Computers and the Humanities **33:** 49–57, 1999.

Using the TEI Writing System Declaration (WSD)

DAVID J. BIRNBAUM[1], MAVIS COURNANE[2,3] and PETER FLYNN[2,4]
[1]*Department of Slavic Languages and Literatures, University of Pittsburgh, 1417 Cathedral of Learning, Pittsburgh, PA 15260, USA (E-mail: djb@clover.slavic.pitt.edu);* [2]*Computer Center, Kan Building, University College Cork, Ireland;* [3]*E-mail: cournane@imbolc.ucc.ie;* [4]*E-mail: pflynn@imbolc.ucc.ie*

Abstract. The TEI's WSD mechanism allows text encoders to document the nature and use of language scripts for a given document or class of documents, but these facilities have not been widely implemented.

This paper describes two implementations which use different approaches, both for encoding and for rendering, and draws some conclusions about the need for improving the utility of WSDs for scholarly texts.

Key words: character, Cyrillic, glyph, Greek, Hebrew, Latin, Slavic, Slavonic, TEI, TEX, Text Encoding Initative, writing system declaration, WSD

1. The TEI WSD

According to *P3*, the Text Encoding Initiative (TEI) Writing System Declaration (WSD) is 'an auxiliary document which provides information on the methods used to transcribe portions of text in a particular language and script' (Sperberg-McQueen and Burnard, 1994, section 25). This suggests at least two potential core functions for the WSD:

- to document transcription methods in a human-readable form, so that those who need to render, transform, or otherwise process the document will understand the meaning of the encoded information, and
- to support automated access to such representational information as the character codes and glyph identifiers associated with the characters and character entities in PCDATA content.

The first of these goals, the documentation of transcription systems for the edification of anticipated human readers, is straightforward, and anyone conversant with WSD architecture can examine a document and its accompanying WSDs and work out the mapping employed to document different textual representations. The second goal, however, the use of WSDs in automated transducing, rendering, and other processing operations, is less accessible.

Despite the indisputable documentation and processing value of a formal writing system description, the WSD appears to be one of the least-used features of the TEI guidelines. Thus, although a few WSDs are available as part of the electronic TEI distribution, these pertain primarily to straightforward, normalized, well-documented writing systems, and they apparently exist exclusively for documentation purposes, for we know of no projects other than our own where WSDs of any sort are employed in automated document processing. That the WSD is not in wide use is clear from examining the WSD support files available on the TEI HTTP and FTP servers, where the `teiwsd2.dtd` file includes a declaration for a `wdgis2.ent` system identifier, which is, in fact, misnamed on the servers as `teiwdgis2.dtd`. That this error in the distribution could have gone unnoticed until the present suggests that these files have not been in significant demand by users.

One of the primary problems encountered in designing and implementing a WSD, and especially in using a WSD for processing, is that the documentation in the *TEI Guidelines* is in some respects insufficient. In particular, *P3* provides no practical example of an implemented WSD, just as it provides no information about using a WSD to generate a legible final-form document without sacrificing document portability. Production of final-form documents requires mapping between characters and glyphs, and the fact that most applications still exist in a pre-Unicode universe makes this task all the more problematic.

A second fundamental practical problem with implementing WSDs is that there is no software currently available that can manipulate them automatically. The proper display of Hebrew or Greek or Cyrillic character sets in a character-mode SGML editor such as *Emacs* (*psgml-mode*) is elusive because of the editor's limited facility for character replacement. Some graphical SGML editors do permit the specification of a particular font for the display of particular elements and attributes, and this should, in theory, enable Hebrew and Greek and Cyrillic entities such as Hebrew `&vavhb;`, Greek `&xgr;`, or early Cyrillic `&aos;` to be rendered with the correct fonts. Unfortunately, not all graphical SGML editors provide this facility of entity replacement from attributes.[1] SGML browsers such as SoftQuad's *Panorama Pro* permit entity substitution in the *before* and *after* replications of attribute values, which means that in these browsing environments attribute values may invoke a specific font that produces the desired substitution, but in these cases it is the character entity set declared in the main SGML document, rather than the more powerful Writing System Declaration, that governs that substitution.[2] Alongside these screen-rendering problems, attempts to print an SGML file containing Hebrew or Greek or Cyrillic characters are frustrated by the absence of print-rendering software capable of processing the WSDs unaided.

A WSD-type approach to encoding documents based on orthographically complex writing systems was introduced in an article by David J. Birnbaum in *Computer Standards and Interfaces* 18 (1996). The principal goal of the present paper is to provide a practical complement to that theoretical discussion, illustrating how the authors have used WSDs to support not only the encoding and

documentation, but also the transformation of TEI-conformant documents. One test file is based on transcriptions from several early Cyrillic manuscripts; the other is a multilingual manuscript transcription involving hellenized Hebrew and latinized Greek in a Latin context. The transformation system used for all tests was Omnimark Technology's *Omnimark LE* (version 3), although several different transformation algorithms were applied to the sample input files:

- parse the WSD, build an in-memory replacement table, and access the table to replace information in the SGML source file with information from the WSD;
- parse the WSD, write out a custom SDATA entity set with replacement strings extracted from the WSD, and reparse the SGML source file to effect the replacements;
- parse the WSD, build an external LaTeX replacement file, and let LaTeX effect the string replacements.

2. Characters, Glyphs, and SDATA Character Entities

The WSD-related issues discussed below were common to all of our test documents, but the Cyrillic materials required an additional step not needed for the others: the absence of any standardized early Cyrillic SGML SDATA entity set standards required that we develop such an entity set alongside our WSD.[3] In the present instance, the early Cyrillic entity set is called `chsl.ent`, and consists of lines of the form

```
<!ENTITY aos SDATA "[aos     ]"
 --Cyrillic small letter a, alternate (early)-->
```

In this example, we have chosen to use the character entity `&aos;` to represent a particular early Cyrillic letter (where 'a' is the name of the letter and 'os' stands for 'Old Slav[on]ic'), and we have defined this entity's SDATA replacement text as the string `[aos    ]`, following the namespace model used in the registered ISO character entity sets in Annex D of the SGML standard. We then declare `chsl.ent` in the DTD subset of the main document with

```
<!ENTITY % chsl.ent PUBLIC "-//DJB//ENTITIES
  General Church Slavonic//EN">
%chsl.ent;
```

adding the appropriate public entity entry to the SGML Open catalog file.

These steps enable the use of early Cyrillic entities in the document instance, thus satisfying the basic encoding need, but they are not sufficient for rendering the document in any useful way. Because the WSD is not automatically parsed along with the document, unless some special action is taken, SDATA

entities will simply be replaced by their replacement text as it appears in the SDATA entity set. For example, the SDATA entity declarations in `chsl.ent` will instruct a parser to substitute for an `&aos;` entity the declared `[aos ]` replacement string; an input string `&pos;&oos;&vos;&jatos;&sos;&tos;&fjeros;` will thus be rendered `[pos ][oos ][vos ][jatos ][sos ][tos ][fjeros]`. Needless to say, we do not want this replacement string, which serves only a documentation purpose, to appear in documents intended for Slavists, and a different approach is required for rendering useful final-form documents.

If one is operating without a WSD, the most efficient way to cope with this situation is to create a new set of entity declarations, removing the default replacement text (such as `[aos ]`) and substituting for it something that will be rendered properly on a local system, such as a system-specific numerical character reference. This is the 'display entity set' discussed in Goldfarb (1990), 504. Technically, users are not permitted to change or modify the file referenced by a public identifier, and if one uses a display entity set in place of the canonic ISO SDATA entity set, one is expected to change the public identifier accordingly. One might argue, however, that changing the file referenced by a public identifier does not make the SGML document itself nonconforming because the unparsed document is not changed, and it continues to include the canonic public identifier reference for the standard SDATA entity set. There is nothing sacrosanct, or even particularly useful, about the replacement text in the ISO registered entity sets, and the value of the ISO registered entity sets is not that they standardize replacement text that may appear in a final form document, but that they standardize the inventory and names of common constituents (primarily characters) of basic writing systems.[4] If, as in our projects, one uses a WSD to support the generation of final form output, the default character entity sets may be left as they are, and the desired local replacement text may be specified in the WSD. This strategy is discussed below.

A sample character entity description in an early Cyrillic WSD looks like:

```
<character class=lexical>
  <form string='' entityLoc='aos' ucs-4='0430'
   afiicode='10993'>
    <desc>Cyrillic small letter a, alternate
     (early)</desc>
  </form>
</character>
```

In this example, we declare that the entity `&aos;` corresponds to a particular standardized ISO 10646 (UCS-4, Unicode) character, that this character may be represented only by an entity (not by a string), and that it should be rendered with the glyph standardized under AFII (Association for Font Information Interchange) code 10993. Unlike the bare SDATA entity set, which specifies a single replacement string for each declared character entity, the WSD associates separate character and glyph information with each entity. This flexibility is valuable when

dealing with writing systems that do not observe a strict one-to-one correspondence of characters (units of information) and glyphs (units of presentation), i.e., writing systems where the same underlying letter may be written in different ways, or where the same written mark may represent more than one underlying letter. This sort of many-to-many correspondence is precisely what one finds in early Cyrillic writing.[5]

3. Sample A: Cyrillic

The early Cyrillic document used for the present exercise is a fragment of the *Rus' Primary Chronicle*, the earliest East Slavic chronicle text, which purports to trace the history of the world from the creation through the establishment and early years of the Rurik dynasty in Kiev and other cities. The sample chosen for the purpose of testing the WSD architecture is a single set of brief parallel readings from a critical edition currently under preparation under the general editorship of Donald Ostrowski, of the Harvard Ukrainian Research Institute, and encoded using the TEI parallel-segmentation architecture, one line of which looks like:

```
<rdg wit='lav'>&Sos;&eos;
&nos;&aos;&chos;&nos;&eos;&mos;&bjeros;
&pos;&oos;&vos;&jatos;&sos;&tos;&fjeros;
&sos;&idecos;&juos;</rdg>
```

The early Cyrillic sample text used for this project benefits from two simplifying assumptions, neither of which is necessarily true in the case of arbitrary early Cyrillic manuscript sources. First, we assume that in a critical edition of an early Cyrillic work, all manuscript witnesses share a WSD. In fact, early Cyrillic was a supranational writing system with many local varieties, and a writing system for early Bulgarian Church Slavonic documents (for example) might differ in several places from one for early East Slavic documents. Second, more narrowly, we assume that each manuscript observes a single, consistent WSD. In fact, early Cyrillic writing was so poorly standardized that different scribes within the same manuscript might use different inventories of early Cyrillic letters and observe different rules about which letters functioned as variants of which others. If one views document encoding as a way to make explicit one's analysis of written sources, and WSD encoding as a way to make explicit one's analysis of the writing system(s) underlying these sources, it would not be unusual for a single early Cyrillic document to require different WSDs for different scribes, however inconvenient this might prove for both encoding and subsequent analysis.

4. Sample B: Hellenized Hebrew and Latinized Greek

Our second test WSD platform is more complex: the eleventh-century Irish (Latin) poem *Adelphus Adelpha Mater* not only includes non-ASCII (and non-Latin)

characters, but furthermore, employs these characters outside their usual writing systems. Specifically, the *Adelphus* text is primarily in Latin, but it contains individual words in transliterated hellenized Hebrew and latinized Greek.[6] The use of Greek letters to render Hebrew text and Latin letters to render Greek text is clearly different from the monolingual and monoalphabetic Slavic Cyrillic material discussed above, but the fundamental encoding problem is comparable: researchers need access to more information about the writing system than can be represented comfortably in an SDATA entity set.

In the case of *Adelphus Adelpha Mater* (see Figure 1), it was decided to hard-code the original Hebrew and Greek words into the poem via the markup.[7] This encoding was achieved by modifying the TEI DTD to include a specification for the attribute `reg` on the element `<frn>` (this is an abbreviation of the TEI's `<foreign>` element for convenience in manual markup systems), which is used to identify words in a foreign language. See Figure 1.

```
<l n="19">
<frn lang="he"
reg="&gimelhb;&vethb;&reshhb;&vavhb;">Gibro</frn>
<frn lang="el" reg="&pgr;&rgr;&agr;&xgr;&ogr;&ngr;
&agr;&ggr;&agr;&thgr;&ogr;&ngr;">praxon
agathon</frn>
</l>
```

Figure 1.

In Figure 1, character entities for Hebrew and Greek are contained in the `reg` attributes attached to the `<frn>` element. The element `<frn>` uses the `lang` attribute to identify the languages concerned, with the values of either *'he'* (Hebrew) or *'el'* (Greek). These character entities are associated with a WSD in the TEIheader. See Figure 2.

```
<profiledesc>
  <langusage>
    <language lang="he" wsd="foo">Some of the words
     are in Hebrew.</language>
    <language lang="el" wsd="bar">Other words are
     in Greek.</language>
  </langusage>
</profiledesc>
```

Figure 2.

As was noted above, the WSD, unlike the SDATA entity sets specified in the DTD for the principal SGML document, provides not only a single replacement

string, but also other mappings. For example, in the case of the Hebrew character whose symbolic representation is `&alephhb;`, it provides a formal UCS code 05D0 and an AFII code E140.

5. The WSD Meets Omnimark: Cyrillic

The mechanisms for encoding multilingual or complex monolingual texts, and for developing the SDATA entity sets and WSDs that document those texts, are not complicated, although, as noted above, they may prove somewhat cumbersome due to the absence of specialized tools. The preparation of the main SGML documents, the SDATA entity sets, and the WSDs fulfills the first function of the WSD mentioned earlier: to document the transcription system in a way that will provide human readers with access to a structured description of this system. This type of encoding fulfills fully the mandate of the Text *Encoding* Initiative, in that it yields a text that has been encoded according to the TEI guidelines, but it does not provide a document that can be rendered easily for use by colleagues who are not also competent SGML engineers. In an attempt to make the SGML documents more accessible to such colleagues, we undertook to process the WSD so as to provide different views of orthographically-complex SGML documents.

Two general strategies were applied to the early Cyrillic test file:

1. parse the WSD into memory when it is mentioned in a general TEI document and access an in-memory table for transformation, or
2. parse the WSD independently of any other TEI document to output different SDATA entity sets, which may be saved as system entities and accessed directly by any parser that supports SDATA entity replacement.

These strategies were applied to three types of problems:

1. generate and use character-level representations,
2. generate and use glyph-level representations, and
3. generate and use a mixed representation, which could be employed, for example, to search according to characters but render with glyphs.

The input files, OmniMark scripts,[8] and output files used in the Cyrillic portion of this project are available on the World Wide Web at `http://clover.slavic.pitt.edu/~djb/sgml/tei10/`.

6. Conversion to LaTeX: Hebrew

The strategy for Hebrew was to modify the TEI DTD by defining for the `<frn>` element a `reg` attribute, which holds the regularized (Hebrew) character sequence in an entity reference. This frees the content of the `<frn>` element to hold the transliterated hellenized Hebrew of the source document, so that `<frn lang="he" reg="&alephhb;&vethhb;&gimelhb;">abg</frn>` results in the rendering: 'abg [גבא]'

A short program was written using Omnimark to convert the SGML to LaTeX for printing. This program invokes the recursive function `do sgml-parse` to suspend processing of the main document when the entity reference in the `WSD` attribute of the `<language>` element is encountered, to process the WSD file itself, and then to resume processing of the main document. This processing model enables the character entity names in the WSD to be interpreted and written out to disk to a style file in LaTeXformat, so that they can be read in again during the LaTeXprocessing to implement the exact character encoding required for the font used.

As was noted in the case of Cyrillic, the WSD does not obviate the need for the standard ISO Hebrew character entity set, which must also be present because the character entity names in the WSD are in attributes declared as `entity`, which means that the ISO declarations must exist at that point. Although, as noted above, editing the ISO Hebrew character entity file to reflect the character encoding code points might technically be a more direct method,

- this would require modifying a standard file that would normally be present in the SGML processing system of any user who intends to process Hebrew, and using the standard version of such SDATA character entity sets improves portability by removing the need to attach a different, edited version to any instance that is exchanged; and
- the use of the WSD method provides for much better control over the encoding specification, as well as allowing ancillary inline documentation.

The files associated with the Hebrew sample are available for inspection at `http://imbolc.ucc.ie/~pflynn/wsd/`.

7. Conclusion: The Need for Dynamic WSD Processing

Omnimark coped successfully, and even elegantly, with all of the tasks it was set, but the batch approach undertaken here is ultimately capable only of generating multiple static views, without real support for dynamic inquiry. For example, there are situations where a Slavist may wish to conflate glyphic variants of character `foo` during searching, while maintaining a distinction between glyphic variants of character `bar`, and the potential number of such hybrid views is for all practical purposes unlimited.[9] The strategies discussed here provide the user with access to character-based, glyph-based, and mixed views of the input text, but they do not support access to ad hoc combinations of character-level and glyph-level information. The development of SGML tools capable of supporting dynamic WSD access will greatly enhance the utility of WSDs for scholars who work with orthographically complex writing systems.

TECHNICAL ADDENDA

As this article goes to press, it should be noted that SoftQuad's *Author/Editor* and *Panorama Pro* SGML editing and display products, referred to in section 1, have been acquired by Interleaf Corp.

The *Omnimark* conversion tool is now available in server form (*Konstruktor*), which in theory means that dynamic enquiry could be combined with formatting such as described here, in real time, but this method is currently untested for the present texts.

Notes

[1] For example, *Author/Editor*, SoftQuad's widely-used graphical SGML editor, does not support this type of replacement.

[2] Panorama also supports SDATA entity replacement indirectly by way of an `sdata.map` file, which associates not the SDATA entity name, but its replacement text, with font and glyph offset specifications.

[3] As is the case with SGML documents in general, character entities included in the principal SGML document instance must be declared in the DTD for the principal SGML document, since any WSD specifications supplement, and do not replace, these basic entity declarations. For general background see especially Harry Gaylord's two essays on character entity sets and WSDs (1992, 1995) and section 25 of *P3* (Sperberg-McQueen and Burnard, 1994).

[4] The format of a Formal Public Identifier as defined in ISO 8879 does allow for presentation-specific variants. See also the discussion in DeRose (1997), 130–34.

[5] See Birnbaum (996) for examples and discussion.

[6] The text is taken from a scholarly reconstruction of the poem by Dr. David Howlett (1995).

[7] The authors are grateful to Professor Lewis M. Barth, Hebrew Union College, for his help in identifying the Hebrew characters and for suggested corrections to the Hebrew words, and to Ms. Sinead O'Sullivan, St. Anne's College, Oxford, for identifying the Greek characters.

[8] Although we chose to implement our project in Omnimark, any system that can perform transformations of SGML documents, including the parsing of SUBDOC entities, could be used in its stead.

[9] See Birnbaum (1996) for examples and discussion.

References

Birnbaum, D. J.: "Standardizing Characters, Glyphs, and SGML Entities for Encoding Early Cyrillic Writing". *Computer Standards & Interfaces* 18 (1996), 201–52.

DeRose, S. J.: *The SGML FAQ Book*, Boston, Dordrecht, London: Kluwer Academic Publishers, 1997.

Gaylord, H.: "Character Entities and Public Entity Sets (TEI TR1 W4)". Technical report, Author, Groningen, 1992.

Gaylord, H.: "Character Representation". *Computers and The Humanities* 29(2) (1995), 51–73.

Goldfarb, C. F.: *The SGML Handbook*, Oxford: Clarendon Press, 1990.

Howlett, D.: "Five Experiments in Textual Reconstruction and Analysis". *Peritia: Journal of the Medieval Academy of Ireland* 9, 1995.

Sperberg-McQueen, C. and L. Burnard: "Guidelines for Electronic Text Encoding and Interchange (TEI P3)". Technical report, ACH/ACL/ALLC Text Encoding Initiative, Chicago and Oxford, 1994.