR.

Note: Since working drafts are subject to frequent change, you are advised to reference the above URL, rather than the URLs for working drafts themselves.

This work is part of the W3C SGML Activity (for current status, see http://www.w3.org/pub/WWW/MarkUp/SGML/Activity).

Abstract

Extensible Markup Language (XML) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Table of Contents

Appendices

1. Introduction

The Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879].

XML documents are made up of storage units called entities, which contain either text or binary data. Text is made up of characters, some of which form the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, referred to as the application. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an SGML Editorial Review Board formed under the auspices of the World Wide Web Consortium (W3C) in 1996 and chaired by Jon Bosak of Sun Microsystems, with the very active participation of an SGML Working Group also organized by the W3C. The membership of the Editorial Review Board is given in an appendix. Dan Connolly served as the ERB's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with the associated standards, provides all the information necessary to understand XML version 1.0 and construct computer programs to process it.

This version of the XML specification (30 June 1997) is for public review and discussion. It may be distributed freely, as long as all text and legal notices remain intact.

1.2 Relationship to Existing Standards

Standards relevant to users and implementors of XML include:

- SGML (ISO 8879:1986). By definition, valid XML documents are conformant SGML documents in the sense described in ISO standard 8879. The current draft of this specification presupposes the successful completion of the current work on a technical corrigendum to ISO 8879 now being prepared by ISO/IEC JTC1/SC18/WG8. If the corrigendum is not adopted in the expected form, some clauses of this specification may change, and some recommendations now labeled "for interoperability" will become requirements labeled "for compatibility".

- Unicode and ISO/IEC 10646. This specification depends on the international standard ISO/IEC 10646 (with amendments AM 1 through AM 5) and the *Unicode Standard, Version 2.0*, which define the encodings and meanings of the characters which make up XML text data. All the characters in ISO/IEC 10646 are present, at the same code points, in Unicode.
- IETF RFC 1738 and RFC 1808. RFC 1738 and RFC 1808 define the syntax and semantics of Uniform Resource Locators, or URLs.

1.3 Terminology

Some terms used with special meaning in this specification are:

may

Conforming data and XML processors are permitted to but need not behave as described.

must

Conforming data and XML processors are required to behave as described; otherwise they are in error.

error

A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

fatal error

An error which conforming software must detect and report to the application. After encountering a fatal error, an XML processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed text from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e. it must not continue to pass character data and information about the document's logical structure to the application in the normal way).

validity constraint

A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.

well-formedness constraint

A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.

at user option

Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.

match

(Of strings or names:) Case-insensitive match: two strings or names being compared match if they are identical after case-folding. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) The content of a parent element in a document matches the content model for that element if (a) the content model matches the rule for `Mixed` and the content consists of character data and elements whose names match names in the content model, or if (b) the content model matches the rule for `elements`, and the sequence of child elements belongs to the language generated by the regular expression in the content model.

case-folding

a process applied to a sequence of characters, in which those identified as non-uppercase (in scripts which have case distinctions) are replaced by their uppercase equivalents, as specified in The

Unicode Standard, Version 2.0, section 4.1. Note that Unicode recommends folding to lowercase; for compatibility reasons, XML processors must fold to uppercase. Case-folding, as described here, neither requires nor forbids the normalization of Unicode character sequences into canonical form (e.g. as described in The Unicode Standard, section 5.9).

exact(ly) match

Case-sensitive string match: two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. At user option, processors may normalize such characters to their canonical form.

for compatibility

A feature of XML included solely to ensure that XML remains compatible with SGML.

for interoperability

A non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the technical corrigendum to ISO 8879 now in the process of preparation by ISO/IEC JTC1/SC18/WG8.

1.4 Notation

The formal grammar of XML is given using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, or with an initial lowercase letter if a recursive grammar is required for recognition. Literal strings are quoted; unless otherwise noted they are case-insensitive. The distinction between symbols which can and cannot be recognized using simple regular expressions may be used to set the boundary between an implementation's lexical scanner and its parser, but this specification neither constrains the placement of that boundary nor presupposes that all implementations will have one.

Within the expression on the right-hand side of a rule, the meaning of symbols is as shown below.

`#xN`

where N is a hexadecimal integer, the expression represents the character in ISO/IEC 10646 whose canonical (UCS-4) bit string, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeroes in the `#xN` form is insignificant; the number of leading zeroes in the corresponding bit string is governed by the character encoding in use and is not significant for XML.

`[a-zA-Z]`, `[#xN-#xN]`

represents any character with a value in the range(s) indicated (inclusive).

`[^a-z]`, `[^#xN-#xN]`

represents any character with a value *outside* the range indicated.

`[^abc]`, `[^#xN#xN#xN]`

represents any character with a value not among the characters given.

`"string"`

represents a literal string matching that given inside the double quotes.

`'string'`

represents a literal string matching that given inside the single quotes.

`a b`

a followed by b.

`a | b`

a or b but not both.

`a - b`

the set of strings represented by a but not represented by b

`a?`

a or nothing; optional a.

`a+`

one or more occurrences of a.

`a*`

zero or more occurrences of a.

`%a`

specifies that *in the external DTD subset* a parameter entity may occur in the text at the position where a may occur; if so, its replacement text must match `S? a S?`. If the expression a is governed by a suffix operator, then the suffix operator determines both the maximum number of parameter-entity references allowed and the number of occurrences of a in the replacement text of the parameter entities: `%a*` means that a must occur zero or more times, and that some of its occurrences may be replaced by references to parameter entities whose replacement text must contain zero or more occurrences of a; it is thus a more compact way of writing `%(a*)*`. Similarly, `%a+` means that a must occur one or more times, and may be replaced by parameter entities with replacement text matching `S? (a S?)+`. The recognition of parameter entities in the internal subset is much more highly constrained.

(expression)

expression is treated as a unit, and may carry the `%` prefix operator, or a suffix operator: `?`, `*`, or `+`.

`/* ... */`

comment.

`[ WFC: ... ]`

Well-formedness check; this identifies by name a check for well-formedness associated with a production.

`[ VC: ... ]`

Validity check; this identifies by name a check for validity associated with a production.

1.5 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White space

[1]	S	::=	`(#x20 | #x9 | #xd | #xa)+`

Legal characters are tab, carriage return, line feed, and the legal graphic characters of Unicode and ISO/IEC 10646.

Character range

```
[2]  Char            ::= #x9 | #xA | #xD | [#x20-#xFFFD]        /* any ISO/IEC 10646
                         | [#x00010000-#x7FFFFFFF]              UCS-4 code, FFFE and
                                                                FFFF excluded */
```

Characters are classified for convenience as letters, digits, or other characters. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character. Certain layout and format-control characters defined by ISO/IEC 10646 should be ignored when recognizing identifiers; these are defined by the classes Ignorable and Extender. Full definitions of the specific characters in each class are given in the appendix on character classes.

A Name is a token beginning with a letter or underscore character and continuing with letters, digits, hyphens, underscores, or full stops (together known as name characters). Names beginning with the string "XML" are reserved for standardization in this or future versions of this specification.

Note: the colon character is also allowed within XML names; it is reserved for experimentation with name spaces and schema scoping. Its meaning is expected to be standardized at some future point, at which point those documents using colon for experimental purposes will need to be updated. (Note: there is no guarantee that any name-space mechanism adopted for XML will in fact use colon as a name-space delimiter.) In practice, this means that authors should not use colon in XML names except as part of name-space experiments, but that implementors should accept colon as a name character.

An Nmtoken (name token) is any mixture of name characters.

Names and tokens

```
[3]  MiscName        ::= '.' | '-' | '_' | ':' | CombiningChar | Ignorable
                         | Extender
[4]  NameChar        ::= Letter | Digit | MiscName
[5]  Name            ::= (Letter | '_' | ':') (NameChar)*
[6]  Names           ::= Name (S Name)*
[7]  Nmtoken         ::= (NameChar)+
[8]  Nmtokens        ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string; different forms of literal data may or may not contain angle brackets, entity references, and character references. Literals are used for specifying the replacement text of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral); for some purposes, the entire literal can be skipped without scanning for markup within it (SkipLit):

Literals

```
[9]  EntityValue     ::= '"' ([^%&"] | PEReference | Reference)* '"'
                         |  "'" ([^%&'] | PEReference | Reference)* "'"
[10] AttValue        ::= '"' ([^<&"] | Reference)* '"'
                         |  "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral   ::= '"' URLchar* '"' | "'" (URLchar - "'")* "'"
[12] URLchar         ::= /* See RFC 1738 and 1808 */
[13] PubidLiteral    ::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[14] PubidChar       ::= #x20 | #x9 | #xd | #xa | #x3000 | [a-zA-Z0-9] | [-
                         '()+,./:=?]
[15] SkipLit         ::= ('"' [^"]* '"') | ("'" [^']* "'")
```

Note that entity references and character references are recognized and processed within EntityValue and AttValue, but not within SystemLiteral.

2. Documents

A textual object is an XML document if it is either valid or well-formed, as defined in this specification.

2.1 Logical and Physical Structure

Each XML document has both a logical and a physical structure.

Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity.

The logical structure contains declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup.

The two structures must be synchronous: see section 4.1.

2.2 Well-Formed XML Documents

A textual object is said to be a well-formed XML document if, first, it matches the production labeled document, and if for each entity reference which appears in the document, either the entity has been declared in the document type declaration or the entity name is one of: amp, lt, gt, apos, quot.

Matching the document production implies that:

1. It contains one or more elements.
2. It meets all the well-formedness constraints (WFCs) given in the grammar.
3. There is exactly one element, called the root, or document element, for which neither the start-tag nor the end-tag is in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest within each other.

As a consequence of this, for each non-root element C in the document, there is one other element P in the document such that C is in the content of P, but is not in the content of any other element that is in the content of P. Then P is referred to as the parent of C, and C as a child of P.

2.3 Characters

The data stored in an XML entity is either text or binary. Binary data has an associated notation, identified by name; beyond a requirement to make available the notation's name and the associated system identifier, XML places no constraints on the contents or use of binary entities. So-called binary data might in fact be textual; its identification as binary means that an XML processor need not parse it in the fashion described by this specification. XML text data is a sequence of characters. A character is an atomic unit of text; valid characters are specified by ISO/IEC 10646. Users may extend the ISO/IEC 10646 character repertoire by exploiting the private use areas.

The mechanism for encoding character values into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UCS-2 encodings of 10646; the mechanisms for signaling which of the two are in use, or for bringing other encodings into play, are discussed later, in the discussion of character encodings.

Regardless of the specific encoding used, any character in the ISO/IEC 10646 character set may be referred to by the decimal or hexadecimal equivalent of its bit string.

2.4 Character Data and Markup

XML text consists of intermingled character data and markup. Markup takes the form of start-tags, end-tags, empty elements, entity references, character references, comments, CDATA sections, document type declarations, and processing instructions.

All text that is not markup constitutes the character data of the document.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within comments, processing instructions, or CDATA sections. If they are needed elsewhere, they must be escaped using either numeric character references or the strings "&" and "<". The right angle bracket (>) may be represented using the string ">", and must, for compatibility, be so represented when it appears in the string "]]>", when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "]]>".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """.

Character data

```
[16] PCData        ::= [^<&]*
```

2.5 Comments

Comments may appear anywhere except in a CDATA section, i.e. within element content, in mixed content, or in a DTD. They must not occur within declarations or tags. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "--" (double-hyphen) must not occur within comments.

Comments

```
[17] Comment       ::= '<!--' (Char* - (Char* '--' Char*)) '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

2.6 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications.

Processing instructions

```
[18] PI            ::= '<?' Name S (Char* - (Char* '?>' Char*)) '?>'
```

PIs are not part of the document's character data, but must be passed through to the application. The Name is called the PI target; it is used to identify the application to which the instruction is directed. XML provides an optional mechanism, NOTATION, for formal declaration of such names. PI targets with names beginning with the string "XML" are reserved for standardization in this or future versions of this specification.

2.7 CDATA Sections

CDATA sections can occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string `<![CDATA[` and end with the string `]]>`:

CDATA sections

[19]	**CDSect**	::=	*CDStart CData CDEnd*
[20]	**CDStart**	::=	`'<![CDATA['`
[21]	**CData**	::=	`(Char* - (Char* ']]>' Char*))`
[22]	**CDEnd**	::=	`']]>'`

Within a CDATA section, only the CDEnd string is recognized, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using `<` and `&`. CDATA sections cannot nest.

An example of a CDATA section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines, denoted by the nonterminal `S` in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that must be retained in the delivered version is common, for example in poetry and source code.

An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must distinguish white space in element content from other non-markup characters and signal to the application that white space in element content is not significant.

A special attribute may be inserted in documents to signal an intention that the element to which this attribute applies requires all white space to be treated as significant by applications.

In valid documents, this attribute must be declared as follows, if used:

```
XML-SPACE (DEFAULT|PRESERVE) #IMPLIED
```

The value `DEFAULT` signals that applications' default white-space processing modes are acceptable for this element; the value `PRESERVE` indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of this element, unless overriden with another instance of the XML-SPACE attribute.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.9 Prolog and Document Type Declaration

XML documents may, and should, begin with an XML declaration which specifies, among other things, the version of XML being used.

The function of the markup in an XML document is to describe its storage and logical structures, and associate attribute-value pairs with the logical structure. XML provides a mechanism, the document type declaration, to define constraints on that logical structure and to support the use of predefined storage units. An XML document is said to be valid if there is an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first start-tag in the document.

XML document

[23]	**document**	::=	*prolog element Misc**
[24]	**prolog**	::=	*XMLDecl*? *Misc** (*doctypedecl Misc**)?
[25]	**XMLDecl**	::=	'<?XML' *VersionInfo EncodingDecl*? *RMDecl*? *S*? '?>'
[26]	**VersionInfo**	::=	*S* 'version' *Eq* ('"1.0"' | "'1.0'")
[27]	**Misc**	::=	*Comment* | *PI* | *S*

The identification of the XML version as "1.0" does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary.

For example, the following is a complete XML document, well-formed but not valid:

```
<?XML version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The XML document type declaration may include a pointer to an external entity containing a subset of the necessary markup declarations, and may also directly include another, internal, subset.

These two subsets make up the document type definition, abbreviated DTD. The DTD, in effect, provides a grammar which defines a class of documents. Properly speaking, the DTD consists of both subsets taken together, but it is a common practice for the bulk of the markup declarations to appear in the external subset, and for this subset, usually contained in a file, to be referred to as "the DTD" for a class of documents.

Document type definition

[28]	**doctypedecl**	::=	'<!DOCTYPE' *S Name* (*S ExternalID*)? *S*? ('[' %*markupdecl** ']' *S*?)? '>'	[VC: Root Element Type] [VC: Non-null DTD]
[29]	**markupdecl**	::=	(%*elementdecl* | %*AttlistDecl* | %*EntityDecl* | %*NotationDecl* | %*PI* | %*S* | %*Comment* | *InternalPERef*)*	
[30]	**InternalPERef**	::=	*PEReference*	[WFC: Integral Declarations]

VALIDITY CHECK: Root Element Type. The Name in the document-type declaration must match the element type of the root element.

VALIDITY CHECK: Non-null DTD. The internal and external subsets of the DTD must not both be empty.

WELL-FORMEDNESS CHECK: Integral Declarations. A parameter-entity reference recognized in this context must have replacement text consisting of zero or more complete declarations, i.e. matching the production for the non-terminal markupdecl.

The external subset must obey substantially the same grammatical constraints as the internal subset; i.e. it must match the production for the non-terminal symbol markupdecl. In the external subset, however, parameter-entity references can be used to replace constructs prefixed by `%` in a production of the grammar, and conditional sections may occur. In the internal subset, by contrast, conditional sections may not occur and the only parameter-entity references allowed are those which match the non-terminal InternalPERef within the rule for markupdecl.

External subset

[31]	**extSubset**	::=	(%*markupdecl* | %*conditionalSect*)*

For example:

```
<?XML version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier `hello.dtd` indicates the location of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?XML version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, an XML processor must read the internal subset first, then the external subset. This has the effect that entity and attribute declarations in the internal subset take precedence over those in the external subset.

2.10 Required Markup Declaration

In some cases, an XML processor can read an XML document and accomplish useful tasks without having first processed the entire DTD. However, certain declarations can substantially affect the actions of an XML processor. It is desirable, therefore, to be able to specify whether a document contains any such declarations. A document author can communicate whether or not DTD processing is necessary using a required markup declaration (abbreviated RMD), which appears as a component of the XML declaration:

Required markup declaration

```
[32] RMDecl    ::= S 'RMD' Eq "'" ('NONE' | 'INTERNAL' | 'ALL') "'"
                 | S 'RMD' Eq '"' ('NONE' | 'INTERNAL' | 'ALL') '"'
```

In an RMD, the value NONE indicates that an XML processor can parse the containing document correctly without first reading any part of the DTD. The value INTERNAL indicates that the XML processor must read and process the internal subset of the DTD, if provided, to parse the containing document correctly. The value ALL indicates that the XML processor must read and process the declarations in both the subsets of the DTD, if provided, to parse the containing document correctly.

The RMD must indicate that the entire DTD is required if the external subset contains any declarations of

- attributes with default values, if elements to which these attributes apply appear in the document without specifying values for these attributes, or
- entities (other than amp, lt, gt, apos, quot), if references to those entities appear in the document, or
- element types with element content, if white space occurs directly within any instance of those types.

If such declarations occur in the internal but not the external subset, the RMD must take the value INTERNAL. It is an error to specify INTERNAL if the external subset is required, or to specify NONE if the internal or external subset is required.

If no RMD is provided, an XML processor must behave as though an RMD had been provided with the value ALL.

An example XML declaration with an RMD:

```
<?XML version="1.0" RMD='INTERNAL'?>
```

3. Logical Structures

Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, are those of the start-tag. Each element has a type, identified by name (sometimes called its generic identifier or GI), and may have a set of attributes. Each attribute has a name and a value.

This specification does not constrain the semantics, use, or (beyond syntax) names of the elements and attributes, except that names beginning with the string "XML" are reserved for standardization in this or future versions of this specification.

3.1 Start- and End-Tags

The beginning of every non-empty XML element is marked by a start-tag.

Start-tag

```
[33] STag       ::= '<' Name (S Attribute)* S? '>'    [WFC: Unique Att Spec]
[34] Attribute  ::= Name Eq AttValue                  [VC: Attribute Value Type]
                                                      [WFC: No External Entity
                                                          References]
[35] Eq         ::= S? '=' S?
```

The Name in the start- and end-tags gives the element's type. The Name-AttValue pairs are referred to as the attribute specifications of the element, with the Name referred to as the attribute name and the content of the AttValue (the characters between the `'` or `"` delimiters) as the attribute value.

VALIDITY CHECK: Unique Att Spec. No attribute may appear more than once in the same start-tag.

VALIDITY CHECK: Attribute Value Type. The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see the discussion of attribute declarations.)

WELL-FORMEDNESS CHECK: No External Entity References. Attribute values cannot contain entity references to external entities.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

The end of every element which is not empty is marked by an end-tag containing a name that echoes the element's type as given in the start-tag:

End-tag

```
[36] ETag       ::= '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

The text between the start-tag and end-tag is called the element's content:

Content of elements

```
[37] content    ::= (element | PCData | Reference      [VC: Content]
                    | CDSect | PI | Comment)*
[38] element    ::= EmptyElement
                    | STag content ETag                [WFC: GI Match]
```

VALIDITY CHECK: Content. Each element type used must be declared. The content of an element instance must match the content model declared for that element type.

WELL-FORMEDNESS CHECK: GI Match. The Name in an element's end-tag must match that in the start-tag.

If an element is empty, the start-tag constitutes the whole element. An empty element takes a special form:

Tags for empty elements

<table>
<tr><td>[39]</td><td>EmptyElement</td><td>::=</td><td>'<' <i>Name</i> (<i>S Attribute</i>)* <i>S</i>? '/>'</td><td></td></tr>
</table>

An example of an empty element:

```
<IMG align="left"
 src="http://www.w3.org/pub/WWW/Icons/WWW/w3c_48x48.gif" />
```

3.2 Element Declarations

The element structure of an XML document may, for validation purposes, be constrained using element and attribute declarations.

An element declaration constrains the element's type and its content.

Element declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a reference is made to an element type for which no declaration is provided, but this is not an error.

An element declaration takes the form:

Element declaration

<table>
<tr><td>[40]</td><td>elementdecl</td><td>::=</td><td>'<!ELEMENT' <i>S</i> %<i>Name S</i> (%<i>S S</i>)? %<i>contentspec S</i>? '>'</td><td>[VC: Unique Element Declaration]</td></tr>
<tr><td>[41]</td><td>contentspec</td><td>::=</td><td>'EMPTY' | 'ANY' | Mixed | elements</td><td></td></tr>
</table>

where the Name gives the type of the element.

VALIDITY CHECK: Unique Element Declaration. No element type may be declared more than once.

The content of an element can be categorized as element content or mixed content , as explained below. An element declared using the keyword EMPTY must be empty when it appears in the document.

If an element type is declared using the keyword ANY, then there are no validity constraints on its content: it may contain child elements of any type and number, interspersed with character data.

Examples of element declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

An element type may be declared to have element content, which means that elements of that type may only contain other elements (no character data). In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they appear. The grammar is built on content particles (CPs), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content models

```
[42] elements   ::= (choice | seq) ('?' | '*' | '+')?
[43] cp         ::= (Name | choice | seq) ('?' | '*' | '+')?
[44] cps        ::= S? %cp S?
[45] choice     ::= '(' S? %ctokplus (S? '|' S? %ctoks)* S? ')'
[46] ctokplus   ::= cps ('|' cps)+
[47] ctoks      ::= cps ('|' cps)*
[48] seq        ::= '(' S? %stoks (S? ',' S? %stoks)* S? ')'
[49] stoks      ::= cps (',' cps)*
```

where each Name gives the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the appropriate location; content particles occurring in a sequence list must each appear in the element content in the order given. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element name in the content model. For compatibility reasons, it is an error if an element in the document can match more than one occurrence of an element name in the content model. More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error. For more information, see the appendix on deterministic content models.

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT head (%head.content; | %head.misc;)*>
```

3.2.2 Mixed Content

An element type may be declared to contain mixed content, that is, text comprising character data optionally interspersed with child elements. In this case, the types of the child elements are constrained, but not their order nor their number of occurrences:

Mixed-content declaration

```
[50] Mixed      ::= '(' S? %( %'#PCDATA' (S? '|' S? %Mtoks)* ) S? ')*'
                    | '(' S? %('#PCDATA') S? ')'
[51] Mtoks      ::= %Name (S? '|' S? %Name)*
```

where the Names give the types of elements that may appear as children. The same name must not appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attributes may appear only within start-tags; thus, the productions used to recognize them appear in the discussion of start-tags. Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish a set of type constraints on these attributes.
- To provide default values for attributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

Attribute list declaration

```
[52] AttlistDecl     ::= '<!ATTLIST' S %Name S? %AttDef+ S? '>'
[53] AttDef          ::= S %Name S %AttType S %Default
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, and at most one attribute definition for a given attribute name. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

Attribute types

```
[54] AttType         ::= StringType | TokenizedType | EnumeratedType
[55] StringType      ::= 'CDATA'
[56] TokenizedType   ::= 'ID'                                [VC: ID]
                         | 'IDREF'                           [VC: Idref]
                         | 'IDREFS'                          [VC: Idref]
                         | 'ENTITY'                          [VC: Entity Name]
                         | 'ENTITIES'                        [VC: Entity Name]
                         | 'NMTOKEN'                         [VC: Name Token]
                         | 'NMTOKENS'                        [VC: Name Token]
```

VALIDITY CHECK: ID. Values of this type must be valid `Name` symbols. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.

VALIDITY CHECK: Idref. Values of this type must match the Name (for `IDREFS`, the Names) production; each Name must match the value of an ID attribute on some element in the XML document; i.e. `IDREF` values must match some ID.

VALIDITY CHECK: Entity Name. Values of this type must match the production for Name (for `ENTITIES`, Names); each Name must exactly match the name of an external binary general entity declared in the DTD.

> **VALIDITY CHECK: Name token.** Values of this type must consist of a string matching the Nmtoken nonterminal (for `NMTOKENS`, the Nmtokens nonterminal) of the grammar defined in this specification.

The XML processor must normalize attribute values before passing them to the application, as described in the section on attribute-value normalization.

Enumerated attributes can take one of a list of values provided in the declaration; there are two types:

Enumerated attribute types

[57]	**EnumeratedType**	::=	*NotationType* | *Enumeration*	
[58]	**NotationType**	::=	%'NOTATION' *S* '(' *S*? %*Ntoks* (*S*? '|' *S*? %*Ntoks*)* *S*? ')'	[VC: Notation Attributes]
[59]	**Ntoks**	::=	%*Name* (*S*? '|' *S*? %*Name*)*	
[60]	**Enumeration**	::=	'(' *S*? %*Etoks* (*S*? '|' *S*? %*Etoks*)* *S*? ')'	[VC: Enumeration]
[61]	**Etoks**	::=	%*Nmtoken* (*S*? '|' *S*? %*Nmtoken*)*	

> **VALIDITY CHECK: Notation Attributes.** The names in the declaration of `NOTATION` attributes must be names of declared notations (see the discussion of notations). Values of this type must match one of the notation names included in the declaration.

> **VALIDITY CHECK: Enumeration.** Values of this type must match one of the `Nmtoken` tokens in the declaration. For interoperability, the same `Nmtoken` should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document:

Attribute defaults

[62]	**Default**	::=	'#REQUIRED' | '#IMPLIED' | ((%'#FIXED' S)? %*AttValue*)	[VC: Attribute Default Legal]

`#REQUIRED` means that the document is invalid should the processor encounter a start-tag for the element type in question which specifies no value for this attribute. `#IMPLIED` means that if the attribute is omitted from an element of this type, the XML processor must inform the application that no value was specified; no constraint is placed on the behavior of the application.

If the attribute is neither `#REQUIRED` nor `#IMPLIED`, then the AttValue value contains the declared default value. If the `#FIXED` is present, the document is invalid if the attribute is present with a different value from the default. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with its value being the declared default value.

> **VALIDITY CHECK: Attribute Default Legal.** The declared default value must meet the constraints of the declared attribute type.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application, the XML processor must normalize it as follows:

1. Line-end characters (or, on some systems, record boundaries) must be replaced by single space (#x20) characters.
2. Character references and references to internal text entities must be expanded. References to external entities are an error.
3. If the attribute is not of type `CDATA`, all strings of white space must be normalized to single space characters (#x20), and leading and trailing white space must be removed.
4. Values of type `ID`, `IDREF`, `IDREFS`, `NMTOKEN`, `NMTOKENS`, or of enumerated or notation types, must be folded to uppercase.

If no DTD is present, attributes should be treated as `CDATA`.

3.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

Conditional section

```
[63] conditionalSect    ::= includeSect | ignoreSect
[64] includeSect        ::= '<![' %'INCLUDE' '[' (%markupdecl*)* ']]>'
[65] ignoreSect         ::= '<![' %'IGNORE' '[' ignoreSectContents* ']]>'
[66] ignoreSectContents ::= ((SkipLit | Comment | PI) - (Char* ']]>' Char*))
                          | ('<![' ignoreSectContents ']]>')
                          | (Char - (']' | [<'"]))
                          | ('<!' (Char - ('-' | '[')))
```

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is `INCLUDE`, then the conditional section is read and processed in the normal way. If the keyword is `IGNORE`, then the declarations within the conditional section are ignored; the processor must read the conditional section to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected. If a conditional section with a keyword of `INCLUDE` occurs within a larger conditional section with a keyword of `IGNORE`, both the outer and the inner conditional sections are ignored.

If the keyword of the conditional section is a parameter entity reference, the parameter entity is replaced by its value before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Physical Structures

An XML document may consist of one or many virtual storage units. These are called entities; they are identified by name and have content. An entity may be stored in, but need not comprise the whole of, a single physical storage object such as a file or database field. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor (and may contain the whole document).

Entities may be either binary or text. A text entity contains text data which is considered as an integral part of the document. A binary entity contains binary data with an associated notation. Only text entities may be referred to using entity references; only the names of binary entities may be given as the value of `ENTITY` attributes.

4.1 Logical and Physical Structures

The logical and physical structures (elements and entities) in an XML document must be synchronous. Tags and elements must each begin and end in the same entity, but may refer to other entities internally; comments, processing instructions, character references, and entity references must each be contained entirely within a single entity. Entities must each contain an integral number of elements, comments, processing instructions, and references, possibly together with character data not contained within any element in the entity, or else they must contain non-textual data, which by definition contains no elements.

4.2 Character and Entity References

A character reference refers to a specific character in the ISO/IEC 10646 character set, e.g. one not directly accessible from available input devices:

Character reference

[67]	**CharRef**	::= `'&#' [0-9]+ ';'` | `'&#x' [0-9a-fA-F]+ ';'`	

An entity reference refers to the content of a named entity. General entities are text entities for use within the document itself; references to them use ampersand (`&`) and semicolon (`;`) as delimiters. In this specification, general entities are sometimes referred to with the unqualified term "entity" when this leads to no ambiguity. Parameter entities are text entities for use within the DTD, or to control processing of conditional sections; references to them use percent-sign (`%`) and semicolon (`;`) as delimiters.

Entity reference

[68]	**Reference**	::= *EntityRef* | *CharRef*	
[69]	**EntityRef**	::= `'&'` *Name* `';'`	[WFC: Entity Declared] [WFC: Text Entity] [WFC: No Recursion]
[70]	**PEReference**	::= `'%'` *Name* `';'`	[WFC: Entity Declared] [WFC: Text Entity] [WFC: No Recursion] [WFC: In DTD]

WELL-FORMEDNESS CHECK: Entity Declared. The Name given in the entity reference must exactly match the name given in the declaration of the entity, except that well-formed documents need not declare any of the following entities: amp, lt, gt, apos, quot. In valid documents, these entities must be declared, in the form specified in the section on predefined entities. In the case of parameter entities, the declaration must precede the reference.

WELL-FORMEDNESS CHECK: Text Entity. An entity reference must not contain the name of a binary entity. Binary entities may be referred to only in attribute values declared to be of type `ENTITY` or `ENTITIES`.

WELL-FORMEDNESS CHECK: No Recursion. A text or parameter entity must not contain a recursive reference to itself, either directly or indirectly.

WELL-FORMEDNESS CHECK: In DTD. In the external DTD subset, a parameter-entity reference is recognized only at the locations where the nonterminal PEReference or the special operator `%` appears in a production of the grammar. In the internal subset, parameter-entity references are recognized only when they match the InternalPERef non-terminal in the production for markupdecl.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
%ISOLat2;
```

4.3 Entity Declarations

Entities are declared thus:

Entity declaration

```
[71] EntityDecl   ::= '<!ENTITY' S %Name S %EntityDef S? '>' /* General entities */
                      | '<!ENTITY' S '%' S %Name S           /* Parameter entities */
                      %EntityDef S? '>'
[72] EntityDef    ::= EntityValue | ExternalDef
```

The Name is that by which the entity is invoked by exact match in an entity reference. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.3.1 Internal Entities

If the entity definition is an EntityValue, the defined entity is called an internal entity. There is no separate physical storage object, and the replacement text of the entity is given in the declaration. Within the EntityValue, parameter-entity references and character references are recognized and expanded immediately. General-entity references within the replacement text are not recognized at the time the entity declaration is parsed, though they may be recognized when the entity itself is referred to.

An internal entity is a text entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.3.2 External Entities

If the entity is not internal, it is an external entity, declared as follows:

External entity declaration

```
[73] ExternalDef  ::= ExternalID %NDataDecl?
[74] ExternalID   ::= 'SYSTEM' S SystemLiteral
                      |  'PUBLIC' S PubidLiteral S SystemLiteral
[75] NDataDecl    ::= S %'NDATA' S %Name                  [VC: Notation Declared]
```

If the NDataDecl is present, this is a binary data entity, otherwise a text entity.

VALIDITY CHECK: Notation Declared. The Name must match the declared name of a notation.

The SystemLiteral that follows the keyword `SYSTEM` is called the entity's system identifier. It is a URL, which may be used to retrieve the entity. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URLs are relative to the location of the entity or file within which the entity declaration occurs. Relative URLs in entity declarations within the internal DTD subset are thus relative to the location of the document; those in entity declarations in the external subset are relative to the location of the files containing the external subset.

In addition to a system literal, an external identifier may include a public identifier. An XML processor may use the public identifier to try to generate an alternative URL. If the processor is unable to do so, it must use the URL specified in the system literal.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3.3 Character Encoding in Entities

Each external text entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UCS-2. It is recognized that for some purposes, the use of additional ISO/IEC 10646 planes other than the Basic Multilingual Plane may be required. A facility for handling characters in these planes is therefore a desirable characteristic in XML processors and applications.

Entities encoded in UCS-2 must begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UCS-2 encoded documents.

Although an XML processor is only required to read entities in the UTF-8 and UCS-2, it is recognized that many other encodings are in daily use around the world, and it may be advantageous for XML processors to read entities that use these encodings. For this purpose, XML provides an encoding declaration processing instruction, which, if it occurs, must appear at the beginning of a system entity, before any other character data or markup. In the document entity, the encoding declaration is part of the XML declaration; in other entities, it is part of an encoding processing instruction:

Encoding declaration

```
[76] EncodingDecl ::= S 'encoding' Eq QEncoding
[77] EncodingPI   ::= '<?XML' S 'encoding' Eq QEncoding S? '?>'
[78] QEncoding    ::= '"' Encoding '"' | "'" Encoding "'"
[79] Encoding     ::= LatinName
[80] LatinName    ::= [A-Za-z] ([A-Za-z0-9._] | '-')*    /* Name containing only
                                                            Latin characters */
```

The values `UTF-8`, `UTF-16`, `ISO-10646-UCS-2`, and `ISO-10646-UCS-4` should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values `ISO-8859-1`, `ISO-8859-2`, ... `ISO-8859-9` should be used for the parts of ISO 8859, and the values `ISO-2022-JP`, `Shift_JIS`, and `EUC-JP` should be used for the various encoded forms of JIS X-0208. XML processors may recognize other encodings; it is recommended that character encodings registered (as charsets) with the Internet Assigned Numbers Authority (IANA), other than those just listed, should be referred to using their registered names.

It is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration.

An entity which begins with neither a Byte Order Mark nor an encoding declaration must be in the UTF-8 encoding.

While XML provides mechanisms for distinguishing encodings, it is recognized that in a heterogeneous networked environment, it may be difficult to signal the encoding of an entity reliably. Errors in this area fall into two categories:

1. failing to read an entity because of inability to recognize its actual encoding, and
2. reading an entity incorrectly because of an incorrect guess of its proper encoding.

The first class of error is extremely damaging, and given a correct encoding declaration, the second class is extremely unlikely. For these reasons, XML processors should make an effort to use all available information, internal and external, to aid in detecting an entity's correct encoding. Such information may include, but is not limited to:

- Using information from an HTTP header
- Using a MIME header obtained other than through HTTP
- Metadata provided by the native OS file system or by document management software
- Analysing the bit patterns at the front of an entity to determine if the application of any known encoding yields a valid encoding declaration. See the appendix on autodetection of character sets for a fuller description.

If an XML processor encounters an entity with an encoding that it is unable to process, it may inform the application of this fact and may allow the application to request either that the entity should be treated as an binary entity, or that processing should cease.

Examples of encoding declarations:

```
<?XML ENCODING='UTF-8'?>
<?XML ENCODING='EUC-JP'?>
```

4.3.4 Document Entity

The document entity serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity might well appear on an input stream of the processor without any identification at all.

4.4 XML Processor Treatment of Entities

XML allows character and general-entity references in two places: the content of elements (content) and attribute values (AttValue). When an XML processor encounters such a reference, or the name of an external binary entity as the value of an `ENTITY` or `ENTITIES` attribute, then:

1. In all cases, the XML processor may inform the application of the reference's occurrence and its identifier (for an entity reference, the name; for a character reference, the character number in decimal, hexadecimal, or binary form).
2. For both character and entity references, the processor must remove the reference itself from the text data before passing the data to the application.
3. For character references, the processor must pass the character indicated to the application in place of the reference.
4. For an external entity, the processor must inform the application of the entity's system identifier and public identifier if any.

5. If the external entity is binary, the processor must inform the application of the associated notation name, and the notation's associated system and public (if any) identifiers.
6. For an internal (text) entity, the processor must include the entity; that is, retrieve its replacement text and process it as a part of the document (i.e. as content or AttValue, whichever was being processed when the reference was recognized), passing the result to the application in place of the reference. The replacement text may contain both text and markup, which must be recognized in the usual way, except that the replacement text of entities used to escape markup delimiters (the entities amp, lt, gt, apos, quot) is always treated as data. (The string `AT&T` expands to `AT&T`; the remaining ampersand is not recognized as an entity-reference delimiter.)

   Since the entity may contain other entity references, an XML processor may have to repeat the inclusion process recursively.
7. If the entity is an external text entity, then in order to validate the XML document, the processor must include the content of the entity.
8. If the entity is an external text entity, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's content.

   This rule is based on the recognition that the automatic inclusion provided by the SGML and XML text entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external text entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

Entity and character references can both be used to escape the left angle bracket, ampersand, and other delimiters. A set of general entities (amp, lt, gt, apos, quot) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized, and must be treated as character data, so the numeric character references `<` and `&` may be used to escape `<` and `&` when they occur in character data.

XML allows parameter-entity references in a variety of places within the DTD. Parameter-entity references are always expanded immediately upon being recognized, and the DTD must match the relevant rules of the grammar after all parameter-entity references have been expanded. In addition, parameter entities referred to in specific contexts are required to satisfy certain constraints in their replacement text; for example, a parameter entity referred to within the internal DTD subset must match the rule for markupdecl.

Implementors of XML processors need to know the rules for expansion of references in more detail. These rules only come into play when the replacement text for an internal entity itself contains other references.

1. In the replacement text of an internal entity, parameter-entity references and character references in the replacement text are recognized and resolved when the entity declaration is parsed, before the replacement text is stored in the processor's symbol table. General-entity references in the replacement text are not resolved when the entity declaration is parsed.
2. In the document, when a general-entity reference is resolved, its replacement text is parsed. Character references encountered in the replacement text are resolved immediately; general-entity references encountered in the replacement text may be resolved or left unresolved, as described above. Character and general-entity references must be contained entirely within the entity's replacement text.

Simple character references do not suffice to escape delimiters within the replacement text of an internal entity: they will be expanded when the entity declaration is parsed, before the replacement text is stored in the symbol table. When the entity itself is referred to, the replacement text will be parsed again, and the delimiters (no longer character references) will be recognized as delimiters. To escape the characters amp, lt, gt, apos, quot in an entity replacement text, use a general-entity reference or a doubly-escaped character reference. See the appendix on expansion of entity references for detailed examples.

4.5 Predefined Entities

As mentioned in the discussion of Character Data and Markup, the characters used as markup delimiters by XML may all be escaped using entity references (for the entities amp, lt, gt, apos, quot).

All XML processors must recognize these entities whether they are declared or not. Valid XML documents must declare these entities, like any others, before using them.

If the entities in question are declared, they must be declared as internal entities whose replacement text is the single character being escaped, as shown below.

```
<!ENTITY lt     "&#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;">
<!ENTITY apos   "'">
<!ENTITY quot   '"'>
```

4.6 Notation Declarations

Notations identify by name the format of external binary entities.

Notation declarations provide a name for the notation, for use in entity and attribute-list declarations and in attribute-value specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

Notation declarations

[81]	**NotationDecl**	::=	'<!NOTATION' *S %Name S %ExternalID S*? '>'

XML processors must provide applications with the name and external identifier of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

5. Conformance

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating systems alike must report violations of the well-formedness constraints given in this specification.

Validating processors must report locations in which the document does not comply with the constraints expressed by the declarations in the DTD. They must also report all failures to fulfill the validity constraints given in this specification.

Appendices

A. XML and SGML

XML is designed to be a subset of SGML, in that every valid XML document should also be a conformant SGML document, using the same DTD, and that the parse trees produced by an SGML parser and an XML processor should be the same. To achieve this, XML was defined by removing features and options from the specification of SGML.

The following list describes syntactic characteristics which XML does not allow but which are legal in SGML. The list may not be complete.

- No tag omission.
- Special tag-form for empty elements, required for elements declared `EMPTY` and optional for other empty elements.
- Comment declarations must have the delimiters <!-- comment text --> and can't have spaces within the markup of <!-- or -->.
- No comments (`--` ... `--`) inside markup declarations.
- Comment declarations can't jump in and out of comments with `-- --`.
- No name groups for declaring multiple elements or making a single `ATTLIST` declaration apply to multiple elements.
- No `RANK` feature.
- No `CDATA` or `RCDATA` declared content in element declarations. (Use CDATA sections instead.)
- No exclusions or inclusions on content models.
- No minimization parameters on element declarations.
- Mixed content models must be optional-repeatable `OR`-groups, with `#PCDATA` first.
- No `AND` (&) content model groups.
- No `NAME[S]`, `NUMBER[S]`, or `NUTOKEN[S]` declared values for attributes. (Use `NMTOKEN[S]` or `CDATA` with application-specific validation instead.)
- No `#CURRENT` or `#CONREF` declared values for attributes.
- Attribute default values must be quoted.
- Marked sections can't have spaces within the markup of <![keyword[or]]>.
- No `RCDATA`, `TEMP`, `IGNORE`, or `INCLUDE` marked sections in document instances.
- Marked sections in document instances must use the `CDATA` keyword literally, not a parameter entity.
- No `CDATA`, `RCDATA`, or `TEMP` marked sections in the DTD.
- Some restrictions on the content of ignored marked-sections: comments, literals, and processing instructions in ignored sections may not contain the delimiter string]]>; this helps ensure that the end-point of the conditional section does not change when the section is changed from `IGNORE` to `INCLUDE`.
- No `SDATA`, `CDATA`, or bracketed internal entities.
- No `SUBDOC`, `CDATA`, or `SDATA` external entities.
- External entities must have a system identifier.
- Parameter-entity references in the internal DTD subset may occur only between declarations.

- Parameter-entity references in the external DTD subset are restricted to certain positions in the grammar, and must replace whole non-terminals of the grammar; this ensures that all valid XML documents are valid SGML, and makes the restrictions on parameter-entity replacement easier to understand and implement.
- No data attributes on `NOTATION`s or attribute value specifications on `ENTITY` declarations.
- No `SHORTREF` declarations.
- No `USEMAP` declarations.
- No `LINKTYPE` declarations.
- No `LINK` declarations.
- No `USELINK` declarations.
- No `IDLINK` declarations.
- No SGML declarations.

In most current SGML systems, XML documents should be able to use the following SGML declaration. Some systems (those which take the document character set to be a description of the input stream) may require different declarations, depending on the character set and the capacities and quantities required.

```
<!SGML -- SGML Declaration for XML --
    "ISO 8879:1986 (ENR)"

    CHARSET
        BASESET
            "ISO Registration Number 176//CHARSET
            ISO/IEC 10646-1:1993 UCS-2 with implementation
            level 3//ESC 2/5 2/15 4/5"
        DESCSET
            0       9 UNUSED
            9       2 9
            11      2 UNUSED
            13      1 13
            14     18 UNUSED
            32     95 32
            127     1 UNUSED
            128    32 UNUSED
            160 65376 160

    CAPACITY SGMLREF
        -- Capacities are not restricted in XML --
        TOTALCAP 99999999
        ENTCAP   99999999
        ENTCHCAP 99999999
        ELEMCAP  99999999
        GRPCAP   99999999
        EXGRPCAP 99999999
        EXNMCAP  99999999
        ATTCAP   99999999
        ATTCHCAP 99999999
        AVGRPCAP 99999999
        NOTCAP   99999999
        NOTCHCAP 99999999
        IDCAP    99999999
        IDREFCAP 99999999
        MAPCAP   99999999
        LKSETCAP 99999999
        LKNMCAP  99999999

    SCOPE DOCUMENT
```

```
SYNTAX
    SHUNCHAR NONE
    BASESET "ISO Registration Number 176//CHARSET
            ISO/IEC 10646-1:1993 UCS-2 with implementation
            level 3//ESC 2/5 2/15 4/5"
    DESCSET
        0 65536 0
    FUNCTION
        RE    13
        RS    10
        SPACE 32
        TAB   SEPCHAR 9
        ITAB  SEPCHAR 12288 -- ideographic space --

    NAMING
        LCNMSTRT
            224-246 248-255 257 259 261 263 265 267 269 271 273
            275 277 279 281 283 285 287 289 291 293 295 297 299
            301 303 305 307 309 311 314 316 318 320 322 324 326
            328 331 333 335 337 339 341 343 345 347 349 351 353
            355 357 359 361 363 365 367 369 371 373 375 378 380
            382 383 387 389 392 396 402 409 417 419 421 424 429
            432 436 438 441 445 453 454 456 457 459 460 462 464
            466 468 470 472 474 476 479 481 483 485 487 489 491
            493 495 498 499 501 507 509 511 513 515 517 519 521
            523 525 527 529 531 533 535 595 596 598-601 603 608
            611 616 617 623 626 643 648 650 651 658 940-943
            945-961 963-974 976 977 981 982 995 997 999 1001
            1003 1005 1007-1009 1072-1103 1105-1116 1118 1119
            1121 1123 1125 1127 1129 1131 1133 1135 1137 1139
            1141 1143 1145 1147 1149 1151 1153 1169 1171 1173
            1175 1177 1179 1181 1183 1185 1187 1189 1191 1193
            1195 1197 1199 1201 1203 1205 1207 1209 1211 1213
            1215 1218 1220 1224 1228 1233 1235 1237 1239 1241
            1243 1245 1247 1249 1251 1253 1255 1257 1259 1263
            1265 1267 1269 1273 1377-1414 7681 7683 7685 7687
            7689 7691 7693 7695 7697 7699 7701 7703 7705 7707
            7709 7711 7713 7715 7717 7719 7721 7723 7725 7727
            7729 7731 7733 7735 7737 7739 7741 7743 7745 7747
            7749 7751 7753 7755 7757 7759 7761 7763 7765 7767
            7769 7771 7773 7775 7777 7779 7781 7783 7785 7787
            7789 7791 7793 7795 7797 7799 7801 7803 7805 7807
            7809 7811 7813 7815 7817 7819 7821 7823 7825 7827
            7829 7841 7843 7845 7847 7849 7851 7853 7855 7857
            7859 7861 7863 7865 7867 7869 7871 7873 7875 7877
            7879 7881 7883 7885 7887 7889 7891 7893 7895 7897
            7899 7901 7903 7905 7907 7909 7911 7913 7915 7917
            7919 7921 7923 7925 7927 7929 7936-7943 7952-7957
            7968-7975 7984-7991 8000-8005 8017 8019 8021 8023
            8032-8039 8048-8061 8064-8071 8080-8087 8096-8103
            8112 8113 8115 8131 8144 8145 8160 8161 8165 8179
            8560-8575 65345-65370

        UCNMSTRT
            -- 305 and 383 should be 73 and 83 respectively,
            but SGML does not allow a letter to be assigned
            to UCNMSTRT --
            192-214 216-222 376 256 258 260 262 264 266 268 270
            272 274 276 278 280 282 284 286 288 290 292 294 296
            298 300 302 305 306 308 310 313 315 317 319 321 323
            325 327 330 332 334 336 338 340 342 344 346 348 350
            352 354 356 358 360 362 364 366 368 370 372 374 377
            379 381 383 386 388 391 395 401 408 416 418 420 423
```

```
    428 431 435 437 440 444 452 452 455 455 458 458 461
    463 465 467 469 471 473 475 478 480 482 484 486 488
    490 492 494 497 497 500 506 508 510 512 514 516 518
    520 522 524 526 528 530 532 534 385 390 393 394 398
    399 400 403 404 407 406 412 413 425 430 433 434 439
    902 904-906 913-929 931-939 908 910 911 914 920 934
    928 994 996 998 1000 1002 1004 1006 922 929
    1040-1071 1025-1036 1038 1039 1120 1122 1124 1126
    1128 1130 1132 1134 1136 1138 1140 1142 1144 1146
    1148 1150 1152 1168 1170 1172 1174 1176 1178 1180
    1182 1184 1186 1188 1190 1192 1194 1196 1198 1200
    1202 1204 1206 1208 1210 1212 1214 1217 1219 1223
    1227 1232 1234 1236 1238 1240 1242 1244 1246 1248
    1250 1252 1254 1256 1258 1262 1264 1266 1268 1272
    1329-1366 7680 7682 7684 7686 7688 7690 7692 7694
    7696 7698 7700 7702 7704 7706 7708 7710 7712 7714
    7716 7718 7720 7722 7724 7726 7728 7730 7732 7734
    7736 7738 7740 7742 7744 7746 7748 7750 7752 7754
    7756 7758 7760 7762 7764 7766 7768 7770 7772 7774
    7776 7778 7780 7782 7784 7786 7788 7790 7792 7794
    7796 7798 7800 7802 7804 7806 7808 7810 7812 7814
    7816 7818 7820 7822 7824 7826 7828 7840 7842 7844
    7846 7848 7850 7852 7854 7856 7858 7860 7862 7864
    7866 7868 7870 7872 7874 7876 7878 7880 7882 7884
    7886 7888 7890 7892 7894 7896 7898 7900 7902 7904
    7906 7908 7910 7912 7914 7916 7918 7920 7922 7924
    7926 7928 7944-7951 7960-7965 7976-7983 7992-7999
    8008-8013 8025 8027 8029 8031 8040-8047 8122 8123
    8136-8139 8154 8155 8184 8185 8170 8171 8186 8187
    8072-8079 8088-8095 8104-8111 8120 8121 8124 8140
    8152 8153 8168 8169 8172 8188 8544-8559 65313-65338

NAMESTRT
    58 95
    170 181 186 223 304 312 329 384 397 405 410 411 414
    415 422 426 427 442 443 446-451 477 496 592-594 597
    602 604-607 609 610 612-615 618-622 624 625 627-642
    644-647 649 652-657 659-680 688-696 699-705 736-740
    768-837 864 865 890 912 944 962 978-980 986 988 990
    992 1010 1011 1155-1158 1216 1369 1415 1425-1441
    1443-1465 1467-1469 1471 1473 1474 1476 1488-1514
    1520-1522 1569-1594 1601-1618 1648-1719 1722-1726
    1728-1742 1744-1747 1749-1768 1770-1773 2305-2307
    2309-2361 2364-2381 2385-2388 2392-2403 2433-2435
    2437-2444 2447 2448 2451-2472 2474-2480 2482
    2486-2489 2492 2494-2500 2503 2504 2507-2509 2519
    2524 2525 2527-2531 2544 2545 2562 2565-2570 2575
    2576 2579-2600 2602-2608 2610 2611 2613 2614 2616
    2617 2620 2622-2626 2631 2632 2635-2637 2649-2652
    2654 2672-2676 2689-2691 2693-2699 2701 2703-2705
    2707-2728 2730-2736 2738 2739 2741-2745 2748-2757
    2759-2761 2763-2765 2784 2817-2819 2821-2828 2831
    2832 2835-2856 2858-2864 2866 2867 2870-2873
    2876-2883 2887 2888 2891-2893 2902 2903 2908 2909
    2911-2913 2946 2947 2949-2954 2958-2960 2962-2965
    2969 2970 2972 2974 2975 2979 2980 2984-2986
    2990-2997 2999-3001 3006-3010 3014-3016 3018-3021
    3031 3073-3075 3077-3084 3086-3088 3090-3112
    3114-3123 3125-3129 3134-3140 3142-3144 3146-3149
    3157 3158 3168 3169 3202 3203 3205-3212 3214-3216
    3218-3240 3242-3251 3253-3257 3262-3268 3270-3272
    3274-3277 3285 3286 3294 3296 3297 3330 3331
    3333-3340 3342-3344 3346-3368 3370-3385 3390-3395
    3398-3400 3402-3405 3415 3424 3425 3585-3630
```

```
            3632-3642 3648-3653 3655-3662 3713 3714 3716 3719
            3720 3722 3725 3732-3735 3737-3743 3745-3747 3749
            3751 3754 3755 3757 3758 3760-3769 3771-3773
            3776-3780 3784-3789 3804 3805 3864 3865 3893 3895
            3897 3902-3911 3913-3945 3953-3972 3974-3979
            3984-3989 3991 3993-4013 4017-4023 4025 4256-4293
            4304-4342 4352-4441 4447-4514 4520-4601 7830-7835
            8016 8018 8020 8022 8114 8116 8118 8119 8126 8130
            8132 8134 8135 8146 8147 8150 8151 8162-8164 8166
            8167 8178 8180 8182 8183 8319 8400-8412 8417 8450
            8455 8458-8467 8469 8472-8477 8484 8486 8488
            8490-8497 8499-8504 8576-8578 12295 12321-12335
            12353-12436 12441 12442 12449-12538 12549-12588
            12593-12686 19968-40869 44032-55203 63744-64045
            64256-64262 64275-64279 64286-64296 64298-64310
            64312-64316 64318 64320 64321 64323 64324
            64326-64433 64467-64829 64848-64911 64914-64967
            65008-65019 65056-65059 65136-65138 65140
            65142-65276 65382-65391 65393-65437 65440-65470
            65474-65479 65482-65487 65490-65495 65498-65500

        LCNMCHAR "-."
        UCNMCHAR "-."

        NAMECHAR
            183 720 721 1600 1632-1641 1776-1785 2406-2415
            2534-2543 2662-2671 2790-2799 2918-2927 3047-3055
            3174-3183 3302-3311 3430-3439 3654 3664-3673 3782
            3792-3801 3872-3881 8204-8207 8234-8238 8298-8303
            12293 12337-12341 12443-12446 12540-12542 65279
            65296-65305 65392 65438 65439
        NAMECASE
            GENERAL YES
            ENTITY  NO

    DELIM
        GENERAL SGMLREF
        NET "/>"
        PIC "?>"
        SHORTREF NONE
    NAMES
        SGMLREF

    QUANTITY SGMLREF
        -- Quantities are not restricted in XML --
        ATTCNT      99999999
        ATTSPLEN    99999999
        -- BSEQLEN  not used --
        -- DTAGLEN  not used --
        -- DTEMPLEN not used --
        ENTLVL      99999999
        GRPCNT      99999999
        GRPGTCNT    99999999
        GRPLVL      99999999
        LITLEN      99999999
        NAMELEN     99999999
        -- no need to change NORMSEP --
        PILEN       99999999
        TAGLEN      99999999
        TAGLVL      99999999

FEATURES
    MINIMIZE
```

```
                DATATAG NO
                OMITTAG NO
                RANK NO
                SHORTTAG YES -- SHORTTAG is needed for NET --
            LINK
                SIMPLE NO
                IMPLICIT NO
                EXPLICIT NO
            OTHER
                CONCUR NO
                SUBDOC NO
                FORMAL NO
        APPINFO NONE
    >
```

B. Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), ideographic characters, combining characters (among others, this class contains most diacritics); these classes combine to form the class of letters. Digits, extenders, and characters which should be ignored for purposes of recognizing identifiers are also distinguished.

Characters

```
[82] BaseChar ::= [#x41-#x5A] | [#x61-#x7A]              /* Latin 1 upper and lowercase */
                | #xAA | #xB5 | #xBA | [#xC0-#xD6]       /* Latin 1 supplementary */
                | [#xD8-#xF6] | [#xF8-#xFF]
                | [#x0100-#x017F] | [#x0180-#x01F5]      /* Extended Latin-A and B */
                | [#x01FA-#x0217]
                | [#x0250-#x02A8]                        /* IPA Extensions */
                | [#x02B0-#x02B8] | [#x02BB-#x02C1]      /* Spacing Modifiers */
                | [#x02E0-#x02E4]
                | #x037A | #x0386 | [#x0388-#x038A]      /* Greek and Coptic */
                | #x038C | [#x038E-#x03A1]
                | [#x03A3-#x03CE] | [#x03D0-#x03D6]
                | #x03DA | #x03DC | #x03DE | #x03E0
                | [#x03E2-#x03F3]
                | [#x0401-#x040C] | [#x040E-#x044F]      /* Cyrillic */
                | [#x0451-#x045C] | [#x045E-#x0481]
                | [#x0490-#x04C4] | [#x04C7-#x04C8]
                | [#x04CB-#x04CC] | [#x04D0-#x04EB]
                | [#x04EE-#x04F5] | [#x04F8-#x04F9]
                | [#x0531-#x0556] | #x0559               /* Armenian */
                | [#x0561-#x0587]
                | [#x05D0-#x05EA] | [#x05F0-#x05F2]      /* Hebrew */
                | [#x0621-#x063A] | [#x0641-#x064A]      /* Arabic */
                | [#x0671-#x06B7] | [#x06BA-#x06BE]
                | [#x06C0-#x06CE] | [#x06D0-#x06D3]
                | #x06D5 | [#x06E5-#x06E6]
                | [#x0905-#x0939] | #x093D               /* Devanagari */
                | [#x0958-#x0961]
                | [#x0985-#x098C] | [#x098F-#x0990]      /* Bengali */
                | [#x0993-#x09A8] | [#x09AA-#x09B0]
                | #x09B2 | [#x09B6-#x09B9]
                | [#x09DC-#x09DD] | [#x09DF-#x09E1]
                | [#x09F0-#x09F1]
                | [#x0A05-#x0A0A] | [#x0A0F-#x0A10]      /* Gurmukhi */
                | [#x0A13-#x0A28] | [#x0A2A-#x0A30]
                | [#x0A32-#x0A33] | [#x0A35-#x0A36]
                | [#x0A38-#x0A39] | [#x0A59-#x0A5C]
                | #x0A5E | [#x0A72-#x0A74]
                | [#x0A85-#x0A8B] | #x0A8D
```

```
| [#x0A8F-#x0A91] | [#x0A93-#x0AA8]  /* Gujarati */
| [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3]
| [#x0AB5-#x0AB9] | #x0ABD | #x0AE0
| [#x0B05-#x0B0C] | [#x0B0F-#x0B10]  /* Oriya */
| [#x0B13-#x0B28] | [#x0B2A-#x0B30]
| [#x0B32-#x0B33] | [#x0B36-#x0B39]
| #x0B3D | [#x0B5C-#x0B5D]
| [#x0B5F-#x0B61]
| [#x0B85-#x0B8A] | [#x0B8E-#x0B90]  /* Tamil */
| [#x0B92-#x0B95] | [#x0B99-#x0B9A]
| #x0B9C | [#x0B9E-#x0B9F]
| [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA]
| [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9]
| [#x0C05-#x0C0C] | [#x0C0E-#x0C10]  /* Telugu */
| [#x0C12-#x0C28] | [#x0C2A-#x0C33]
| [#x0C35-#x0C39] | [#x0C60-#x0C61]
| [#x0C85-#x0C8C] | [#x0C8E-#x0C90]  /* Kannada */
| [#x0C92-#x0CA8] | [#x0CAA-#x0CB3]
| [#x0CB5-#x0CB9] | #x0CDE
| [#x0CE0-#x0CE1]
| [#x0D05-#x0D0C] | [#x0D0E-#x0D10]  /* Malayalam */
| [#x0D12-#x0D28] | [#x0D2A-#x0D39]
| [#x0D60-#x0D61]
| [#x0E01-#x0E2E] | #x0E30           /* Thai */
| [#x0E32-#x0E33] | [#x0E40-#x0E45]
| [#x0E81-#x0E82] | #x0E84           /* Lao */
| [#x0E87-#x0E88] | #x0E8A | #x0E8D
| [#x0E94-#x0E97] | [#x0E99-#x0E9F]
| [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7
| [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE]
| #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD
| [#x0EC0-#x0EC4] | [#x0EDC-#x0EDD]
| [#x0F40-#x0F47] | [#x0F49-#x0F69]  /* Tibetan */
| [#x10A0-#x10C5] | [#x10D0-#x10F6]  /* Georgian */
| [#x1100-#x1159] | [#x115F-#x11A2]  /* Hangul Jamo */
| [#x11A8-#x11F9]
| [#x1E00-#x1E9B] | [#x1EA0-#x1EF9]  /* Add'l Extended Latin */
| [#x1F00-#x1F15] | [#x1F18-#x1F1D]  /* Greek Extensions */
| [#x1F20-#x1F45] | [#x1F48-#x1F4D]
| [#x1F50-#x1F57] | #x1F59
| #x1F5B | #x1F5D | [#x1F5F-#x1F7D]
| [#x1F80-#x1FB4] | [#x1FB6-#x1FBC]
| #x1FBE | [#x1FC2-#x1FC4]
| [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3]
| [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC]
| [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC]
| #x207F                             /* Super-, subscripts */
| #x2102 | #x2107 | [#x210A-#x2113]  /* Letterlike Symbols */
| #x2115 | [#x2118-#x211D] | #x2124
| #x2126 | #x2128 | [#x212A-#x2131]
| [#x2133-#x2138]
| [#x2160-#x2182]                    /* Number forms */
| [#x3041-#x3094]                    /* Hiragana */
| [#x30A1-#x30FA]                    /* Katakana */
| [#x3105-#x312C]                    /* Bopomofo */
| [#x3131-#x318E]                    /* Hangul Jamo */
| [#xAC00-#xD7A3]                    /* Hangul syllables */
| [#xFB00-#xFB06] | [#xFB13-#xFB17]  /* Alphabetic presentation forms */
| [#xFB1F-#xFB28] | [#xFB2A-#xFB36]
| [#xFB38-#xFB3C] | #xFB3E
| [#xFB40-#xFB41] | [#xFB43-#xFB44]
| [#xFB46-#xFB4F]
| [#xFB50-#xFBB1] | [#xFBD3-#xFD3D]  /* Arabic presentation forms */
```

```
                         | [#xFD50-#xFD8F] | [#xFD92-#xFDC7]
                         | [#xFDF0-#xFDFB] | [#xFE70-#xFE72]
                         | #xFE74 | [#xFE76-#xFEFC]
                         | [#xFF21-#xFF3A] | [#xFF41-#xFF5A]  /* Half- and fullwidth forms */
                         | [#xFF66-#xFF6F] | [#xFE71-#xFF9D]
                         | [#xFFA0-#xFFBE] | [#xFFC2-#xFFC7]
                         | [#xFFCA-#xFFCF] | [#xFFD2-#xFFD7]
                         | [#xFFDA-#xFFDC]

[83] Ideographic    ::=  [#x4E00-#x9FA5] | [#xF900-#xFA2D] | #x3007
                         | [#x3021-#x3029]
[84] CombiningChar  ::=  [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486]
                         | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD]
                         | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652]
                         | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF]
                         | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED]
                         | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D
                         | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983]
                         | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4]
                         | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7
                         | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F
                         | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D]
                         | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC
                         | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD]
                         | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43]
                         | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57]
                         | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8]
                         | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03]
                         | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D]
                         | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4]
                         | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6]
                         | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48]
                         | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A]
                         | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9]
                         | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19]
                         | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F
                         | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95]
                         | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9
                         | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099
                         | #x309A | #xFB1E | [#xFE20-#xFE23]
[85] Letter         ::=  BaseChar | Ideographic
[86] Digit          ::=  [#x30-#x39]            /* ISO 646 digits */
                         | [#x0660-#x0669]      /* Arabic-Indic digits */
                         | [#x06F0-#x06F9]      /* Eastern Arabic-Indic digits */
                         | [#x0966-#x096F]      /* Devanagari digits */
                         | [#x09E6-#x09EF]      /* Bengali digits */
                         | [#x0A66-#x0A6F]      /* Gurmukhi digits */
                         | [#x0AE6-#x0AEF]      /* Gujarati digits */
                         | [#x0B66-#x0B6F]      /* Oriya digits */
                         | [#x0BE7-#x0BEF]      /* Tamil digits (no zero) */
                         | [#x0C66-#x0C6F]      /* Telugu digits */
                         | [#x0CE6-#x0CEF]      /* Kannada digits */
                         | [#x0D66-#x0D6F]      /* Malayalam digits */
                         | [#x0E50-#x0E59]      /* Thai digits */
                         | [#x0ED0-#x0ED9]      /* Lao digits */
                         | [#x0F20-#x0F29]      /* Tibetan digits */
                         | [#xFF10-#xFF19]      /* Fullwidth digits */
[87] Ignorable      ::=  [#x200C-#x200F]        /* zw layout */
                         | [#x202A-#x202E]      /* bidi formatting */
                         | [#x206A-#x206F]      /* alt formatting */
                         | #xFEFF               /* zw nonbreak space */
```

```
[88] Extender ::= #xB7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                  | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309B-#x309E]
                  | [#x30FC-#x30FE] | #xFF70 | #xFF9E | #xFF9F
```

C. Expansion of Entity and Character References

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion.

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity example:

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to &example; will cause the text to be reparsed, at which time the start- and end-tags of the p element will be recognized and the three references will be recognized and expanded, resulting in a p element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?XML version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity xx is stored in the symbol table with the value %zz;. Since the replacement text is not rescanned, the reference to parameter entity zz is not recognized. (And it would be an error if it were, since zz is not yet declared.)
- in line 5, the character reference < is expanded immediately and the parameter entity zz is stored with the replacement text <!ENTITY tricky "error-prone" >, which is a well-formed entity declaration.
- in line 6, the reference to xx is recognized, and the replacement text of xx (namely %zz;) is parsed. The reference to zz is recognized in its turn, and its replacement text (<!ENTITY tricky "error-prone" >) is parsed. The general entity tricky has now been declared, with the replacement text error-prone.
- in line 8, the reference to the general entity tricky is recognized, and it is expanded, so the full content of the test element is the self-describing (and ungrammatical) string "This sample shows a error-prone method."

D. Deterministic Content Models

For compatibility, it is required that content models in element declarations be deterministic. SGML requires deterministic content models (it calls them `unambiguous'); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model `((b, c) | (b, d))` is non-deterministic, because given an initial b the parser cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read `(b, (c | d))`. An initial b now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either c or d would be accepted.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Br÷ggemann-Klein 1991.

E. Autodetection of Character Encodings

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use--which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases.

Because each XML entity not in UTF-8 or UCS-2 format *must* begin with an XML encoding declaration, in which the first characters must be '`<?XML`', any conforming processor can detect, after four octets of input, which of the following cases apply (in reading this list, it may help to know that in UCS-4, '<' is `#x0000003C` and '?' is `#x0000003F`, and the Byte Order Mark required of UCS-2 data streams is `#xFEFF`):

- `00 00 00 3C`: UCS-4, big-endian machine (1234 order)
- `3C 00 00 00`: UCS-4, little-endian machine (4321 order)
- `00 00 3C 00`: UCS-4, unusual octet order (2143)
- `00 3C 00 00`: UCS-4, unusual octet order (3412)
- `FE FF 00 3C`: UCS-2, big-endian
- `FF FE 3C 00`: UCS-2, little-endian
- `00 3C 00 3F`: UCS-2, big-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 00 3F 00`: UCS-2, little-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 3F 58 4D`: UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the ASCII characters, the encoding declaration itself may be read reliably
- `4C 6F E7 D4`: EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
- other: UTF-8 without an encoding declaration, or else the data are corrupt, fragmentary, or enclosed in a wrapper of some kind

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to ASCII characters, a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-line labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

F. A Trivial Grammar for XML Documents

The grammar given in the body of this specification is relatively simple, but for some purposes it is convenient to have an even simpler one. A very simple, though non-conforming, XML processor could parse a well-formed XML document using the following simplified grammar, recognizing all element boundaries correctly, though not expanding entity references and not detecting all errors:

Trivial text grammar

```
[89] simpleDoc  ::= (SimpleData | Markup)*
[90] SimpleData ::= [^<&]*                                        /* cf. PCData */
[91] SimpleLit  ::= ('"' [^"]* '"')
                  | ("'" [^']* "'")                               /* cf. SkipLit */
[92] Markup     ::= '<' Name (S Name S? '=' S? SimpleLit)* S? '>' /* start-tags */
                  | '<' Name (S Name S? '=' S? SimpleLit)* S? '/>' /* empty elements */
                  | '</' Name S? '>'                              /* end-tags */
                  | '&' Name ';'                                  /* entity references */
                  | '&#' [0-9]+ ';'                               /* decimal character references */
                  | '&#x' [0-9a-fA-F]+ ';'                        /* hexadecimal character references */
                  | '<!--' (Char* - (Char* '--' Char*)) '-->'     /* comments */
                  | '<?' (Char* - (Char* '?>' Char*)) ''?>''      /* processing instructions */
                  | '<![CDATA[' (Char* - (Char* ']]>' Char*)) ']]>' /* CDATA sections */
                  | '<!DOCTYPE' (Char - ('[' | ']'))+ ('[' simpleDTD* ']')? '>' /* doc type declaration */
[93] simpleDTD  ::= '<!--' (Char* - (Char* '--' Char*)) '-->'     /* comment */
                  | '<?' (Char* - (Char* '?>' Char*)) ''?>''      /* processing instruction */
                  SimpleLit
                  (Char - (']' | '<' | '"' | "'"))+
                  '<!' (Char - ('-'))+                            /* declarations other than comment */
```

Most processors will require the more complex grammar given in the body of this specification.

G. References

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Br÷ggemann-Klein, Anne. *Regular Expressions into Finite Automata*. Universitßt Freiburg, Institut f÷r Informatik, Bericht 33, Juli 1991.

Br÷ggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universitßt Freiburg, Institut f÷r Informatik, Bericht 38, Oktober 1991.

ISO (International Organization for Standardization). *ISO/IEC 8879-1986 (E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 5).

ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators*. 1991.

The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

H. W3C SGML Editorial Review Board

This specification was prepared and approved for publication by the W3C SGML Editorial Review Board (ERB). ERB approval of this specification does not necessarily imply that all ERB members voted for its approval. At the time it approved this specification, the SGML ERB comprised the following members:

Jon Bosak, Sun *(Chair)*
James Clark *(Technical Lead)*
Tim Bray, Textuality *(Co-editor)*
C. M. Sperberg-McQueen, U. of Ill. *(Co-editor)*
Steve DeRose, INSO
Dave Hollander, HP
Eliot Kimber, Highland
Tom Magliery, NCSA
Eve Maler, ArborText
Jean Paoli, Microsoft
Peter Sharpe, SoftQuad