25 Writing System Declaration

This chapter may be substantially revised or withdrawn in the next edition of these Guidelines

The *writing system declaration* or WSD is an auxiliary document which provides information on the methods used to transcribe portions of text in a particular language and script. We use the term *writing system* to mean a given method of representing a particular language, in a particular script or alphabet; the WSD specifies one method of representing a given writing system in electronic form. A single WSD thus links three distinct objects:

- the language in question
- the writing system (script, alphabet, syllabary) used to write the language
- the coded character set, entity names, or transliteration scheme used to represent the graphic characters of the writing system

Different natural languages thus have different writing system declarations, even if they use the same script. Different methods used to write the same language (e.g. Cyrillic or Latin encoding of Serbo-Croatian), and different methods of representing the same script in electronic form (e.g. different coded character sets such as ASCII or EBCDIC, or different transliteration schemes) similarly must use different writing system declarations.

This chapter describes first the overall structure of the WSD (section 25.1 *Overall Structure of Writing System Declaration*), and then the specific elements used to document the natural language, writing system, coded character sets, entity names, and transliteration schemes united by the WSD. Section 25.6 *Linkage between WSD and Main Document* describes how the WSD is associated with different portions of a document. Predefined TEI writing system declarations, which should suffice for many uses, are described in section 25.7 *Predefined TEI WSDs*. There follows a brief description of how to create a new WSD on the basis of an existing WSD. Finally, in section 25.8 *Details of WSD Semantics* we provide a formal discussion of the semantics of the writing system declaration.

25.1 Overall Structure of Writing System Declaration

A writing system declaration is a distinct auxiliary document, separate from any transcription for which it is used. A TEI document specifies the writing system declarations applicable to it by means of declarations in its header, and by means of its use of the global lang attribute, as further specified in section 25.6 *Linkage between WSD and Main Document*. Each writing system declaration is itself a free-standing document, encoded as a single `<writingSystemDeclaration>` element, and containing the following elements:

<writingSystemDeclaration> declares the coded character set, transliteration scheme, or entity set used to transcribe a given writing system of a given language. Attributes include:

name gives a formal name for the writing system declaration
Values any string of characters

date gives the date on which the writing system declaration was last revised.
Values A date in valid ISO format.

<language> identifies the language being described in the writing system declaration. Attributes include:

iso639 gives the two-letter standard language code from ISO 639:1988 (or its revised version ISO 639-1 when that becomes standard), or a three-letter code from ISO 639-2: 1998.
Values any two- or three-letter code included in ISO 639; if the language is not included in the list in ISO 639, the value should be given as the empty string.

<script> contains a prose description of the script declared by a writing system declaration.

<direction> specifies one or more conventional directions in which a language is written using a given script. Attributes include:

chars (characters) indicates the order in which characters within a line are conventionally presented in this writing system
Suggested values include:

LR left to right
RL right to left
TB top to bottom
BT bottom to top

lines indicates the order in which lines conventionally follow each other in this writing system.

Suggested values include:

TB top to bottom
BT bottom to top
LR left to right
RL right to left

<characters> contains a specification of the characters used in a particular writing system to write a particular language, and of how those characters are represented in electronic form.

<note> (in a writing system) contains a note of any type.

All elements in the writing system declaration may bear either of the following two attributes:

id gives a unique identifier for the element.

lang gives the language in which the content of the element is written.

These attributes function in the same way as the global id and lang attributes of the main TEI DTD (although for technical reasons the latter is declared differently): the former provides a unique identifier for the element, and the latter identifies the language in which the contents of the element are expressed, using a code from ISO 639.

The overall structure of a writing system declaration is thus as follows:

```
<writingSystemDeclaration lang='eng' name='...' date='1993-05-29'>
<language iso639='...'>
    <!-- name of language here -->
    </language>
<script>
    <!-- description of script here ... -->
    </script>
<direction chars='LR' lines='TB'/>
<characters>
    <!-- description of character inventory here ... -->
    </characters>
</writingSystemDeclaration>
```

The attributes date and name are required on the <writingSystemDeclaration> element. The date attribute is used to specify the date on which the WSD was written or last changed; this must be given in the format yyyy-mm-dd. (As defined by ISO 8601: 2000(E), *Data elements and interchange formats — Information interchange — Representation of dates and times*, section 5.2.1.1, extended format.)

The name attribute is used to assign a formal name to the writing system declaration, for references to it from elsewhere. It is recommended that the name be constructed as a *formal public identifier*, as described above (2.7 *Entities*); for purposes of writing system declarations, this means it should follow the pattern of the following examples:

```
-//TEI P2: 1993//NOTATION WSD for Modern English//EN
-//WWP 1993//NOTATION WSD for 17th-century English//EN
-//OTA 1991//NOTATION WSD for Old English//EN
-//GLDV 1997//NOTATION WSD Mittelhochdeutsch//DE
```

The other elements of the WSD are described in the following sections.

The DTD for writing system declarations is included in file teiwsd2.dtd; a writing system declaration will thus begin with a document type declaration invoking that file:

```
<!DOCTYPE writingSystemDeclaration
           PUBLIC "-//TEI P4//DTD Auxiliary Document Type:
           Writing System Declaration//EN"
          "teiwsd2.dtd" >
```

The formal declaration of the writing system declaration is as follows:

```
<!-- 25.1: Writing System Declaration-->
<!--Text Encoding Initiative Consortium:
Guidelines for Electronic Text Encoding and Interchange.
Document TEI P4, 2002.
Copyright (c) 2002 TEI Consortium. Permission to copy in any form
is granted, provided this notice is included in all copies.
These materials may not be altered; modifications to these DTDs should
be performed only as specified by the Guidelines, for example in the
chapter entitled 'Modifying the TEI DTD'
These materials are subject to revision by the TEI Consortium. Current versions
are available from the Consortium website at http://www.tei-c.org-->
<!ENTITY % INHERITED '#IMPLIED' >
<!ENTITY % ISO-date 'CDATA' >
<!--Embed entities for TEI generic identifiers.-->
<!ENTITY % TEI.elementNames PUBLIC '-//TEI P4//ENTITIES Generic
Identifiers//EN' 'teigis2.ent' >%TEI.elementNames;
<!--Insert switch for XML/SGML -->
<!ENTITY % TEI.XML 'IGNORE' >
<![%TEI.XML;[
<!ENTITY % om.RO '' >
<!ENTITY % om.RR '' >
]]>
<!ENTITY % om.RO '- O' >
<!ENTITY % om.RR '- -' >
<!ENTITY % a.global '
      id ID #IMPLIED
      lang CDATA %INHERITED;'>
<!ELEMENT writingSystemDeclaration %om.RR;  (language, script,
direction*, characters, note*)>
<!ATTLIST writingSystemDeclaration
      %a.global;
      name CDATA #REQUIRED
      date %ISO-date; #REQUIRED
      TEIform CDATA 'writingSystemDeclaration'  >
<!--declarations from 25.2: Language identification inserted here -->
<!--declarations from 25.3: Script and writing direction inserted here -->
<!--declarations from 25.4.1: Base components inserted here -->
<!--declarations from 25.4.2: Exceptions to the base components inserted here -->
<!--declarations from 25.5: Notes inserted here -->
<!-- end of 25.1-->
```

25.2 Identifying the Language

The <language> element is used to name the language associated with the WSD. Its iso639 attribute gives the ISO standard code for the language as defined by *ISO 639: 1988. Code for the representation of names of languages*, or its successor standards.

<language> identifies the language being described in the writing system declaration. Attributes include:

iso639 gives the two-letter standard language code from ISO 639:1988 (or its revised version ISO 639-1 when that becomes standard), or a three-letter code from ISO 639-2: 1998.

Values any two- or three-letter code included in ISO 639; if the language is not included in the list in ISO 639, the value should be given as the empty string.

If the language in question is not included in the list in ISO 639, the value of the attribute iso639 should be the empty string, as in the following example:

```
<language iso639=''>Various</language>
```

The <language> element should not be confused with the global lang attribute; the element identifies the language whose writing system is being documented, while the attribute identifies the language in which the description is being written. A writing system declaration for classical Greek, for example, which itself is written in English, would have the value eng for the lang attribute on the top-level element, and the value grc for the iso639 attribute on the <language> element:

```
<writingSystemDeclaration
          id='GRC.beta'
          lang='eng'
          name='-//TEI P2: 1993//NOTATION WSD for TLG Beta Code
               transliteration of ancient greek//EN'
          date='1993-05-29'>
<language iso639='grc'>Classical Greek. This WSD documents the Beta
  transcription code for classical Greek developed by the Thesaurus
  Linguae Graecae of the University of California, Irvine.</language>
<!-- ... -->
</writingSystemDeclaration>
```

Normally, the language described is a natural language; in some cases, however, artificial languages, dialects, or other sublanguages may be usefully regarded as a language and documented in a writing system declaration. When a sublanguage is documented, a description of the sublanguage should be included in the <language> element:

```
<language iso639='jpn'>
   Japanese (specialized writing system for waka)
</language>
```

When a writing system declaration is prepared solely in order to document a coded character set or entity set suitable for use with many natural languages, the content of the <language> element should be "Various" (or the equivalent in the language of the WSD):

```
<writingSystemDeclaration
          lang='fra'
          name='-//TEI P2: 1993//NOTATION WSD for ISO 646 IRV//FR'
          date='1993-05-29'>
  <language iso639=''>Plusieurs</language>
  <!-- ... -->
</writingSystemDeclaration>
```

The <language> element is formally defined thus:

```
<!-- 25.2: Language identification-->
<!ELEMENT language %om.RO;  (#PCDATA)>
<!ATTLIST language
      %a.global;
      iso639 CDATA #REQUIRED
      TEIform CDATA 'language'  >
<!-- end of 25.2-->
```

25.3 Describing the Writing System

The writing system itself is described in general terms using the following elements:

<script> contains a prose description of the script declared by a writing system declaration.

<direction> specifies one or more conventional directions in which a language is written using a given script. Attributes include:

chars (characters) indicates the order in which characters within a line are conventionally presented in this writing system

Suggested values include:

- LR left to right
- RL right to left
- TB top to bottom
- BT bottom to top

lines indicates the order in which lines conventionally follow each other in this writing system.

Suggested values include:

- TB top to bottom
- BT bottom to top
- LR left to right
- RL right to left

The <script> element contains a prose description of the script, alphabet, syllabary, or other system of writing used to write the language in question. The <direction> element indicates the direction(s) in which the script is conventionally written. Both these elements are provided for the sake of human readers; neither is likely to be suited to machine processing without human intervention.

The Latin alphabet conventionally used to write English, for example, might be described thus:

```
<script>Latin alphabet (with diacritics for loan words)</script>
<direction chars='LR' lines='TB'/>
```

The chars and lines attributes are used to indicate the direction in which characters within a line, and lines on the page, may legitimately be written using the script in question. If more than one direction is possible, the <direction> element may repeat or its attributes may be given more complex values. A script written vertically top to bottom, with lines arranged either left to right or right to left, for example, might be declared either thus:

```
<direction chars='TB' lines='LR'/>
<direction chars='TB' lines='RL'/>
```

or thus:

```
<direction chars='TB' lines='LR RL'/>
```

In very complex cases, the attributes may be given prose values:

```
<direction chars='boustrophedon:  LR, then RL, then LR, etc.' lines='TB'/>
```

or the element may be omitted entirely, in which case experts on the script should be consulted for advice on proper processing.

It should be noted that the <direction> element describes conventional display only: all scripts are subject to unusual treatment for aesthetic or other reasons, and such unusual treatment need not be foreseen here. (The Latin alphabet, for example, although conventionally written left-to-right, top-to-bottom, can be set vertically in signs or in other special cases.) Unusual methods of arranging the text on a page are best documented within the document instance by means of the global rend attribute.

The <script> and <direction> elements are declared thus:

```
<!-- 25.3: Script and writing direction-->
<!ELEMENT script %om.RO;  (#PCDATA)>
<!ATTLIST script
      %a.global;
      TEIform CDATA 'script'  >
<!ELEMENT direction %om.RO;  EMPTY>
<!ATTLIST direction
      %a.global;
      chars CDATA #REQUIRED
      lines CDATA #REQUIRED
      TEIform CDATA 'direction'  >
<!-- end of 25.3-->
```

25.4 Documenting the Character Set and Its Encoding

25.4.1 Base Components of the WSD

The characters or graphic symbols of the writing system are documented in the <characters> element of the WSD. This documentation can take any of the following forms:

- reference to an international standard, national standard, or private coded character set
- reference to a public set of entities
- reference to another WSD which documents the same script and the same methods of representing it electronically
- formal declaration of each graphic unit in the writing system
- a combination of the above: reference to one or more standard coded character sets, entity sets, or writing system declarations, followed by individual declaration of all exceptions

The coded character sets, entity sets, and external WSDs referred to are called the *base components* of the writing system declaration. The base components of a WSD are declared within the <characters> element using the following elements:

<characters> contains a specification of the characters used in a particular writing system to write a particular language, and of how those characters are represented in electronic form.

<codedCharSet> identifies a public or private coded character set which is used as a basic component of a writing system declaration.

<baseWsd> identifies a writing system declaration whose mappings among characters, forms, entity names, and bit patterns are to be incorporated (possibly with modifications) in this writing system declaration.

<entitySet> identifies a public or private entity set whose mappings between entity names and characters are to be incorporated (perhaps with modifications) into this writing system declaration.

The elements <codedCharSet>, <baseWsd>, and <entitySet> are all members of the class *baseStandard* and inherit from it the following attributes:

name gives the normal citation form for the standard being referred to.

authority indicates the authority responsible for issuing the standard being referred to: the TEI, the International Organization for Standardization (ISO), a national body, or a private body. Legal values are:

- tei the base writing system declaration is a standard WSD issued by the Text Encoding Initiative
- iso the character set or entity set was issued by ISO
- national the character set or entity set was issued by a national standards body
- private the writing system declaration, character set, or entity set was issued publicly by a private organization or project
- none the writing system declaration, character set, or entity set has not been publicly issued by any organization; it is specific to an individual text or project

Some simple examples of the use of these elements follow:

```
<codedCharSet name='ANSI X3.4' authority='national'/>
<codedCharSet name='ISO 646: 1991' authority='iso'/>
<baseWsd      name='-//TEI P4: 2001//WSD ISO 8859-1//EN'
              authority='tei'/>
<entitySet    name='ISO 8879:1986//ENTITIES Added Latin 1//EN'
              authority='iso'/>
```

The base components identify the set of characters used in the writing system, and further specify, for each character, the string(s) of bytes and entity names used to encode it in the text. This information may be modified by further information given within the <exceptions> element, as described below in section 25.4.2 *Exceptions in the WSD*.

The elements for identifying the base components of the writing system declaration are declared thus:

```
<!-- 25.4.1: Base components-->
<!ELEMENT characters %om.RO;  ( codedCharSet*, baseWsd*,
                                entitySet*, exceptions? ) >
<!ATTLIST characters
      %a.global;
      TEIform CDATA 'characters'  >
<!ENTITY % a.baseStandard '
      name CDATA #REQUIRED
      authority (tei | iso | national | private | none) #REQUIRED'>
<!ELEMENT codedCharSet %om.RO;  EMPTY>
<!ATTLIST codedCharSet
      %a.global;
      %a.baseStandard;
      TEIform CDATA 'codedCharSet'  >
<!ELEMENT baseWsd %om.RO;  EMPTY>
<!ATTLIST baseWsd
      %a.global;
      %a.baseStandard;
```

```
      TEIform CDATA 'baseWsd'  >
<!ELEMENT entitySet %om.RO;  EMPTY>
<!ATTLIST entitySet
      %a.global;
      %a.baseStandard;
      TEIform CDATA 'entitySet'  >
<!-- end of 25.4.1-->
```

25.4.2 Exceptions in the WSD

The `<exceptions>` element contains definitions for any character which differs in any respect from the specifications contained in the base components of the WSD. If no base components are named, then every character in the writing system must be defined explicitly.

The documentation for each character in the writing system indicates at least the following:

- the string of bytes used to represent the character
- whether the character is a letter, a punctuation mark, a diacritical mark, or falls into some other class
- a brief conventional name or description of the character
- any standard or local entity names used for the character
- the position of the character in the Universal Character Set (UCS) defined by ISO 10646, if known

In addition, images of the character encoded in a graphics format or other notation may be associated with the character as internal or external figures. This information is encoded using the following elements:

<exceptions> documents ways in which a writing system declaration differs from the coded character sets, base writing system declarations, and entity sets which form its bases.

<character> defines one unit in a writing system, supplementing or overriding information provided in the base coded character sets, writing system declarations, and entity sets. Attributes include:

class describes the function of the character using a prescribed classification.

Legal values are:

`lexical` character is used in writing words (lexical items) of the language (includes members of syllabaries and ideographic systems, as well as composite letter-plus-diacritic combinations)

`punc` character is a punctuation mark which does not appear within lexical items

`lexpunc` character can appear as a normal punctuation mark, but can also appear within a lexical item (and should usually, when occurring between two lexical characters, be treated as lexical—in English, hyphen and apostrophe are typically treated as members of this class)

`digit` character is an Arabic decimal numeral (0, 1, ... 9) (does not include superscript numbers, circled numbers, numeric dingbats, etc.)

`space` character represents some form of white space (space character, horizontal or vertical tab, newline, etc.)

`dl` character is a diacritic applying to the following lexical character

`ld` character is a diacritic applying to the preceding lexical character

`dia` character is a diacritic which is explicitly joined to a lexical character by a joiner character

`joiner` character is used to join a diacritic to the lexical character to which it applies (in some encoding schemes, the backspace control character may be used as a joiner; in others, a graphic character is used for the same function)

`other` character does not fall into any of the other classes (dingbats and other unusual characters fall here)

<desc> (in a writing system declaration) contains a description of a character or character form.

<form> identifies one letter form taken by a particular character in a writing system declaration. Attributes include:

string gives the byte string used to encode the letter form in the text.

Values any string of characters (often a single byte)

codedCharSet (coded character set) specifies which base coded character set the string value occurs in.

Values a reference to the identifier of a <codedCharSet> element in the current writing system declaration.

entityStd (standard entity name) gives the name of one or more entities defined for this character form in some standard entity set(s).

Values One or more valid SGML entity names declared in the document type definition of the WSD; the entity must also be included in an entity set mentioned in an <entitySet> declaration in the current writing system declaration or in some base writing system referred to by a <baseWsd> element.

entityLoc (local entity name) gives one or more entity names used locally for this character form.

Values One or more valid SGML entity names declared in the document type definition of the WSD; the entity must also be included in an entity set mentioned in an <entitySet> declaration in the current writing system declaration or in some base writing system referred to by a <baseWsd> element.

ucs-4 (universal-character-set code) gives the position of the character form in the thirty-two bit 'universal character set' defined by ISO 10646.

Values one or more sets of two or four two-digit hexadecimal numbers giving a valid ISO 10646 code point for the character form; for legibility the two-digit hexadecimal numbers should be separated by hyphens. If more than one UCS-4 code is associated with a given character form, the two UCS-4 codes should be given separated by blanks. If the character form is associated with a sequence of UCS-4 codes (e.g. a base character followed by one or more non-spacing diacritics), then the components of the sequence should be separated by +.

<figure> (in a writing system declaration) contains an image of a character form, stored in-line in some declared notation. Attributes include:

notation identifies the notation in which the figure is encoded.

Values a valid name associated with a given notation by means of an NOTATION declaration in the document type definition.

<extFigure> (in a writing system declaration) refers to a figure or illustration depicting the character form, which is stored in some declared notation external to the text. Attributes include:

notation identifies the notation in which the figure is stored.

Values a valid name associated with a given notation by means of a NOTATION declaration in the document type definition.

entity gives the name of an external entity which contains the figure.

Values a valid name associated with the external entity by means of an ENTITY declaration in the document type declaration .

The <exceptions> element contains a series of <character> elements only, each of which may contain descriptions of the character (including its name), notes, and a series of <form> elements documenting the different forms the character can take. Attributes on the <character> and <form> elements are used to convey the information mentioned above: byte string, entity names, UCS-4 code, etc.

A simple example:

```
<character class='lexical'>
  <form string='A' ucs-4='0041'>
    <desc>Latin capital letter A</desc>
  </form>
</character>
```

When transliteration schemes are used, the string used to encode the character will typically be in a different alphabet:

```
<character class='lexical'>
  <form string='*G' entityStd="Ggr" ucs-4='0393'>
    <desc>Greek capital letter Gamma</desc>
  </form>
</character>
```

The UCS-4 code is given as eight hexadecimal digits, one for each four bits of the thirty-two-bit value. For legibility a hyphen may be inserted as a separator after the fourth hexadecimal digit: 00000308 has the same meaning as 0000-0308. Since in almost all cases at present the leading sixteen bits are zero, however, by convention the leading four hexadecimal zeros may be dropped entirely: the value 0308 is identical in meaning to the value 0000-0308.

In some cases, the character is represented not as a single UCS character but as a sequence of such characters; in this case, each thirty-two-bit value except the last must be followed by a plus sign:

```
<character class='lexical'>
  <form string='*=+U' entityStd="Ucdgr" ucs-4='03A5+0302+0308'>
    <desc>Greek capital letter Upsilon with
          circumflex and diaresis</desc>
  </form>
</character>
```

If a given <character> element has more than one encoding using ISO 10646 (e.g. both as “a-umlaut” and as “a” plus “umlaut”), then both encodings may be given, separated by blanks:

```
<character class='lexical'>
  <form string=''
        entityStd="Auml"
        ucs-4='0041+0308 00C4' >
    <desc>Latin capital letter A with umlaut</desc>
  </form>
</character>
```

In most cases, identifying the character or character form by means of its UCS-4 code will suffice to identify the character for all later users of the WSD. In some cases, however, further information must be provided. This may be provided in a <note> attached to the <character> or <form> element:

```
<character class='lexical'>
  <form string='N'
        entityLoc="nn"
        ucs-4="0274"   >
    <desc>Standard ms symbol for double n.</desc>
  </form>
  <note>This character has the form of a capital-letter N,
        but is written the same height as a lower-case N.
        Its appearance is thus that of UCS-4 0274, but it
        does not have the same semantics.</note>
</character>
```

In some cases, it will be necessary or useful to provide an image of the character in question, or to refer to a standard reference work for such an image. The following <character> element might be used to describe, for example, a common Old French abbreviation for “est”, for which the local entity est has been defined:

```
<character class='lexical'>
  <form string='' entityLoc="est">
     <desc>Old French abbreviation for 'est':  lowercase
           'e' with a tilde or macron above.</desc>
     <note>For an image of this character, see
           Cappelli, p. 113, column 1, line 4
           (leftmost and rightmost item).</note>
  </form>
</character>
```

Here, “Cappelli” is the name of a standard reference work which may be consulted to see what the character in question looks like.[165]

Where recourse to reference works is impossible, a picture of the character may be encoded using any standard graphics format, and associated with the character by standard SGML techniques. The SGML document must then have:

- a notation declaration for the graphics format used
- an external entity declaration for the file containing the image
- an <extFigure> element to name the notation and the entity

For a discussion of graphic images and of the declaration of notations, see chapter 22 *Tables, Formulae, and Graphics*. If the Old French abbreviation is encoded using CGM (Computer Graphics Metafile) format in a file called est.cgm, then it may be associated with the appropriate character declaration as follows. In the DTD subset of the WSD, the following declarations are required:

```
<!NOTATION cgm PUBLIC 'ISO 8632/2//NOTATION
                       Computer Graphics Metafile Character encoding//EN'>
<!ENTITY estFigure SYSTEM 'est.cgm' NDATA cgm>
```

In the body of the WSD itself:

```
<character class='lexical'>
  <form string=''
        entityLoc="est">
            <desc>Old French abbreviation for 'est': lowercase
                  'e' with a tilde or macron above.</desc>
            <extFigure notation='cgm' entity='estFigure'/>
            <note>For an image of this character, see
                  Cappelli, p. 110, column 1, line 4
                  (leftmost and rightmost item).</note>
  </form>
</character>
```

Despite now having a picture of the character, we retain the prose description and reference to Cappelli, for the sake of those without ready access to the appropriate graphics processors.

The <exceptions> element and its contents are declared thus:

```
<!-- 25.4.2: Exceptions to the base components-->
<!ELEMENT exceptions %om.RO;  (character*)>
<!ATTLIST exceptions
      %a.global;
      TEIform CDATA 'exceptions'  >
<!ELEMENT character %om.RO;  (desc*, form+, note*)>
<!ATTLIST character
      %a.global;
      class (lexical | punc | lexpunc | digit | space | DL | LD | dia | joiner | other)
"lexical"
      TEIform CDATA 'character'  >
<!ELEMENT desc %om.RO;  (#PCDATA)>
<!ATTLIST desc
      %a.global;
      TEIform CDATA 'desc'  >
<!ELEMENT form %om.RO;  (desc+, (figure | extFigure)*, note*)>
<!ATTLIST form
      %a.global;
      string CDATA #IMPLIED
      codedCharSet IDREF #IMPLIED
      entityStd ENTITIES #IMPLIED
      entityLoc ENTITIES #IMPLIED
      ucs-4 CDATA #IMPLIED
      TEIform CDATA 'form'  >
```

[165] *Dizionario di Abbreviature latine ed italiane* per cura di Adriano Cappelli, 6th ed. (Milan: Ulrico Hoepli, 1979). This work on Latin abbreviations might be less convenient for the purpose than one concentrating on Old French, but it is more widely used than any other.

```
<!ELEMENT figure %om.RR;  (#PCDATA)>
<!ATTLIST figure
      %a.global;
      notation CDATA #REQUIRED
      TEIform CDATA 'figure'  >
<!ELEMENT extFigure %om.RO;  EMPTY>
<!ATTLIST extFigure
      %a.global;
      notation CDATA #REQUIRED
      entity CDATA #REQUIRED
      TEIform CDATA 'extFigure'  >
<!-- end of 25.4.2-->
```

25.4.3 Documenting Coded Character Sets and Entity Sets

Public or private coded character sets and entity sets may be usefully documented using WSDs; the WSD will make explicit some information (such as the UCS-4 code) not normally given explicitly in character set standards or public entity sets. The coded character set or entity set being documented should be included by means of a <codedCharSet> or <entitySet> element; the <exceptions> element should include one <character> element for each character included in the character set or the entity set. Deciding whether to treat two entities or two bit patterns as separate characters or as forms of the same character will require knowledge of the script involved, and different encoders may reach different decisions. In cases of doubt, though, it is usually acceptable practice to treat each bit pattern in a coded character set, and each entity in an entity set, as a distinct character.

A non-standard local coded character set (e.g. an EBCDIC character set) may be documented in a WSD by defining one <character> element for each printable code point in the character set, adding the names of standard (and local) entities and UCS-4 codes as appropriate. Since this extra information is useful in packing documents for interchange, and in processing pattern arguments in the TEI extended-pointer syntax described in section 14.2 *Extended Pointers*, those responsible for a local installation are strongly encouraged to document the local system character set in a WSD, if it is not already so documented.

25.4.4 Documenting Transliteration Schemes

When a script is encoded not in a character set designed for it, but in one designed for another script, (e.g. Greek encoded using the Latin alphabet), a transliteration scheme is necessary. In documenting such a transliteration scheme, the coded character set actually in use should be named as a base component. An <exceptions> element can then be used to override the normal meaning of the individual byte strings used in the transliteration. For example, the following <character> element overrides the usual association of the byte representing A with the Latin letter A and substitutes instead an association with the Greek letter alpha:

```
<character class='lexical'>
  <form string='A' entityStd="agr" ucs-4='03B1'>
    <desc>Greek small letter alpha</desc>
  </form>
</character>
```

Care should be taken in choosing or developing transliteration schemes to ensure that they are unambiguously reversible.

25.5 Notes in the WSD

Notes on the WSD, individual characters, or individual character forms may be included in the <note> element at the appropriate level.

<note> (in a writing system) contains a note of any type.

Unlike its counterpart in the main TEI DTD, the <note> element within the writing system declaration may contain no paragraphs and no phrase-level elements: only character data. It is formally declared thus:

```
<!-- 25.5: Notes-->
<!ELEMENT note %om.RO;  (#PCDATA)>
<!ATTLIST note
      %a.global;
      TEIform CDATA 'note'  >
<!-- end of 25.5-->
```

25.6 Linkage between WSD and Main Document

The writing system declaration is associated with different portions of a main document by means of the global lang attribute. This attribute is defined as an IDREF and its value must be the identifier on a <language> element within the TEI header of the main document. The <language> element in turn provides, in its wsd attribute, the name of the entity (which usually resolves to an external file) containing the writing system declaration associated with that lang value. For a more detailed account of this process, compare the discussion in section 26.1 *Linking a TEI Text to Feature System Declarations*.

A default writing system declaration may be associated with any TEI document by supplying a value for the lang attribute on the outermost element (<TEI.2> or <teiCorpus.2>). This lang attribute is required to point at a <language> element in the TEI header, which in turn is required to indicate an entity containing the writing system declaration associated with that language.

The following schematic shows how this can be achieved:

```
<!DOCTYPE TEI.2 PUBLIC "-//TEI P4//DTD Main Document Type//EN" "tei2.dtd" [
<!ENTITY % TEI.prose "INCLUDE">
<!ENTITY % TEI.XML "INCLUDE">
<!NOTATION wsd PUBLIC '-//TEI P3-1994//NOTATION Writing System Declaration//EN'>
<!ENTITY myWSD SYSTEM "myWSD.xml" NDATA wsd>
]>
<TEI.2>
<teiHeader>
<!-- ... -->
<language id="GRG" wsd="myWSD">
<!-- ... -->
</teiHeader>
<text lang="GRG">
<!-- ... -->
</text>
</TEI.2>
```

This example begins by including the TEI prose tagset in its XML version. This is followed by a notation declaration for the WSD notation itself, and the declaration of an unparsed XML entity called myWSD which is resolved to the SYSTEM file myWSD.xml and which uses the WSD notation. The notation itself must be declared in order that it may be referenced on the subsequent entity declaration, (or declarations, if more than one writing system is in use). All these declarations are located in the DTD subset. In the document proper, the Header contains a <language> element, with the identifier GRG, which references the WSD entity by means of the entity name myWSD. The <text> element supplies the identifier of that language on its lang attribute to indicate that, by default, all the component elements of the document use that language and hence also that Writing System.

25.7 Predefined TEI WSDs

The Text Encoding Initiative has defined several writing system declarations to demonstrate the features of the system. These include WSDs for most modern European languages, for common transcription systems such as TLG Beta code, and for the International Phonetic Alphabet.

A list of the Writing System Declarations released with the current version of these Guidelines is given below in chapter 37 *Obtaining TEI WSDs*

The standard TEI writing system declarations are expected to meet the needs of many encoders; some, however, will need to prepare new WSDs to describe character-encoding schemes not included in the standard WSDs.

25.8 Details of WSD Semantics

This section describes the meaning of the WSD in more formal terms than have been used elsewhere in this chapter; it can be skipped by most readers, but should be read carefully by those who wish to write complex writing system declarations or to implement software to process writing system declarations or to interpret them in the processing of TEI-conformant documents.

25.8.1 WSD Semantics: General Principles

A writing system declaration provides a complicated bundle of mappings:

- a 1:1 partial function from strings in given coded character sets to character forms
- a function from entity names to character forms, and therefore derivatively a function from entity names to strings
- a function from character forms to characters, and therefore derivatively:
  - a function from strings to characters
  - a function from entity names to characters
- a relation between UCS-4 codes and character forms
- a function from UCS-4 codes to characters

To ensure that the relations described as functions are in fact functional, the following constraints apply on the WSD:

- No two <form> elements can have the same values for both codedCharSet and string. Since usually there is only one <codedCharSet> used as a basic component, this usually means each string attribute value must be unique in the WSD.
- No two <form> elements can name the same entity in either entityStd or entityLoc. It is legal, though pointless, for both entityStd and entityLoc on the same <form> element to name the same entity.
- More than one <form> element may have the same UCS-4 value, but if so they must be within the same <character> element.

These constraints may be summarized thus: one 'character' (however the creator of the WSD defines a character) can be associated with more than one byte string, entity name, or UCS-4 code, but any single byte string (given a specific coded character set), any single entity name, and any single UCS-4 code must be associated with only one single <character>. One can, for example, associate both "tilde" and "logical not" with a <character> meaning "logical negation", but one cannot associate both a <character> called "tilde" and one called "logical negation" with the ASCII character 7/14: given a 7/14 in the text, it must be unambiguously clear whether the character is a "tilde" or a "logical negation". If one wishes to retain the ambiguity, one must define a <character> called (for example) "logical-not or tilde or swung-dash". Similar restrictions apply to entity names and UCS-4 codes: each must be associated with a single <character> element.

25.8.2 Semantics of WSD Base Components

The effects of naming coded character sets, entity sets, and other WSDs as base components may now be defined thus:

- reference to a coded character set makes available the set of bit-pattern-to-character mappings defined in the coded character set. That is, if a WSD refers to a coded character set, then whenever the WSD is in use, any character in that coded character set may be used with its standard meaning unless it has been redefined using the <exceptions> element. It is recommended that a WSD be provided for each coded character set, to make the mappings fully explicit.
- reference to an entity set makes available the set of entity-name-to-character mappings defined in the entity set. It is assumed that standard public entity sets contain enough information to count as a valid mapping; for private entity sets, the preferred method of providing the necessary information is to define the entity set in a WSD. If for example a WSD refers to the ISO Latin 1 entity set, then whenever that WSD is in use, any entity in that set may be used with its public meaning, unless it has been redefined in the <exceptions> element.
- reference to a WSD makes available the set of mappings declared in that WSD; the language and writing system direction information given in the base WSD is ignored.

If reference is made only to standard character sets and entity sets, there is no mechanical method of associating the 'characters' involved in one mapping with those involved in another. E.g. a reference to ISO 646 IRV provides a map from code point 5/11 to a character one might call "left square bracket". A reference to entity set ISOpub1 provides a map from the entity name lbr to what should probably be considered the same character. There is however no guarantee that any processing software will necessarily be sufficiently intelligent to make this association of mappings automatically; it requires hard-coded knowledge of the specifics of certain character sets and entity sets.

When, however, base WSDs are used to document important entity sets and character sets, it does become possible to define mechanical methods of associating <character> elements in different base components.

25.8.3 Multiple Base Components

When multiple bases of the same type are referred to, the effects are these:

- if more than one coded character set is named, then it is expected that character-set shifting as described in ISO 2022 or some equivalent is in use, and proper shifting is the responsibility of the user. All strings in the WSD must specify the ID of the proper coded-character-set base, using the codedCharSet attribute.
- if more than one entity set is named, then entity names from all named sets may be used as values of the entityStd and entityLoc attributes. If the same name occurs in more than one entity set, the assumption is made that it refers each time to the same character.
- if more than one base WSD is named, then all characters declared in all the WSDs are available. For this case, we can define what happens to merge the different base components more precisely than for the other types of base component:
  - any two <form> elements which name the same entity or the same string in the same coded character set are considered the same form, and are merged as described below in section 25.8.5 *Merger of Form and Character Elements*.
  - any two <form> elements which give the same UCS-4 code are considered forms of the same <character>, and their parent <character> elements are merged. The forms themselves may be merged or may remain distinct: if the forms have conflicting values for any attribute, they must remain distinct; if they don't conflict, they may be merged, at the option of the processing software. In the general case, there might be more than one way to perform mergers, so merger is not required.

The result of invoking multiple base WSDs is thus a merged WSD in which the <form> and <character> elements have been merged as prescribed. If the merger is impossible because the two WSDs are incompatible, a semantic error occurs. A set of WSDs is compatible and may be invoked together if all of the following are true:

- any given entity name is associated with a single string (in a given coded character set) and a single character class
- any given string or UCS-4 code is associated with a single character class

25.8.4 Semantics of Exceptions

We can now define the semantics of the <exceptions> element.

The base components provide a preliminary set of mappings, as described above. For convenience let us call this the *default map*. The <exceptions> element allows the user to modify the default map by defining further mappings and by overriding parts of the default map. There are three cases: a new <character> element replaces an old one, is merged with an old one, or is added to the set without affecting any old ones.

25.8.4.1 Case 1: replacement

If a <form> element within <exceptions> (F-new) 'collides' with a <form> element in the default map (F-old), then the parent <character> element of F-new replaces the parent element of F-old. Two

<form> elements collide if they have the same values for codedCharSet and string. (N.B. if this condition occurs within the default map, the two <form> elements are merged.)

For example, to define the TLG Beta code transliteration of alpha as a we first name ISO 646 IRV as a base component; this has the effect of creating the following (possibly imaginary) <form> element:

```
<character id='A' class='lexical'>
   <form string='a'>
      <desc>lowercase latin letter A</desc>
   </form>
</character>
```

We then include the following within the <exception> element:

```
<character id='ALPHA' class='lexical'>
   <form string='a' entityStd='gkalpha'>
      <desc>lowercase Greek alpha</desc>
   </form>
</character>
```

This overrides the <character> element for latin A, and indicates that in the transliteration scheme documented by this WSD, character 6/01 represents a Greek alpha, no matter what ISO 646 says.

25.8.4.2 Case 2: merger

If a <character> element within <exceptions> 'overlaps' with one in the default map, then the two <character> elements are merged. Two <character> elements overlap if any of their <form> elements name the same entity or UCS-4 code. (N.B. if these conditions occur within the default map, they lead to merger either of the two <form> elements — for entity name overlap — or of the two <character> elements.)

For example: suppose we wish to document the three-Rs transcription described in section 4.2 *Entry and display of characters*. We name ISO 646 IRV as a base character set (or WSD) and add the following exceptions:

```
<exceptions>
    <character id='R' class='lexical'>
        <desc>lowercase latin letter r</desc>
        <form string='' entityLoc='r' ucs-4='0072'>
            <desc>'normal' form, similar to modern print r and to
                  Cappelli, p. 318, line 2, items 3, 6, 15.</desc>
        </form>
        <form string='' entityLoc='r2' ucs-4='0072'>
            <desc>'round' form, usually following 'o', similar
                  to a modern Arabic digit 2 (or to Cappelli, p. 318,
                  line 2, items 13 and 14)</desc>
        </form>
        <form string='' entityLoc='r3' ucs-4='0072'>
            <desc>'small-cap' form, like a capital R but
                  same height as lowercase (cf. Cappelli, p. 318,
                  line 1, items 2 and 3)</desc>
        </form>
    </character>
  </exceptions>
```

As a second example, imagine we wish to document a local entity set for Old English in which we use non-standard short entity names t (for þ or thorn), d (for ð or eth), and a (for æ or æsc). Assuming the TEI has provided a WSD for the Latin 1 entities, the whole WSD is this:

```
<writingSystemDeclaration
     name='-//OTA 1990//NOTATION WSD Old English entities//EN'
     date='1993-05-25'
     lang='eng'>
   <language iso639=''>Various</language>
   <script>Latin alphabet, extended</script>
   <direction lines='TB' chars='LR'/>
   <characters>
   <baseWsd name='-//TEI P4: 2001//NOTATION WSD ISO Added Latin 1//EN'
         authority='tei'/>
```

```
   <exceptions>
   <character class='lexical'>
   <form entityStd='thorn' entityLoc='t'>
      <desc>lowercase latin letter thorn</desc></form></character>
   <character class='lexical'>
   <form entityStd='Thorn' entityLoc='T'>
      <desc>uppercase latin letter thorn</desc></form></character>
   <character class='lexical'>
   <form entityStd='eth' entityLoc='d'>
      <desc>lowercase latin letter eth</desc></form></character>
   <character class='lexical'>
   <form entityStd='Eth' entityLoc='D'>
      <desc>uppercase latin letter eth</desc></form></character>
   <character class='lexical'>
   <form entityStd='aelig' entityLoc='a'>
      <desc>lowercase latin aesc (= digraph aelig)</desc></form></character>
   <character class='lexical'>
   <form entityStd='AElig' entityLoc='A'>
      <desc>uppercase latin aesc (= digraph aelig)</desc></form></character>
   </exceptions>
   </characters>
   <note>This WSD is just to document the local entities; it should be
   named as a base WSD by the actual writing system declaration.
   </note>
</writingSystemDeclaration>
```

This has the effect of merging the <character> elements for thorn, eth, and aesc (or a-e ligature) defined in the ISO Latin 1 WSD with those given here, which specify the local entity name. The <form> elements may or may not be merged, so the software may or may not actually realize that the local entity t corresponds with the UCS-4 code given in the TEI WSD for ISO Latin 1.

The full local WSD can then be this:

```
<writingSystemDeclaration
         name='-//OTA 1993//NOTATION Old English WSD//EN'
         date='1993-05-25'
         lang='eng'>
    <language iso639='ang'>Anglo-Saxon / Old English</language>
    <script>Latin alphabet, extended</script>
    <direction lines='TB' chars='LR'/>
    <characters>
          <baseWsd name='-//TEI P4: 2001//NOTATION WSD ISO 646 IRV//EN'
                   authority='tei'/>
          <baseWsd name='-//OTA 1990//NOTATION
                         WSD Old English entities//EN'
                   authority='private'/>
          <baseWsd name='-//TEI P4: 2001//NOTATION
                         WSD ISO Added Latin 1//EN'
                   authority='tei'/>
    </characters>
  </writingSystemDeclaration>
```

We refer explicitly to ISO Latin 1, for clarity, but in theory it has already been included in -//OTA 1990//WSD Old English entities//EN and need not be repeated. At this time, the rules for merger would force our local <form> elements to be merged with the standard <form> elements, so the local entity t would map correctly into the UCS-4 character set.

25.8.4.3 Case 3: expansion

If a <character> element has no <form> children which collide with anything in the default map, and does not itself overlap with anything in the default map, then it is simply added to the default map.

For example, suppose we wish to document an abbreviation used for Old French “est” in our manuscript, which resembles e with a tilde or macron. Since we expect we may have more abbreviations for “est”, we use the local entity name est1 for this one. Within <exceptions>, we declare the abbreviation thus:

```
<character id='EST1' class='lexical'>
   <form string='' entityLoc='est1'>
```

```
      <desc>abbreviation for 'est', lowercase latin e
            with a tilde or macron above, similar to
            Cappelli p. 113, col 1, items 4(a) and 8.</desc>
   </form>
</character>
```

25.8.5 Merger of Form and Character Elements

In some cases, the <form> and <character> elements introduced notionally by reference to a coded character set or entity set, or introduced explicitly by reference to a base WSD, may be considered as referring to identical objects; this is called *merger*. Two <form> elements F1 and F2 can be merged if they both have the same values for codedCharSet and string, or if codedCharSet and string are unspecified (implied) in at least one. When F1 and F2 are merged, the result is a (possibly imaginary) <form> element (F3) the attributes of which are derived thus:

- if F1 has no value for a codedCharSet, then F3 has the same value for this attribute as does F2. If both F1 and F2 have explicit values, the values must be identical.
- if F1 has an empty string for string, then F3 has the same value as F2. If both have values other than the empty string, they must be identical.
- for entityStd, entityLoc, and ucs-4, F3 gets a value containing all the entity names or codes which appear in the corresponding attribute values of either F1 or F2. I.e. the attribute values are viewed as sets, and F3 gets the union of F1 and F2.

The children of the new element are derived by taking all the <desc> children of F1, then all the <desc> children of F2; all the <figure> children of F1, then those of F2; all the <note> children of F1, then all the <note> children of F2. In other words, all the children of the source elements survive as children of the result element.

Provided that their forms are compatible, two <character> elements C1 and C2 may be merged unless their values for class differ. The resulting <character> element C3 has the same value for its class attribute as C1 and C2, and all the children of C1 and C2 are made children of C3 (<desc> children first, then <form> children).

Note that merger is sometimes required by the semantic rules given above, and sometimes optional. If merger is required but not legal (because the two elements to be merged are incompatible), then a semantic error has occurred and the two base WSDs which give rise to it should not be invoked together.