The Association for Computers and the Humanities (ACH)

The Association for Computational Linguistics (ACL)

The Association for Literary and Linguistic Computing (ALLC)

Guidelines for Electronic Text Encoding and Interchange

Edited by

C. M. Sperberg-McQueen and Lou Burnard

TEI P2, Chapter 42
Text Encoding Initiative
Chicago, Oxford

December 4, 1992

Draft Version 2, December 4, 1992

Note

This is the current draft (as of December 4, 1992) of one section of the *Guidelines for Electronic Text Encoding and Interchange*, version 2 (also known as TEI P2), produced by the Text Encoding Initiative. This fascicle contains:

- a table of contents for TEI P2, indicating the status of each section as of the release date of this fascicle
- a draft for one section of TEI P2
- entries derived from this section for the reference list of tags of TEI P2 (if any)
- a User Response and Comment Form

Please feel free to send us your comments on the contents of this draft and their applicability to your needs. Comments may be sent to the TEI-L list server for general discussion, or directly to either of the TEI editors (adresses on the User Response and Comment form). Bear in mind in reading, however, that TEI P2 is a reference manual, not a tutorial in the use of electronic text, and unavoidably assumes either a certain amount of prior knowledge or patience with unfamiliar technical material. Note also that some matters which may appear obscure in one section may well be treated in more detail elsewhere in the full text of TEI P2.

TEI P2, like P1, is a working draft. The observant will find several typographic and presentational flaws in the current text: for example, cross-references are not properly resolved in the LaTeX versions and some parts of the reference material are not yet complete. We hope to rectify as many as possible of these flaws in the coming months, but have decided not to allow them to delay any further the release of the draft. We hope that those interested in the Guidelines will find the current state of the text readable and usable enough for the purpose of substantive comment. As far as possible, all comments received on this section will be taken into account in the preparation of the next version of the Guidelines, which is due by the end of 1992.

Updates on the status of the draft as a whole will be distributed automatically to all subscribers of the Listserv list TEI-L. To subscribe to TEI-L, send electronic mail to the address LISTSERV@UICVM (or Listserv@uicvm.uic.edu) containing the single line

```
subscribe tei-l J. Smith
```

(substituting your name for “J. Smith”). TEI-L is also the appropriate place to pose questions or offer public comments on the TEI guidelines and other relevant issues. The TEI-L file server contains all the sections of TEI P2 thus far released, as well as other TEI materials. Its contents are shadowed at various other sites around the world; for more information consult one of the following documents:

- TEI ED J8, “Obtaining the Second Version of the TEI Guidelines (TEI P2)” (describes how to retrieve electronic copies of TEI P2 and the various formats they are available in)

- TEI ED J9, “Obtaining Paper Copies of the Second Version of the TEI Guidelines (TEI P2)” (describes how to request paper copies of TEI P2, for those without electronic mail access)

These documents are available from the TEI-L file server or through the editors at the addresses given on the User Response and Comment Form.

This document is made possible in part by financial support from the U.S. National Endowment for the Humanities, an independent federal agency; Directorate General XIII of the Commission of the European Communities; the Andrew W. Mellon Foundation; and the Social Science and Humanities Research Council of Canada.

Overall Table of Contents for TEI P2

The following list shows the expected contents of TEI P2 as a whole and indicates the current status of each chapter.

Contents

Part 8

Reference Material

Chapter 42

Formal Grammar for the TEI-Interchange-Format Subset of SGML

This grammar is intended to help make SGML more comprehensible for formal manipulation. For this reason, a number of simplifications have been undertaken, which are described below in section 42.8, "Differences from ISO 8879," on page 16. These simplifications may cause this grammar to accept some documents not accepted by the official grammar. As far as is known, however, the grammar provided here will recognize any valid SGML document in the TEI Interchange Format.

For ease in relating this grammar to the formal grammar defined in ISO 8879, comments for each group here give the numbers of the related productions in that grammar. Where the changes to SGML syntax suggested by the SGML working group in its document ISO/IEC JTC1 / SC18 / WG8 / N1035 would affect the productions here, that fact is noted after the affected production.

Each sub-grammar given here has been checked for ambiguity with *bison*, a public-domain workalike for *yacc*, and *flex*, a public-domain workalike for *lex*. The bison and flex source files, including the simple modifications needed to implement the recognition-mode stack, are available from the TEI.

The SGML declaration grammar and the DTD grammar have no ambiguities. The document grammar has several ambiguities, which are discussed in section 42.5, "Grammar for Document Instance," on page 10.

42.1 Notation

The notation used here is based on notations commonly used in writing context-free grammars. All non-terminals are written as single tokens.

::=: "is defined as"
a b: instance of *a* followed by instance of *b*, possibly separated by white space; white space may be required if *a* and *b* cannot otherwise be delimited
a | b: instance of *a* or instance of *b*
a || b: instance of *a* followed immediately by an instance of *b*, without any intervening white space
/* */: comment; right-hand sides comprising only a comment are used to call attention to productions in which the left-hand sides can reduce to the empty string
//: comment; runs to end of line

All non-quoted strings are non-terminals. All terminals are quoted.

42.2 Grammar for SGML Document (Overview)

An SGML document is preceded by an SGML declaration and a prolog comprising one or more document-type declarations. It may be accompanied by one or more subdocument entities, text entities, non-SGML entities, etc., but for simplicity these last are not discussed here.

```
SGMLdoc ::= SGMLdeclaration prolog docinstance  // cf. 1, 2
```

The grammars for the SGML declaration, the prolog, and the document instance are provided in the following three sections.

42.3 Grammar for SGML Declaration

This grammar is substantively the same as that in ISO 8879; it does not reflect the restrictions placed on SGML declarations for TEI-conformant documents.

```
SGMLdeclaration ::= '<!SGML'                // cf. 171
                    'ISO 8879:1986'
                    'CHARSET' charset       // cf. 172
                    'CAPACITY' capacity     // cf. 180
                    'SCOPE' scope           // cf. 181
                    'SYNTAX' syntax
                    'FEATURES'              // cf. 195-198
                        'MINIMIZE'
                                'DATATAG'  yesno
                                'OMITTAG'  yesno
                                'RANK'     yesno
                                'SHORTTAG' yesno
                        'LINK'
                                'SIMPLE' count
                                'IMPLICIT' yesno
                                'EXPLICIT' count
                        'OTHER'
                                'CONCUR' count
                                'SUBDOC' count
                                'FORMAL' yesno
                    'APPINFO' appinfo       // cf. 199
                    '>'
/*
** Strictly speaking, the blank in 'ISO 8879:1986' may be
** replaced by any amount or type of white space.
*/

charset     ::= baseset descset             // cf. 173
              | charset baseset descset
baseset     ::= 'BASESET' pubid             // cf. 174
/* For pubid, see Common Constructs below */
descset     ::= 'DESCSET' chardesc          // cf. 175
              |  descset  chardesc
chardesc    ::= NUMBER NUMBER NUMBER        // cf. 176-179
```

```
              | NUMBER NUMBER LITERAL
              | NUMBER NUMBER 'UNUSED'
/*
** The first two numbers in chardesc identify starting point
** and a number of characters in the described character set;
** the third number gives the starting position for that run
** in the base set; a literal provides a description of the
** character(s); 'UNUSED' marks the run as unused.
*/

capacity    ::= 'PUBLIC' pubid              // cf. 180
              | 'SGMLREF' caplist
caplist     ::= capname NUMBER
              | caplist capname NUMBER
capname     ::= 'TOTALCAP' | 'ENTCAP'   | 'ENTCHCAP'
              | 'ELEMCAP'  | 'GRPCAP'   | 'EXGRPCAP'
              | 'EXNMCAP'  | 'ATTCAP'   | 'ATTCHCAP'
              | 'AVGRPCAP' | 'NOTCAP'   | 'NOTCHCAP'
              | 'IDCAP'    | 'IDREFCAP' | 'MAPCAP'
              | 'LKSETCAP' | 'LKNMCAP'      // cf. Fig. 5

scope       ::= 'DOCUMENT' | 'INSTANCE'     // cf. 181

syntax      ::= 'PUBLIC' pubid              // cf. 182-183
              | 'PUBLIC' pubid switchlist
              | 'SHUNCHAR' shunchars        // cf. 184
                charset                     // cf. 185
                'FUNCTION'                  // cf. 186
                    'RE' NUMBER 'RS' NUMBER 'SPACE' NUMBER
                    funlist
                'NAMING'                    // cf. 189
                    'LCNMSTRT' LITERAL
                    'UCNMSTRT' LITERAL
                    'LCNMCHAR' LITERAL
                    'UCNMCHAR' LITERAL
                    'NAMECASE' 'GENERAL' yesno 'ENTITY' yesno
                'DELIM'                     // cf. 190-92
                    'GENERAL' 'SGMLREF' gendelim
                    'SHORTREF' srdelim
                'NAMES'    'SGMLREF' nameset // cf. 193
                'QUANTITY' 'SGMLREF' quantityset // cf. 194
/*
** Document WG8 / N1035 suggests allowing multiple literals
** after each keyword in the NAMING section; this may be
** effected by adding the non-terminal literalset after each
** LITERAL.
*/

switchlist  ::= 'SWITCHES' NUMBER NUMBER    // cf. 183
              | switchlist NUMBER NUMBER

shunchars   ::= 'NONE'                      // cf. 184
              | shunlist
```

```
shunlist    ::= 'CONTROLS'
              | NUMBER
              | shunlist NUMBER

funlist     ::= NAME funclass NUMBER        // cf. 186-187
              | funlist NAME funclass NUMBER
funclass    ::= 'FUNCHAR' | 'MSICHAR' | 'MSOCHAR' // cf. 188
              | 'MSSCHAR' | 'SEPCHAR'

gendelim    ::= /* nil */                   // cf. 191
              | gendelim delimname LITERAL
delimname   ::= 'AND'   | 'COM'  | 'CRO'    // cf. Fig. 3
              | 'DSC'   | 'DSO'  | 'DTGC'   // clause 9.6
              | 'DTGO'  | 'ERO'  | 'ETAGO'
              | 'GRPC'  | 'GRPO' | 'LIT'
              | 'LITA'  | 'MDC'  | 'MDO'
              | 'MINUS' | 'MSC'  | 'NET'
              | 'OPT'   | 'OR'   | 'PERO'
              | 'PIC'   | 'PIO'  | 'PLUS'
              | 'REFC'  | 'REP'  | 'RNI'
              | 'SEQ'   | 'SHORTREF'
              | 'STAGO' | 'TAGC' | 'VI'

srdelim     ::= 'SGMLREF' literalset        // cf. 192
              | 'NONE' literalset
literalset  ::= /* nil */
              | literalset LITERAL
nameset     ::= /* nil */
              | nameset sgmlname NAME
/*
** WG8 / N1035 substitutes a LITERAL for the NAME of the
** preceding rule; the value must be a NAME in the declared
** concrete syntax, but it need not be a legal name in the
** reference concrete syntax.
*/

sgmlname    ::= 'ANY'      | 'ATTLIST'  | 'CDATA'
              | 'CONREF'   | 'CURRENT'  | 'DEFAULT'
              | 'DOCTYPE'  | 'ELEMENT'  | 'EMPTY'
              | 'ENDTAG'   | 'ENTITIES' | 'ENTITY'
              | 'FIXED'    | 'ID'       | 'IDLINK'
              | 'IDREF'    | 'IDREFS'   | 'IGNORE'
              | 'IMPLIED'  | 'INCLUDE'  | 'INITIAL'
              | 'LINK'     | 'LINKTYPE' | 'MD'
              | 'MS'       | 'NAME'     | 'NAMES'
              | 'NDATA'    | 'NMTOKEN'  | 'NMTOKENS'
              | 'NOTATION' | 'NUMBER'   | 'NUMBERS'
              | 'NUTOKEN'  | 'NUTOKENS' | 'O'
              | 'PCDATA'   | 'PI'       | 'POSTLINK'
              | 'PUBLIC'   | 'RCDATA'   | 'RE'
              | 'REQUIRED' | 'RESTORE'  | 'RS'
              | 'SDATA'    | 'SHORTREF' | 'SIMPLE'
              | 'SPACE'    | 'STARTTAG' | 'SUBDOC'
```

```
              | 'SYSTEM'   | 'TEMP'     | 'USELINK'
              | 'USEMAP'

quantityset ::= /* nil */
              | quantityset quantity NUMBER
quantity    ::= 'ATTCNT'   | 'ATTSPLEN' | 'BSEQLEN' // Cf.
              | 'DTAGLEN'  | 'DTEMPLEN' | 'ENTLVL'  // Fig. 6
              | 'GRPCNT'   | 'GRPGTCNT' | 'GRPLVL'
              | 'LITLEN'   | 'NAMELEN'  | 'NORMSEP'
              | 'PILEN'    | 'TAGLEN'   | 'TAGLVL'

yesno       ::= 'NO' | 'YES'
count       ::= 'NO' | 'YES' NUMBER

appinfo     ::= 'NONE' | LITERAL            // cf. 199
/*
** The literal string is restricted to letters, digits,
** whitespace, and 'specials':  viz. any of
** ' ( ) + , - . / : = ?
*/
```

42.4 Grammar for DTD

An SGML *prolog* is composed of one or more document type declarations; if multiple DTDs are present, the SGML declaration must include “CONCUR YES” in the FEATURES section. A document type declaration names the root element of the document and declares (in an external file, in a DTD subset, or both) elements, attributes, notations, and entities used in the document instance. Interspersed with these declarations may be comments and processing instructions.

```
prolog      ::= misc dtdseq                 // cf. 7, 9

/* For misc, see Common Constructs below */

dtdseq      ::= dtd misc                    // cf. 7
              | dtdseq dtd misc

                                            // cf. 110, 111, 30
dtd         ::= '<!DOCTYPE' NAME extid '[' dtdsubset ']' '>'
              | '<!DOCTYPE' NAME       '[' dtdsubset ']' '>'
              | '<!DOCTYPE' NAME extid                   '>'
              | '<!DOCTYPE' NAME                         '>'

dtdsubset   ::= /* */                       // cf. 112, 113, 114
              | dtdsubset elementdecl
              | dtdsubset attlistdecl
              | dtdsubset notationdecl
              | dtdsubset entitydecl
              | dtdsubset commdecl
              | dtdsubset procinst

/* Element Declarations */                  // cf. 116
```

```
elementdecl ::= '<!ELEMENT' elemtype minimiz contentdecl '>'

elemtype    ::= NAME                         // cf. 117, 30, 72
              | '(' namegrp ')'

namegrp     ::= andnames                     // cf. 69, 131
              | ornames
              | seqnames
seqnames    ::= NAME
              | seqnames ',' NAME
ornames     ::= NAME     '|' NAME
              | ornames  '|' NAME
andnames    ::= NAME     '&' NAME
              | andnames '&' NAME

minimiz     ::= min min                      // cf. 122-124
min         ::= 'O' | '-'

contentdecl ::= 'CDATA'                      // cf. 125, 126
              | 'RCDATA'
              | 'EMPTY'
              | 'ANY'  exceptions
              | model  exceptions

model       ::= '(' tokengrp ')'             // cf. 127
              | '(' tokengrp ')?'
              | '(' tokengrp ')*'
              | '(' tokengrp ')+'

tokengrp    ::= seqtokens                    // cf. 127
              | ortokens
              | andtokens
seqtokens   ::= token
              | seqtokens ',' token
ortokens    ::= token     '|' token
              | ortokens  '|' token
andtokens   ::= token     '&' token
              | andtokens '&' token

token       ::= '#PCDATA'                    // cf. 128-130
              | NAME || occurrence
              | model

occurrence  ::= /* nil */                    // cf. 132
              | '?'
              | '*'
              | '+'

exceptions  ::= exclusions inclusions        // cf. 138-140
exclusions  ::= /* nil */
              | '-(' namegrp ')
inclusions  ::= /* nil */
              | '+(' namegrp ')'
```

```
/* Attribute List Declarations */
attlistdecl ::= '<!ATTLIST' associated attdeflist '>' // Cf. 141

associated  ::= elemtype
              | assocnotatn

assocnotatn ::= '#NOTATION' NAME            // Cf. 149.1
              | '#NOTATION' '(' namegrp ')'

attdeflist  ::= attdef                      // Cf. 142
              | attdeflist attdef

attdef      ::= NAME valtype default        // Cf. 143-44

valtype     ::= 'CDATA'   |                 // Cf. 145
              | 'ENTITY'  | 'ENTITIES'
              | 'ID'      |
              | 'IDREF'   | 'IDREFS'
              | 'NAME'    | 'NAMES'
              | 'NMTOKEN' | 'NMTOKENS'
              | 'NUMBER'  | 'NUMBERS'
              | 'NUTOKEN' | 'NUTOKENS'
              | 'NOTATION' '(' namegrp ')'
              | '(' nmtokgrp ')'

nmtokgrp    ::= nmtokcom | nmtokbar | nmtokamp   // Cf. 68, 131
nmtokcom    ::= nametoken
              | nmtokcom  ',' nametoken
nmtokbar    ::= nametoken '|' nametoken
              | nmtokbar  '|' nametoken
nmtokamp    ::= nametoken '&' nametoken
              | nmtokamp  '&' nametoken
nametoken   ::= NAME
              | NUMBER
              | NUMTOKEN

default     ::=  value                      // Cf. 147
              | '#FIXED' value
              | '#REQUIRED'
              | '#CURRENT'
              | '#CONREF'
              | '#IMPLIED'

/* For value, see Common Constructs below */


/* Notation Declarations */                 // cf. 148-49, 41
notationdecl ::= '<!NOTATION' NAME extid '>'


/* Entity Declarations */                   // cf. 101-04
```

```
entitydecl  ::= '<!ENTITY' NAME       enttext '>'
              | '<!ENTITY' '#DEFAULT' enttext '>'
              | '<!ENTITY' '% ' NAME  enttext '>'
/*
** Strictly, any white space is acceptable after the %
** in a parameter entity declaration, not just a single
** space.
*/

enttext     ::= LITERAL                     // cf. 105-08
              | 'CDATA'    LITERAL
              | 'SDATA'    LITERAL
              | 'PI'       LITERAL
              | 'STARTTAG' LITERAL
              | 'ENDTAG'   LITERAL
              | 'MS'       LITERAL
              | 'MD'       LITERAL
              | extid enttype

enttype     ::= /* */                       // cf. 108-109, 149.2
              | 'SUBDOC'
              | 'CDATA' NAME
              | 'CDATA' NAME '[' attspecset ']'
              | 'NDATA' NAME
              | 'NDATA' NAME '[' attspecset ']'
              | 'SDATA' NAME
              | 'SDATA' NAME '[' attspecset ']'
/* For attspecset, see Common Constructs below. */

extid       ::= 'SYSTEM'                    // cf. 73
              | 'SYSTEM' sysid
              | 'PUBLIC' pubid
              | 'PUBLIC' pubid sysid
/* For pubid, see Common Constructs below */

sysid       ::= LITERAL                     // cf. 75
```

42.5 Grammar for Document Instance

The SGML document instance is composed of one element (the *root element*), followed optionally by white space, comments, and processing instructions. The root element, like any other, has a start-tag, content, and an end-tag, or only a start-tag (if it is empty). The specific sort of content recognized within an element depends upon its element declaration.

```
docinstance ::= element misc                // cf. 10-12

element     ::= start-tag content end-tag   // cf. 13
              | start-tag
/*
** N.B. this BNF is for minimal SGML documents; tags may
** not be omitted.
```

```
*/

start-tag   ::= '<' || NAME attspecset '>'  // cf. 14, 28-30
              | '<(' namegrp ')' || NAME attspecset '>'
/*
** The name group is used only if the SGML declaration
** specifies CONCUR YES.
** For attspecset, see Common Constructs below.
*/

content     ::= mixedcontent                // cf. 24
              | elemcontent
              | rcdata
              | cdata

mixedcontent ::= /* nil */                  // cf. 25
              | mixedcontent STRING
              | mixedcontent element
              | mixedcontent misccontent

elemcontent ::= /* nil */                   // cf. 26
              | elemcontent element
              | elemcontent misccontent
cdata       ::= STRING                      // cf. 47
              | cdata STRING
rcdata      ::= STRING                      // cf. 46
              | rcdata STRING
/*
** White space is ignored between elements in element content,
** but not in mixed content.  In CDATA and RCDATA, start-tag
** delimiters are not recognized.  In CDATA, entity reference
** delimiters are not recognized.  An element's content model
** determines whether it is scanned for mixed content, element
** content, CDATA content, or RCDATA content.
*/

misccontent ::= commdecl                    // cf. 27
              | procinst
/*
** For commdecl and procinst, see Common Constructs below.
** Omitted here for simplicity are short-reference and
** link-set use declarations and short references (which are
** not allowed in TEI interchange format), entity references
** (which are assumed to be handled by the lexical scanner),
** and marked-section declarations (also in lexical
** scanner).
*/

end-tag     ::= '</' || NAME '>'            // cf. 19, 21
              | '</(' namegrp ')' || NAME '>'
              | '</>'
/*
** The name group is used only if the SGML declaration
```

```
** specifies CONCUR YES.
** N.B. The last form (short end-tag) is not allowed in the
** TEI Interchange Format.
*/
```

The document-instance grammar just given contains two sets of formal ambiguities. One set concerns the distinction among mixed content, element content, RCDATA, and CDATA, which depends not on the document content but on the definition of the element within which they appear. These conflicts can be eliminated by eliminating the distinction and assigning the task of distinguishing content type (and alerting the lexical analyzer to modify its behavior) to the semantic rules of the parser, rather than to the syntax.

The second set of ambiguities arises in connection with start-tags: after a start-tag, empty elements are complete, others not, and the ambiguity can be resolved only by consulting the DTD, not by lookahead. Such conflicts can be avoided by defining document content as an unstructured sequence of start-tags, end-tags, and data content; the parser's semantic actions must enforce the pairing and nesting of start- and end-tags and the distinction between empty and non-empty elements. Despite the ambiguities, the grammar given here seems to express the nature of SGML documents more clearly than the unambiguous alternative and so has not been changed; the changes needed to eliminate the parsing conflicts are these:

- delete *element*, *mixedcontent*, *elemcontent*, *cdata*, and *rcdata*.
- redefine *docinstance* and *content* as follows:

```
docinstance ::= start-tag content          // cf. 10-12
content     ::= /* nil */                  // cf. 25
              | content STRING
              | content start-tag
              | content end-tag
              | content misccontent
```

Applications using this simplification must distinguish mixed content, element content, RCDATA, and CDATA using other methods than document syntax. They can check the appropriate matching of start- and end-tags using a simple element stack with provision for empty elements. Some SGML normalizers provide explicit end-tags for empty elements to simplify this task.

42.6 Common Syntactic Constructs

This section defines syntactic constructions used in more than one of the three preceding grammar fragments.

```
misc        ::= /* nothing */              // cf. 8
              | misc commdecl
              | misc procinst

commdecl    ::= '<!--' STRING '--' commseq '>'   // cf. 91, 92
              | '<!>'
```

```
commseq     ::= /* nil */
              | commseq '--' STRING '--'

procinst    ::= '<?' STRING '>'             // cf. 44

attspecset  ::= /* nil */                   // cf. 31
              | attspecset attspec

attspec     ::= NAME '=' value              // cf. 32
              | value
/*
** NAME may be omitted only if the attribute has an
** enumerated range of values and the value is an unquoted
** name token.
*/

value       ::= LITERAL                     // Cf. 33
              | NAME
              | NUMBER
              | NUMTOKEN

pubid       ::= LITERAL                     // cf. 74, 76
/*
** The literal string is restricted to letters, digits,
** whitespace, and 'specials':  viz. any of
** ' ( ) + , - . / : = ?
*/
```

42.7 Lexical Scanner

The grammar given above assumes a lexical scanner which

- scans for the terminal strings represented here in quotes
- scans for certain other token types (listed below)
- handles white space and some comments without returning them
- recognizes and expands entity references appropriately without notifying the parser

N.B. the literals given here for delimiters, keywords, and in the definitions of character classes and character types, are those used in the reference concrete syntax of SGML; a full SGML parser must be able to use other concrete syntaxes.

The token types to be returned by the lexical scanner include (in addition to the literals used in the grammars above):

- name
- number
- numtoken
- literal
- string

These are printed in all caps in the grammar and are defined thus:

```
NAME        ::= letter                      // Cf. 55
              | NAME || letter
              | NAME || digit
              | NAME || othernamech
NUMBER      ::= digit                       // cf. 56
              | NUMBER || digit
NUMTOKEN    ::= digit || letter             // cf. 58
              | digit || othernamech
              | NUMTOKEN || letter
              | NUMTOKEN || digit
              | NUMTOKEN || othernamech
LITERAL     ::= "'" || STRING || "'"        // Cf. 66, 76, 34
              | '"' || STRING || '"'
STRING      ::= /* */
              | STRING || character
character   ::= letter | digit | othercharacter
letter      ::= 'a' | 'b' | 'c' | 'd' ... | 'z'
              | 'A' | 'B' | 'C' | 'D' ... | 'Z'
digit       ::= '0' | '1' | '2' | '3' ... | '9'
othernamech ::= '-' | '.'
whitespace  ::= space | tab | record-end | record-start
space          ::= /* as defined in SGML declaration */
tab            ::= /* as defined in SGML declaration */
record-end     ::= /* as defined in SGML declaration */
record-start   ::= /* as defined in SGML declaration */
othercharacter ::= /* as defined in SGML declaration */
```

This list of primitive token types differs slightly from that of ISO 8879, which defines names, numbers, name tokens, and number tokens as overlapping sets of tokens distinguished by context. The redefinition provided here assigns each string to a single class and thus allows a simpler lexical analyzer. The terms in ISO 8879 correspond to those here in the following way:

- ISO 8879 *name* = NAME
- ISO 8879 *number* = NUMBER
- ISO 8879 *name token* = NAME | NUMBER | NUMTOKEN
- ISO 8879 *number token* = NUMBER | NUMTOKEN

The grammar given above assumes that the lexical scanner will recognize and handle entity references and marked sections. Entity references take one of the following forms (parameter entities within the DTD and within marked section declarations, general entities within document content and attribute values):

```
entityref ::= '&' NAME erc
            | '&' '(' namegrp ')' erc
peref     ::= '%' NAME erc
            | '%' '(' namegrp ')' erc
/*
** The name groups are used only if the SGML declaration
** specifies CONCUR YES.  The TEI Interchange Format
** specifies that entities should never be defined
** differently in different DTDs, so the name group form
** is strictly speaking unnecessary for that format.
```

```
*/
erc       ::= ';'
            | '&#RE;'                       // i.e. record-end
            | /* nothing */
/*
** If the entity reference is not ended explicitly by a
** semicolon or end of line, it is ended implicitly by the
** first non-name character.  Unlike the semicolon or
** record-end, this non-name character counts as data and is
** passed to the application as data.
*/
```

The processing of an entity reference may involve scanning its replacement text for delimiters, passing its content to the parser without scanning for delimiters, opening a new file if the entity is external to the SGML document, and other special processing not described here.

Marked sections take the following forms (n.b. the marked section keywords may be replaced by parameter entity references).

```
/* Marked Sections */                       // Cf. 97-100
msignore  ::= '<![' ignore   '[' anything  ']]' || '>'
mscdata   ::= '<![' kwcdata  '[' chardata  ']]' || '>'
msrcdata  ::= '<![' kwrcdata '[' rchardata ']]' || '>'
msinclude ::= '<![' include  '[' scandata  ']]' || '>'

/* Marked Section keywords */
ignore    ::= kwcdata 'IGNORE' kwcdata
            | ignore  'IGNORE' kwcdata
kwcdata   ::= kwrcdata 'CDATA' kwrcdata
            | kwcdata  'CDATA' kwrcdata
kwrcdata  ::= include  'CDATA' include
            | kwrcdata 'CDATA' include
include   ::= /* nothing */
            | include 'INCLUDE'
            | include 'TEMP'
/*
** Multiple keywords may appear; rank order is IGNORE,
** CDATA, RCDATA, INCLUDE.  TEMP is also legal but has
** no effect.
*/

chardata  ::= /* characters returned to parser as
                 character data, regardless of what
                 delimiters are present */
rchardata ::= /* characters scanned for entity references,
                 and then returned to parser as character
                 data, regardless of what other
                 delimiters are present */
scandata  ::= /* characters scanned and returned to parser
                 as normal */
anything  ::= /* characters scanned only for ']]' || '>' and not
                 returned to parser.  */
```

42.8 Differences from ISO 8879

This grammar assumes the reference concrete syntax; if an alternate concrete syntax is used, some literal strings given in the DTD and document-instance grammars would need to be replaced accordingly.

White space, entity reference, entity end, and comments within markup declarations are assumed to be handled by the lexical scanner and are omitted from this grammar. This shortens and simplifies the grammar somewhat.

The grammar is written as a Backus-Naur-Form (BNF) grammar rather than a regular-right-part grammar; some additional constructs have thus been introduced to deal with repeating and optional items in the original grammar. Nonterminals optional in the original may be required here, and vice versa, depending on how the optionality of a construct has been expressed; in no case does such a change actually affect the set of strings accepted by the grammar.

Some constructs are omitted entirely because they do not occur in the subset of SGML prescribed for use in the TEI Interchange Format:

- link type declaration
- short reference set
- ranked elements and ranked groups
- data tag group
- minimized start-tags
- formal identifiers

Finally, the recognition and expansion of entity references and the handling of marked sections, CDATA and RCDATA elements, and CDATA, SDATA, or NDATA entities have been ignored in the current version. Entity references are assumed to be handled by the lexical scanner, though in a fully conformant SGML parser they are in part dependent on the state of the syntactic parser. CDATA, RCDATA, SDATA, and NDATA elements or entities, like marked sections, are assumed either not to occur or to be handled by the lexical scanner.

Usability Report and Comment

If you have comments of any kind on the current draft of the TEI Guidelines, please complete this form and return it to one of the following addresses:

C. M. Sperberg-McQueen, Computer Center (M/C 135), University of Illinois at Chicago, 1940 W. Taylor Room 124, Chicago IL 60612-7352, USA. Bitnet: U35395@UICVM, Internet: U35395@uicvm.uic.edu

Lou Burnard, Oxford University Computing Service, 13 Banbury Road, Oxford OX2 6NN, UK. Fax: +44 (865) 273275, Janet: LOU@UK.AC.OXFORD.VAX Internet: LOU@vax.oxford.ac.uk

Further copies of this form may be downloaded from LISTSERV @ UICVM (or listserv @ uicvm.uic.edu). Send a mail message containing the line 'GET TEIURC MEMO TEI-L'

Your name:
Your postal address:

Your e-mail address (if any):
Your occupation:
Your academic background:
Your immediate reactions to the Guidelines (please tick or cross one only):

Relevance to your present concerns or interests: high medium low
Importance to your future research plans: high medium low
Comprehensibility/usability of current draft: high medium low

Detailed technical comment: [Please feel free to comment on any part of the Guidelines or in general. Comments relating to specific parts of the text are most helpful if they include a section or chapter number]

Thank you!

Temporary Index