Extensible Markup Language (XML)

W3C Working Draft 14-Nov-96

This version:
http://www.w3.org/pub/WWW/TR/WD-xml-961114.html

Previous versions:

Latest version:

Editors:
Tim Bray (Textuality) <tbray@textuality.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Status of this memo

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.

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. For this reason, XML has been designed for ease of implementation, and for interoperability with both SGML and HTML.

Note on status of this document: This is even more of a moving target than the typical W3C working draft. Several important decisions on the details of XML are still outstanding - members of the W3C SGML Working Group will recognize these areas of particular volatility in the spec, but those who are not intimately familiar with the deliberative process should be careful to avoid actions based on the content of this document, until the notice you are now reading has been removed.

Table of Contents

- 1. Introduction
  - 1.1 Origin and Goals
  - 1.2 Relationship to Other Standards
  - 1.3 Notation
1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects stored on computers and partially describes the behavior of programs which process these objects. Such objects are called XML documents. 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, structure, and arbitrary attribute-value pairs associated with that 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 some of the required behavior of an XML processor in terms of the manner it must read XML data, and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by a Generic SGML Editorial Review Board formed under the auspices of the W3 Consortium in 1996 and chaired by Jon Bosak of Sun Microsystems, with the very active participation of a Generic SGML Working Group also organized by the W3C. The membership of these groups is given in an appendix.

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 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 (0.01) is for internal discussion within the SGML ERB only. It should not be distributed outside the ERB.

Known problems in version 0.01:

1. Several items in the bibliography have no references to them; several references in the text do not point to anything in the bibliograhy.
2. The EBNF grammar has not been checked for completeness, and has at least two start productions.
3. The description of conformance in the final section is incomplete.
4. Language exists in the spec which describes the effect of several decisions which have not been taken. Specifically, XML may have INCLUDE/IGNORE marked sections as does SGML, the comment syntax may change, XML may have CONREF attributes, the 8879 syntax for EMPTY elements may be outlawed, XML may choose to rule out what 8879 calls "ambiguous" content models, XML may choose to prohibit overlap between enumerated attribute values for different attributes, the handling for attribute values in the absence of a DTD may be specified, there may be a way to signal whether the DTD is complete, the DTD summary may be dropped, and XML may support parameter entities, and XML may predefine a large number of character entities, for example those from HTML 3.2.

1.2 Relationship to Other Standards

Other standards relevant to users and implementors of XML include:

2. Unicode, ISO 10646. This specification depends on ISO standard 10646 and the technically identical Unicode Standard, Version 2.0, which define the encodings and meanings of the characters which make up XML text data.
3. IETF RFC 1738. This specification defines the syntax and semantics of Uniform Resource Locators, or URLs.

1.3 Notation

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

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, with an initial lowercase letter if they have a more complex definition (i.e. if they require a stack for proper recognition). Literal strings are quoted; unless otherwise noted they are not case-sensitive. The distinction between symbols which can and cannot be recognized using simple regular-expressions is made for clarity only. It may be reflected in the boundary between an implementation's lexical scanner and its parser, but there is are no assumptions about the placement of such a boundary, nor even that the implementation has separate modules for parser and lexical scanner.

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

- `#NN` where `NN` is a decimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated
- `#xNN` where `NN` is a hexadecimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated
- `[#xNN-#xNN], [a-zA-Z]` matches any character with a value in the range(s) indicated (inclusive)
- `[^#xNN-#xNN],[^a-z]` matches any character with a value outside the range indicated
- `[^abc]` matches any character with a value not among the characters given
- `"string"` matches the literal string given inside the double quotes
- `'string'` matches the literal string given inside the single quotes
- `a b` `a` followed by `b`
- `a | b` `a` or `b` but not both
- `a?` `a` or nothing; optional `a`
- `a+` one or more occurrences of `a`
- `a*` zero or more occurrences of `a`
- `(expression)` `expression` is treated as a unit; allows subgroups to carry the operators ?, *, or +
- `/* ... */` comment
- `[ WFC: ... ]`
Well-formedness check; this identifies by name a check for well-formedness that is associated with a production.

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

1.4 Terminology

Some terms used with special meaning in this specification are:

may
Conforming data and software may but need not behave as described.

must
Conforming data and software must 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.

reportable error
An error which conforming software must report to the user, unless the user has explicitly disabled error reporting.

validity constraint
A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must 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 reportable errors.

at user option
Conforming software may or must (depending on the verb in the sentence) provide users a means to select the behavior described; it must also allow the user not to select it.

match
Case-insensitive match: two strings or names being compared must be identical except for differences between upper- and lower-case letters in scripts which have such a distinction. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and base+diacritic forms) match only if they have the same representation, except for case differences, in both strings. Case folding must be performed as specified in The Unicode Standard, Version 2.0, section 4.1; in particular, it is recommended that case-insensitive matching be performed by folding uppercase letters to lowercase, not vice versa.

exact(ly) match
Case-sensitive match: two strings or names being compared must be identical. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and base+diacritic forms) match only if they have the same representation in both strings.

for compatibility
A feature of XML included solely to ensure that XML remains compatible with SGML; the expectation is that in many cases, those aspects of SGML that are not required to satisfy XML’s requirements but mandated only to achieve conformance may be removed or replaced in the near future by the organizations that maintain that standard.

1.5 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more blank characters, carriage returns, line feeds, or tabs.
For some purposes, characters are classified as letters, digits, or other characters:

\[
S := (#x0020 | #x000a | #x000d | #x0009) +
\]

\[< 2 \text{Name} >\]

\[
\text{Character} ::= [#x20-#xFF] \quad /* \text{any ISO 10646 32-bit code} */
\]

\[
\text{Letter} ::= [#x41-#x5A] | [#x61-#x7A] | #xAA | #xB5 | #xBA |
\]

\[
\text{Character} ::= [#x0030-#x0039] \quad /* \text{Correct this table using section 4.5 of Unicode} */
\]

\[
\text{Digit} ::= [#x0030-#x0039] \quad 2.0 \quad /* \text{ISO 646 digits} */
\]

\[
\text{Digit} ::= [#x0060-#x0069] \quad /* \text{Arabic-Indic digits} */
\]
A **Name** is a token beginning with a letter or hyphen and continuing with letters, digits, hyphens, or full stops (together known as name characters). The use of any name beginning with a string which matches "-XML-" in a fashion other than those described in this specification is a *reportable error*.

A **Number** is a sequence of digits. An **Nmtoken** (name token) is any mixture of name characters.

### < 3 Names, Numbers, and Tokens >

| Name ::= (Letter | '-' ) (Letter | Digit | '-' | '.')* |
| Number ::= Digit+ |
| Nmtoken ::= (Letter | Digit | '-' | '.')+ |
| Nmtokens ::= Nmtoken (S Nmtoken)* |
Literal data is any quoted string containing neither the quotation mark used as a delimiter nor angle brackets. It may contain entity and character references.

2. Documents

A textual object is an XML Document if it is either valid, or failing that, 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; it begins in a "root" or document entity, which may refer to other entities, and so on ad infinitum. Entities referred to are embedded in the document at the point of reference.

The document contains declarations, elements, comments, entity references, character references, and processing instructions, all of which are indicated in the document by explicit markup. These concepts and their markup are all explained elsewhere in this specification.

The two structures 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.

2.2 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 name and the size in bytes of the binary data in a storage object, XML provides no information about and places no constraints on binary data. In fact, so-called binary data may in fact be textual, perhaps even well-formed XML text; but its identification as binary means that an XML processor need not parse it in the fashion described by the specification. XML text data is a sequence of characters. A character is an atomic unit of text represented by a bit string; valid bit strings and their meanings are specified by ISO 10646.

Users may extend the ISO 10646 character repertoire, in the rare cases where this is necessary, by making use of 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 signalling which of the
two are in use, and 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 10646 character set may be referred to by the decimal or hexadecimal equivalent of its bit string:

\[
\text{Hex} ::= [0-9a-fA-F]
\]
\[
\text{Hex4} ::= \text{Hex} \text{Hex} \text{Hex} \text{Hex}
\]
\[
\text{CharRef} ::= '\&#' \text{Number} ';'
\]
\[
| '\&u-' \text{Hex4} ';
\]

### 2.3 Syntax of Text 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, marked sections, document type declarations, and processing instructions. The simplest form of XML processor thus could parse a well-formed XML document using the following rules:

<table>
<thead>
<tr>
<th>6 Trivial text grammar</th>
</tr>
</thead>
</table>

```
Trivial ::= (PCDATA | Markup)*
Eq ::= $? '=' $?
Markup ::= '<' Name ($ Name Eq QuotedCData)* $? /* start-tags */
         | '</' Name $? '>' /* end-tags */
         | '<' Name ($ Name Eq QuotedCData)* $? /* empty elements */
         | '/'
         | '&' Name ';' /* entity references */
         | '&#' Number ';' /* character references */
         | '&u-' Hex4 ';' /* character references */
         | '<!-- [^-]* ('-' [^-]+)* '-->' /* comments */
         | '<![CDATA[ ' MsData ' ]]> /* marked sections */
         | '<!DOCTYPE' (Name | $)+ ('[' '[^]]*' /* doc type declaration */
           )? '>'
         | '<?' [^?]* ('?' [^>]+)* '?>' /* processing instructions */
```

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

All text that is not markup constitutes the character data of the document.
The ampersand character (&) and the left and right angle bracket (< and >) may appear in their literal form only when used as markup delimiters, or within comments, processing instructions, or marked sections. If they are needed in the text, they must be represented using the strings "&amp;", "&lt;", and "&gt;".Parsed character data is thus any string of characters which does not contain the start-delimiter of any markup. Character data is any string of characters not including the marked-section-close delimiter, "]>". For convenience, the single-quote character (') may be represented as "&sq;'", and the double-quote character (") as "&dq;'".

### 7 Character Data

PCDATA ::= [^<&]*
MsData ::= [^]*)((('['] ([^])) | ([']) [^>])) [^]*)*

### 2.4 Comments

Comments may appear anywhere that character data may, except in a marked section (more properly, comments appearing in a marked section will not be recognized as such). 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. An XML processor must inform the application of the length of comments if they are not passed through, to enable the application to keep track of the correct location of objects in the XML document. For compatibility, the string -- (double-hyphen) may not occur within comments.

### 8 Comments

Comment ::= '<!--' [^-]* ('-' [^-]+)* '-->'

### 2.5 Processing Instructions

Processing instructions, usually referred to as PIs, allow the XML processor to pass instructions directly to selected applications.

### 9 Processing Instructions

PI ::= '<?' Name S [^?]*)(([''] [^>])*) '?'>

PIs are not part of the document's character data, but must be passed through to the application. The Name which follows the '?' at the beginning of the PI is called the PI target. It is normally the name of a declared notation, identifying the application to which it belongs. The use of the PI target "XML" in any other way other than those described in this specification is a reportable error.

### 2.6 Marked Sections

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

### 10 Marked Sections

MS ::= MsStart MsData MsEnd
MsStart ::= '<![CDATA['
Within a marked section, only the \texttt{MsEnd} string is recognized, so that angle brackets and ampersands may occur in their literal form and need not be escaped using \texttt{&lt;} etc. Marked sections cannot nest.

### 2.7 White Space Handling

While authoring XML documents, it is often convenient to use "white space" (spaces, tabs, 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, to be retained in the delivered version, is common; for example in poetry or source code.

An XML processor must provide two distinct white space handling modes, \texttt{COLLAPSE} and \texttt{KEEP}, and have the ability to apply these modes on a per-element basis. They operate as follows:

- **COLLAPSE**
  - The XML processor must suppress (i.e. not pass to the application) all white space in an element which immediately follows the \texttt{start-tag} and all that which immediately precedes the \texttt{end-tag}. In the element's character data, it must convert all sequences of white space characters to a single space (\#x0020) character, before passing the data to the application.

- **KEEP**
  - The XML processor must suppress initial line break characters which immediately follow the \texttt{start-tag} of the element, and which immediately precede its \texttt{end-tag}. All other characters in the character data of the element must be passed to the application without change.

The white space handling mode is signaled through the use of a reserved \texttt{attribute}, whose declaration is as follows:

\[
\text{<!ATTLIST * -XML-SPACE (KEEP|COLLAPSE) #IMPLIED>}
\]

where the ** signifies that this attribute may apply to any element.

The \texttt{value} of the attribute sets the white space handling mode for the element and for any contained elements. Unless otherwise specified, an XML processor is to set the white space handling mode for the root element of a document to \texttt{COLLAPSE}.

### 2.8 Prolog and Document Type Declaration

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 \texttt{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 \texttt{element} in the document.

\[
\text{< 11 XML document >}
\]

\[
\text{document ::= Prolog element Misc*}
\]

\[
\text{Prolog ::= EncodingDecl? Misc* RMDecl? Misc* (doctypedecl | DtdSummary)? Misc*}
\]

\[
\text{Misc ::= Comment | PI | S}
\]
For example, the following is a complete XML document, well-formed but not valid:

```xml
<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 of the necessary markup declarations.

```xml
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

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

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

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

The character-set label `<XML encoding="UTF-8"` indicates that the document entity is encoded using the UTF-8 transformation of ISO 10646. The legal values of the character set code are given in the discussion of character encodings.

Individual markup declaration types are described elsewhere in this specification.

### 2.9 Required Markup Declaration

In many 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. A document author can communicate whether or not DTD processing is necessary using a required markup declaration (abbreviated RMD) processing instruction:

```xml
<?XML encoding="UTF-8"?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]> 
<greeting>Hello, world!</greeting>
```
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 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 to parse the containing document directly.

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

1. undistinguished empty elements, and these elements occur in the document, or
2. attributes with default values, and elements to which these attributes apply appear in the document, or
3. entities, and references to those entities appear in the document.

If such declarations occur in the internal but not the external subset, the RMD should 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, the effect is identical to an RMD with the value ALL.

3. Logical Structures

Each XML document contains one or more elements, the boundaries of which are either delineated by start-tags and end-tags, or, for empty elements, are limited to 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.

3.1 Start- and End-Tags

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

```
< 14 Start-tag Recognition >

STag ::= '<' Name (S Attribute)* S? '>

Attribute ::= Name Eq QuotedCData [ VC: Attribute Value Type ]
```

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

**Validity Constraint - Attribute Value Type:**
Attribute values must be of the type declared for the attribute. (For attribute types, see the discussion of attribute declarations.)

The end of every element which is not empty is marked by an end-tag:

```
< 15 End-tag Recognition >

ETag ::= '</' Name S? '>
```

The Name, once again, gives the element's type.
If an element is empty, the start-tag constitutes the whole element. An empty element takes a special form:

```xml
<16 Tags for empty elements>

EmptyElement ::= '<' Name (S Attribute)* S? '/>';
</16 Tags for empty elements>
```

For compatibility, an empty element may have the same syntax as a start-tag; in this case, it cannot be recognized based on syntax, but must be declared as being empty. Such elements are called undistinguished empty elements.

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

```xml
<17 Content of elements>

content ::= (element | PCDATA | MS | PI | Comment)*

element ::= EmptyElement /* empty elements */
   | STag content ETag

Well-Formedness Constraint - GI Match:
The Name in an element's end-tag must match that in the start-tag.
</17 Content of elements>
```

### 3.2 Well-Formed XML Documents

A textual object is said to be a well-formed XML document if, first, it matches the production above labeled XML Document, and if:

1. There are no undistinguished empty elements which have not been specified as such in an element declaration.
2. For each entity reference which appears in the document, the entity name has been declared in the document type declaration.

Matching the "XML Document" production implies that:

1. It contains one or more elements.
2. There is one element, called the root, for which neither the start-tag nor the end-tag are 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, delineated by start- and end-tags, nest within each other properly.

As a consequence of this, for each non-root element C, there is one other element P 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 the child of P.

### 3.3 Element Declaration

The element structure of an XML document may be declared fully or partially. Such declarations serve two purposes:

1. To establish a set of structural constraints, i.e. a grammar, for a class of documents, and to verify that documents are valid, i.e. comply with that grammar.
2. To make XML documents well-formed by declaring undistinguished empty elements.
An element declaration constrains the element's type and its content. The content constraints will be described first; four forms are available: empty, any, mixed content, and element content.

Declarations often contain references to element types, for example when constraining which element types can appear as children of others, and which attributes may be attached to which element types. At user option, an XML processor may issue a warning when no element is declared whose type matches that given, but this is not an error.

An element declaration takes the form:

```xml
<18 Element declarations>

elementdecl ::= <!ELEMENT Name ('EMPTY' | 'ANY' | Mixed | elements ) S? '>

[VC: Unique Element Declaration]
```

where Name identifies the type of the element.

**Validity Constraint - Unique Element Declaration:**
No element type may be declared more than once.

For compatibility, an element type may be declared using the keyword EMPTY; this is an undistinguished empty element.

If an element type is declared using the keyword ANY, then there are no constraints on its content aside from its being well-formed: it may contain subelements of any type and number, interspersed with character data.

### 3.3.1 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:

```xml
<19 Mixed-content declaration>

Mixed ::= '(' S? '#PCDATA' ( S? '|' S? Name )* S? ')'*
       | '(' S? '#PCDATA' S? ')

where the Names give the types of elements that may appear as children.
```

### 3.3.2 Element Content

An element type may be declared to have element content, consisting only of other elements. In this case, the constraint includes a content model, a context-free 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:

```xml
<20 Element-content models>

elements ::= cp

cp ::= (Name | choice | seq) ('?' | '*' | '+')?
```
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 respectively. The syntax and meaning is identical to that used in the productions in this specification.

### 3.4 Attribute Declaration

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 that discussion. Attribute declarations may be used:

1. To define the set of attributes pertaining to a given element type.
2. To establish a set of type constraints on these attributes.
3. 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:

```
< 21 Attribute list declaration >

AttlistDecl ::= '<!ATTLIST' S Name AttDef+ [ VC: Unique Attribute-List Declaration ] S? '>'
AttDef ::= S Name S AttType S Default [ VC: Unique Attribute Name ]
```

The Name in the AttlistDecl rule is the type of an element; it is a reportable error to declare attributes for an element type not itself declared. The Name in the AttDef rule is the name of the attribute.

**Validity Constraint - Unique Attribute-List Declaration:**
At most one attribute-list declaration may be provided for a given element type.

**Validity Constraint - Unique Attribute Name:**
An attribute name may appear at most once in an attribute-list declaration.

### 3.4.1 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:

```
< 22 Attribute types >

AttType ::= StringType | TokenizedType | EnumeratedType
StringType ::= 'CDATA'
TokenizedType ::= 'ID' [ VC: ID ]
```
Validity Constraint - ID:
Values of this type must be valid Names. 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 Constraint - Idref:
Values of this type must be one (for IDREFS one or more) Names, which must each match the value of an ID attribute on some element in the XML document; i.e. IDREF values must match some ID.

Validity Constraint - Entity Name:
Values of this type must be one (for ENTITIES one or more) Names, which must each exactly match the name of an external binary entity declared in the XML DTD.

Validity Constraint - Name token:
Values of this type must consist of a string matching the Nmtoken nonterminal of the grammar defined in this specification. The XML processor must normalize the case of the attribute value before passing it to the application.

Validity Constraint - Name tokens:
Values of this type must consist of a string matching the Nmtokens nonterminal of the grammar defined in this specification. The XML processor must reduce white space to single blanks, and normalize the case of the attribute value, before passing it to the application.

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

< 23 Enumerated attribute declarations >

EnumeratedType ::= NotationType | Enumeration
NotationType ::= 'NOTATION' S '(' S? Name (S? '|' S)? S? [ VC: Notation Name]* S? ')'
3.4.2 Attribute Defaults

An attribute declaration may provide 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:

```
< 24 Attribute defaults >

Default ::= '#REQUIRED'
   | '#IMPLIED'
   | ( '#FIXED'? Literal)
```

#REQUIRED means that it is a reportable error should the processor encounter a start-tag where this attribute is omitted, i.e. could occur but does not. #IMPLIED means that if an attribute is omitted, 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 Literal value contains the declared default value. If the #FIXED is present, it is a reportable error 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 assume that the attribute is present and has the declared default value.

3.5 Partial DTD Information

The prolog of an XML document may contain an abbreviated summary of the DTD for the convenience of non-validating processors.

```
< 25 DTD summary >

DtdSummary ::= (EmptyInfo | TextInfo | NoText | IdInfo | DefaultInfo)*
EmptyInfo ::= '<?XML' S 'empty' S 'names' Eq QuotedNames S? '?>'
TextInfo ::= '<?XML' S 'text' S 'names' Eq QuotedNames S? '?>'
NoText ::= '<?XML' S 'notext' S 'names' Eq QuotedNames S? '?>'
IdInfo ::= '<?XML' S 'idinfo' S
   'ids' Eq QuotedPairs S
   'refs' Eq QuotedPairs S? '?>'
QuotedPairs ::= '"' Pairs '"' | "" Pairs '""
Pairs ::= ('*' | Name) S Name (S ('*' | Name) S Name)*
DefaultInfo ::= '<?XML' S 'default' Name (S Name Eq Literal)* S? '?>'
```

The empty, text, and notext declarations give, respectively, lists of element types declared as empty, mixed-content, or element-content elements. The idinfo declaration lists attributes declared as id or idref(s) (in the ids and refs values, respectively). Each attribute name is given as an element-attribute pair; an asterisk matches all element types which have attributes of the given name. The default declaration specifies default values for the attributes of a given element type.

For example:

```
<?XML empty names="ptr xptr pb" ?>
```
Elements not listed as notext are implicitly declared as text elements; the text declaration is thus not strictly necessary.

4. Physical Structures

An XML document may consist of one or many storage units. The unit of storage is the entity; entities are identified by name and have content. 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 to be considered as an integral part of the document. A binary entity contains binary data identified by notation. Text and binary entities cannot be distinguished by syntax; their types are established in their declarations, described below.

4.1 Character and Entity References

A character reference refers to a specific character in the ISO 10646 character set, often one not directly accessible from available input devices:

```
< 26 Character Reference >
```

```
CharRef ::= '\&' Number ';'
   | '\&u-' Hex4 ';
```

An entity reference refers to the content of a named entity.

```
< 27 Entity Reference >
```

```
Reference ::= EntityRef | CharRef
EntityRef ::= '\&' Name ';
```

Well-Formedness Constraint - Entity Declared:
The Name given in the entity reference must match exactly the name given in the declaration of the entity.

Well-Formedness Constraint - Text Entity:
An entity reference may 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 Constraint - No Recursion:
A text entity may not contain a reference to itself.

4.2 Declaring Entities
Entities are declared thus:

```
<EntityDecl> ::= '&lt;ENTITY' S Name EntityDef S? ' &gt;'
EntityDef ::= Literal | ExternalDef;
```

The *Name* is that by which the entity is invoked by *exact match* in an *entity reference*.

### 4.2.1 Internal Entities

If the entity definition is just a *Literal*, this is called an *internal* entity - there is no separate storage unit, and the content of the entity is given in the declaration. An internal entity is a *text entity*.

### 4.2.2 External Entities

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

```
<ExternalDecl> ::= NDataDecl? 'SYSTEM' Literal

NDataDecl ::= 'NDATA' S Name S
```

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

**Validity Constraint - Notation Declared:**

The *Name* must match one of those in a *notation declaration*.

The *Literal* that follows the keyword *SYSTEM* called the entity's *system identifier*. It is a URL, which may be used to retrieve the content of the entity.

### 4.2.3 Character Encoding in Entities

Each 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 the UTF-8 or UCS-2 encodings. It is recognized that for some advanced work, particularly with Asian languages, the use of the UTF-16 encoding is required, and correct handling of this encoding is a desirable characteristic in XML processor implementations.

Entities encoded in UCS-2 must begin with the Byte Order Mark described by ISO 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, U+FEFF) -- 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 non-standard encodings. For this purpose, XML provides an encoding declaration *processing instruction*, which must appear at the beginning of an entity, before any other character data or *markup*:

```
<EncodingDecl>
```
It is a reportable error for an entity including an encoding declaration to be stored 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, there is often difficulty in reliably signaling the encoding of an entity. 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 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:

1. Using information from an HTTP header
2. Using a MIME header obtained other than through HTTP
3. Metadata provided by the native OS filesystem or through other mechanisms
4. Analyzing the bit patterns at the front of an entity to determine if the application of any known encoding yields a valid encoding declaration.

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

4.2.4 The 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.

4.3 XML Processor Treatment of Entities

When an XML processor encounters a character reference or an entity name (in a reference or attribute value):

1. In all cases, the XML processor must inform the application of the reference's occurrence and its identifier (for an entity reference, the name; for a character reference, the "&..." string.)
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 insert the indicated ISO 10646 bit pattern into the text data at the location of the reference before passing it to the application.
4. For an external entity reference, the processor must inform the application of the entity's system identifier.
5. If the external entity is binary, the processor must inform the application of the associated notation name, and the notation's associated system identifier.

6. For an internal (text) entity, the processor must include the entity; that is, retrieve its content, and treat the content as a part of the text data of the XML document, processing this replacement text data (which may contain both text and markup) to determine what data to pass to the application. Since the entity's content text 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 inclusion semantic provided by the SGML and XML text entity mechanism, primarily designed to support modularity in authoring, may not be 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. While this behavior would not allow the application to validate the document, it is compliant with this specification.

### 4.4 Notation Declaration

Notations identify by name the format of external binary entities. 

Notation declarations provide a name for the notation, for use in entity and attribute 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.

\[
\text{Notation declarations} \quad < \text{Notation declarations} > \\
\text{NotationDecl ::= '<!NOTATION' S Name S Extid S? '} '>
\]

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, API address, 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 systems 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.

### A. XML and SGML

XML is designed to be a subset of SGML, in that every valid XML document should also be a valid 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.

Despite this, there are a small number of cases where XML fails to be a pure subset of SGML, including:

1. XML’s white space handling rules are much less elaborate than those of SGML. One effect is that for elements using the KEEP mode for white-space handling, an XML processor will pass through a few record-separator markers that an SGML processor will eat.
2. XML defines, for documents, the property of being well-formed; this does not really correspond to any SGML concept.

The following list describes features which are available in SGML but not in XML. It may not be complete.

1. Tag omission
2. The CONCUR, LINK, DATATAG, and SHORTREF features
3. The "&" connector in content models
4. Inclusions and exclusions in content models
5. CURRENT, CONREF, NAME, NAMES, NUMBER, NUMBERS, NUTOKEN, and NUTOKENS declarations for attributes
6. The NET construct
7. Abstract syntax
8. Capacities and quantities
9. Comments appearing within other markup declarations
10. Public Identifiers
11. Omission of quotes on attribute values

B. References

ISO/IEC 8879

ISO/IEC 10646

ISO/IEC 10744

Unicode

C. Working Group and Editorial Review Board Membership

C.1 Working Group

This specification was prepared with the assistance and participation of the W3C SGML Working Group. Publication of this specification does not necessarily imply that all members of the working group agree with its content. At the time this specification was prepared, the W3C SGML Working Group comprised the following members.
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Sharon Adler, EBT</td>
<td><a href="mailto:sca@ebt.com">sca@ebt.com</a></td>
</tr>
<tr>
<td>2.</td>
<td>Paula Angerstein, Texcel</td>
<td><a href="mailto:paula@texcel.no">paula@texcel.no</a></td>
</tr>
<tr>
<td>3.</td>
<td>Todd Bauman, U. of Maryland</td>
<td><a href="mailto:bbaumal@cs.umbc.edu">bbaumal@cs.umbc.edu</a></td>
</tr>
<tr>
<td>4.</td>
<td>Anders Berglund, EBT</td>
<td><a href="mailto:alb@ebt.com">alb@ebt.com</a></td>
</tr>
<tr>
<td>5.</td>
<td>Karl Best, Novell</td>
<td><a href="mailto:kbest@novell.com">kbest@novell.com</a></td>
</tr>
<tr>
<td>6.</td>
<td>Michel Biezunski, High Text</td>
<td><a href="mailto:michel@hightext.com">michel@hightext.com</a></td>
</tr>
<tr>
<td>7.</td>
<td>Harvey Bingham, SGML consultant</td>
<td><a href="mailto:hbingham@acm.org">hbingham@acm.org</a></td>
</tr>
<tr>
<td>8.</td>
<td>Steve Brown, InfoObjects</td>
<td><a href="mailto:sbrown@cortland.com">sbrown@cortland.com</a></td>
</tr>
<tr>
<td>9.</td>
<td>Martin Bryan, SGML Centre</td>
<td><a href="mailto:mbryan@sgml.u-net.com">mbryan@sgml.u-net.com</a></td>
</tr>
<tr>
<td>10.</td>
<td>Mark Buckley, Microsoft</td>
<td><a href="mailto:mbuckley@microsoft.com">mbuckley@microsoft.com</a></td>
</tr>
<tr>
<td>11.</td>
<td>Len Bullard, Lockheed-Martin</td>
<td><a href="mailto:cbullard@HiWAAY.net">cbullard@HiWAAY.net</a></td>
</tr>
<tr>
<td>12.</td>
<td>Lou Burnard, Oxford University</td>
<td><a href="mailto:lou@vax.ox.ac.uk">lou@vax.ox.ac.uk</a></td>
</tr>
<tr>
<td>13.</td>
<td>Steve Byrne, JavaSoft</td>
<td><a href="mailto:steve.byrne@sun.com">steve.byrne@sun.com</a></td>
</tr>
<tr>
<td>14.</td>
<td>Kurt Conrad, Sagebrush Group</td>
<td><a href="mailto:conrad@cbvcp.com">conrad@cbvcp.com</a></td>
</tr>
<tr>
<td>15.</td>
<td>Paul Cope, auto-graphics</td>
<td><a href="mailto:prc@auto-graphics.com">prc@auto-graphics.com</a></td>
</tr>
<tr>
<td>16.</td>
<td>Keith Corbett, Harlequin</td>
<td><a href="mailto:kmc@harlequin.com">kmc@harlequin.com</a></td>
</tr>
<tr>
<td>17.</td>
<td>Robin Cover</td>
<td><a href="mailto:robin@acadcomp.sil.org">robin@acadcomp.sil.org</a></td>
</tr>
<tr>
<td>18.</td>
<td>Derek Denny-Brown, TechnoTeacher</td>
<td><a href="mailto:derdb@techno.com">derdb@techno.com</a></td>
</tr>
<tr>
<td>19.</td>
<td>Dave Durand, Boston University</td>
<td><a href="mailto:dgd@cs.bu.edu">dgd@cs.bu.edu</a></td>
</tr>
<tr>
<td>20.</td>
<td>Carol Ellerbeck, INSO</td>
<td><a href="mailto:Carol_Ellerbeck@inso.com">Carol_Ellerbeck@inso.com</a></td>
</tr>
<tr>
<td>21.</td>
<td>Joe English, CR Laboratories</td>
<td><a href="mailto:jenglish@crl.com">jenglish@crl.com</a></td>
</tr>
<tr>
<td>22.</td>
<td>Ralph Ferris, Fujitsu OSSI</td>
<td><a href="mailto:ralph@ossi.com">ralph@ossi.com</a></td>
</tr>
<tr>
<td>23.</td>
<td>Lee Fugal, Digital Equipment</td>
<td><a href="mailto:fogal@zk3.dec.com">fogal@zk3.dec.com</a></td>
</tr>
<tr>
<td>24.</td>
<td>Todd Freter, Novell</td>
<td><a href="mailto:tfreter@novell.com">tfreter@novell.com</a></td>
</tr>
<tr>
<td>25.</td>
<td>Matthew Fuchs, Disney Imagineering</td>
<td><a href="mailto:matt@wdi.disney.com">matt@wdi.disney.com</a></td>
</tr>
<tr>
<td>26.</td>
<td>Charles Goldfarb</td>
<td><a href="mailto:charles@sgmlsource.com">charles@sgmlsource.com</a></td>
</tr>
<tr>
<td>27.</td>
<td>Paul Grosso, ArborText</td>
<td><a href="mailto:paul@arbortext.com">paul@arbortext.com</a></td>
</tr>
<tr>
<td>28.</td>
<td>Eduardo Gutentag</td>
<td><a href="mailto:eduardo@eng.sun.com">eduardo@eng.sun.com</a></td>
</tr>
<tr>
<td>29.</td>
<td>Hasse Haitto, Synex</td>
<td><a href="mailto:haitto@synex.se">haitto@synex.se</a></td>
</tr>
<tr>
<td>30.</td>
<td>Catherine Hamon, Hightext</td>
<td><a href="mailto:hamon@hightext.com">hamon@hightext.com</a></td>
</tr>
<tr>
<td>31.</td>
<td>Ken Holman, Microstar</td>
<td><a href="mailto:gkholman@microstar.com">gkholman@microstar.com</a></td>
</tr>
<tr>
<td>32.</td>
<td>Rick Jelliffe, Allette Systems</td>
<td><a href="mailto:ricko@allette.com.au">ricko@allette.com.au</a></td>
</tr>
<tr>
<td>33.</td>
<td>Alan Karben, Wall Street Journal</td>
<td><a href="mailto:karben@interactive.wsj.com">karben@interactive.wsj.com</a></td>
</tr>
<tr>
<td>34.</td>
<td>Pär Karlsson, Ericsson</td>
<td><a href="mailto:etxpkar@stdoca.ericsson.se">etxpkar@stdoca.ericsson.se</a></td>
</tr>
<tr>
<td>35.</td>
<td>Bill Lindsey, BDM</td>
<td><a href="mailto:blindsey@BDMTech.com">blindsey@BDMTech.com</a></td>
</tr>
<tr>
<td>36.</td>
<td>Per Åke Ling, Ericsson</td>
<td><a href="mailto:etxperl@stdoca.ericsson.se">etxperl@stdoca.ericsson.se</a></td>
</tr>
<tr>
<td>37.</td>
<td>Debbie Lapeyre, Atlis</td>
<td><a href="mailto:dlapeyre@netcom.com">dlapeyre@netcom.com</a></td>
</tr>
<tr>
<td>38.</td>
<td>Megan MacMillan, BDM Technologies</td>
<td><a href="mailto:megan@bdmtech.com">megan@bdmtech.com</a></td>
</tr>
<tr>
<td>39.</td>
<td>Christopher Maden, EBT</td>
<td><a href="mailto:crm@ebt.com">crm@ebt.com</a></td>
</tr>
<tr>
<td>40.</td>
<td>James Mason, ORNL</td>
<td><a href="mailto:masonjd@ornl.gov">masonjd@ornl.gov</a></td>
</tr>
<tr>
<td>41.</td>
<td>Alex Milowski, Copernican Solutions</td>
<td><a href="mailto:alex@copsol.com">alex@copsol.com</a></td>
</tr>
<tr>
<td>42.</td>
<td>Steve Newcomb, TechnoTeacher</td>
<td><a href="mailto:srn@techno.com">srn@techno.com</a></td>
</tr>
<tr>
<td>43.</td>
<td>Gavin Nicol, EBT</td>
<td><a href="mailto:gtn@ebt.com">gtn@ebt.com</a></td>
</tr>
<tr>
<td>44.</td>
<td>Nancy Paisner, Hitachi</td>
<td><a href="mailto:nancy@hi.com">nancy@hi.com</a></td>
</tr>
<tr>
<td>45.</td>
<td>Gary Palmer, ActiveSystems</td>
<td><a href="mailto:gpalmer@sonetis.com">gpalmer@sonetis.com</a></td>
</tr>
<tr>
<td>46.</td>
<td>Dave Peterson, SGML Works</td>
<td><a href="mailto:davep@acm.org">davep@acm.org</a></td>
</tr>
<tr>
<td>47.</td>
<td>Paul Prescod, U. of Waterloo</td>
<td><a href="mailto:papresco@calum.csclub.uwaterloo.ca">papresco@calum.csclub.uwaterloo.ca</a></td>
</tr>
<tr>
<td>48.</td>
<td>Lynne Price, SGML consultant</td>
<td><a href="mailto:lprice@ix.netcom.com">lprice@ix.netcom.com</a></td>
</tr>
<tr>
<td>49.</td>
<td>Arjun Ray, Q2-II</td>
<td><a href="mailto:aray@nmnds.com">aray@nmnds.com</a></td>
</tr>
<tr>
<td>50.</td>
<td>Bill Smith, SunSoft</td>
<td><a href="mailto:bill.smith@sun.co">bill.smith@sun.co</a></td>
</tr>
<tr>
<td>51.</td>
<td>Bob Stayton, SCO</td>
<td><a href="mailto:bobs@sco.com">bobs@sco.com</a></td>
</tr>
<tr>
<td>52.</td>
<td>Robert Streich, Schlumberger</td>
<td><a href="mailto:streich@slb.com">streich@slb.com</a></td>
</tr>
<tr>
<td>53.</td>
<td>Jeff Suttor, SunSoft</td>
<td><a href="mailto:jeff.suttor@sun.com">jeff.suttor@sun.com</a></td>
</tr>
<tr>
<td>54.</td>
<td>James Tauber, U. of Western Australia</td>
<td><a href="mailto:jtauber@library.uwa.edu.au">jtauber@library.uwa.edu.au</a></td>
</tr>
</tbody>
</table>

https://www.w3.org/TR/WD-xml-961114#sec1.1
C.2 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 had the following members:

1. Jon Bosak, Sun (jon.bosak@sun.com), chair
2. Tim Bray, Textuality (tbray@textuality.com), editor
3. James Clark (jjc@jclark.com), technical lead
4. Dan Connolly (connolly@w3.org), W3C contact
5. Steve DeRose, EBT (sjd@ebt.com)
6. Dave Hollander, HP (dmh@hpsgml.fc.hp.com)
7. Eliot Kimber, Passage Systems (kimber@passage.com)
8. Tom Magliery, NCSA (mag@ncsa.uiuc.edu)
9. Eve Maler, ArborText (elm@arbortext.com)
10. Jean Paoli, Microsoft (jeanpa@microsoft.com)
11. Peter Sharpe, SoftQuad (peter@sqwest.bc.ca)
12. C. M. Sperberg-McQueen, U. of Ill. at Chicago (cmsmcq@uic.edu), editor