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 16
Text Encoding Initiative
Chicago, Oxford

January 27, 1993

Draft Version 2, January 27, 1993

Note

This is the current draft (as of January 27, 1993) 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. Direct and indirect support has also been received from the University of Illinois at Chicago, the Oxford University Computing Service, the University of Arizona, the University of Oslo, Queen's University (Kingston, Ont.), and Ohio State University. We acknowledge assistance with software acquisition from the following firms: SEMA Group and Yard Software (for Mark-It and Write-It), Software Exoterica (for CheckMark and Xtran), SoftQuad, Inc., (for Author/Editor and RulesBuilder), Electronic Book Technology (for demonstration copies of DynaText), Xerox Corporation (for Ventura Publisher).

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 4

Additional Tag Sets

Chapter 16

Segmentation and Alignment

This chapter and the following two propose a number of ways in which encoders may represent analyses of the structure of a text which are not simply linear or hierarchic. In particular, mechanisms are provided for the following common requirements:[1]

- the ability to encode the fact that distinct textual segments *correspond*, for example because one is a translation of the other
- the ability to show that distinct textual segments should be *aggregated*, for example because they form a discontinuous segment of some kind
- the ability to show that one segment is an *echo* or *copy* of another
- the ability to associate segments of texts with an abstract interpretation or analysis of their significance

These mechanisms are all implemented using the same basic set of techniques, all of which depend on the ability to refer to a segment by some form of identifier. The most convenient such identifier, and that which is recommended by these Guidelines wherever possible, is provided by the global *id* attribute, as defined in section 3.3, "Global Attributes," [in separate fascicle]. In addition, for segments which are located in different SGML documents, or to which identifiers cannot be attached (perhaps because they are held on read-only media), an additional *TEI extended pointer mechanism* is defined, (see section 16.3, "External Pointers and References," on page 13).

This chapter therefore begins with a discussion of the ways in which textual elements of any kind can be referred to within TEI documents (section 16.1, "Pointers and Links," on page 5), together with the definition of the TEI extended pointer mechanism. The facilities described in this section provide for very general linkages, of the type commonly described as *hypertextual*; this is followed by a discussion of two specific kinds of linkage, representing *correspondence* (section 16.4, "Correspondence," on page 33), and *aggregation* (section 16.5, "Aggregation and Virtual Elements," on page 45).

Other chapters also describe some topics relevant to linking and alignment. Chapter 17, "Simple Analytic Mechanisms," [in separate fascicle] discusses ways of representing simple analyses of text and of linking them with textual segments; it also discusses the third kind of link mentioned above, where one segment is regarded as a copy or echo of another (in section 17.4, "Virtual Copies," [in separate fascicle]). Chapter 18, "Feature Structure Analysis," [in separate fascicle] discusses in detail the most fully articulated way of representing arbitrary analyses proposed in these Guidelines, as *feature structures*; such analyses may be linked with a text with the same mechanisms described in this chapter.

---

[1] The primary responsibility for this chapter was borne by TEI work groups TR3 (hypertext and hypermedia) and AI1 (general linguistics). The text was drafted in part by Steven J. DeRose (for TR3) and in part by D. Terence Langendoen (for AI1).

In each case a choice of mechanisms is offered between simple but less general methods depending on the use of attribute values only, and more general but more complex methods depending on the use of specialized elements.

The following DTD fragment shows the overall organization of the class of analytic elements discussed in the remainder of this chapter, and declares the set of global analytic attributes available when this tag set is used.

```
<!-- 16:  Analytic attributes and elements                    -->

<!-- First, we declare a set of additional global attributes  -->
<!-- made available by this additional tag set.               -->

<!ENTITY % a.ana '
          corresp           IDREFS            #IMPLIED
          link              IDREFS            #IMPLIED
          partof            IDREF             #IMPLIED
          prev              IDREF             #IMPLIED
          next              IDREF             #IMPLIED       ' >

<!-- Next, we declare the various elements, group by
group.  -->

<!-- ... declarations from section 16.1                       -->
<!--     (Links)                                              -->
<!--     go here ...                                          -->
<!-- ... declarations from section 16.3                       -->
<!--     (Extended pointers)                                  -->
<!--     go here ...                                          -->
<!-- ... declarations from section 16.4                       -->
<!--     (Correspondence)                                     -->
<!--     go here ...                                          -->
<!-- ... declarations from section 16.5                       -->
<!--     (Virtual elements)                                   -->
<!--     go here ...                                          -->
```

This tag set is made available by the mechanisms described in section 3.5, “Invocation of TEI DTDs,” [in separate fascicle]; in a document which uses the markup described in this chapter, the document type declaration should contain the following declaration of the entity *TEI.sa*, or an equivalent one:

```
<!ENTITY % TEI.sa system 'teisa2.dtd'>
```

The entire document type declaration for a document using this additional tag set might look like this:

```
<!DOCTYPE TEI.2 system 'tei2.dtd' [
   <!ENTITY % TEI.base system 'teipros2.dtd'>
   <!ENTITY % TEI.sa   system 'teisa2.dtd'>
]>
```

16.1 Pointers and Links

A *pointer* is a special element, the function of which is to represent an association of some kind between one location in a document (at which the element is placed) and one or more others, known collectively as its *target*. One familiar kind of pointer, discussed in section 6.6, "Simple Cross References," [in separate fascicle], is a cross-reference. Another, typical of many "hypertext" systems, is a button or other device used to allow the user to control the non-sequential processing which characterizes such systems. A *link* (in the terminology of these Guidelines) is a special element, similar to a pointer, which represents an association between two (or more) locations by specifying each location explicitly. Its own location is irrelevant to the intended linkage.

In the simplest case, pointers and links are encoded using the elements **<ptr>**, **<ref>**, **<link>**, and **<linkGrp>**. (For the more general case, see section 16.3, "External Pointers and References," on page 13.)

<ptr> : defines a pointer to another location in the current document. Attributes include:
- *target* : specifies the destination of the pointer as one or more SGML identifiers

<ref> : defines a reference to another location in the current document, in terms of one or more identifiable elements, possibly modified by additional text or comment. Attributes include:
- *target* : specifies the destination of the reference as one or more SGML identifiers

<link> : defines an association or hypertextual link among elements or passages, of some type not more precisely specifiable by other elements. Attributes include:
- *targets* : specifies the SGML identifiers of the elements or passages to be linked or associated.

<linkGrp> : defines a collection of associations or hypertextual links. Attributes include:
- *type* : categorizes the group of associations in some respect, using any convenient set of categories.

The **<ptr>** and **<ref>** elements bear a *target* attribute (in the singular), because they point, conceptually, at a single target, even if that target may be discontinuous in the document. The **<link>** element bears a *targets* attribute, with a plural name, because it points, conceptually, to at least two targets, each of which is a unitary object.

The **<ptr>**, **<ref>**, and **<link>** elements are all members of the class *pointer*, and share a common set of attributes:

type : categorizes the pointer in some respect, using any convenient set of categories.
resp : specifies the creator of the pointer.
crdate : specifies when the pointer was created.
targType : specifies the kinds of elements to which this pointer may point.
targOrder : where more than one identifier is supplied as the value of the *target* attribute, this attribute specifies whether the order in which they are supplied is significant. Legal values include:

- *Y* : Yes: the order in which IDREFs are specified as the value of a *target* attribute should be followed when combining the targeted elements.
- *N* : No: the order in which IDREFs are specified as the value of a *target* attribute has no significance when combining the targeted elements.
- *U* : Unspecified: the order in which IDREFs are specified as the value of a *target* attribute may or may not be significant.

evaluate : specifies the intended meaning when the target of a pointer is itself a pointer. Legal values include:

- *all* : if the element pointed to is itself a pointer, then the target of that pointer will be taken, and so on, until an element is found which is not a pointer.
- *one* : if the element pointed to is itself a pointer, then its target (whether a pointer or not) is taken as the target of this pointer.
- *none* : no further evaluation of targets is carried out beyond that needed to find the element specified in the pointer's target.

The elements **<ptr>** and **<ref>**, which are used to implement pointers within a single TEI document, are discussed in section 6.6, “Simple Cross References,” [in separate fascicle]. As noted there, these elements specify their target or targets by supplying a list of identifiers as the value of their *target* attribute. The position of **<ref>** and **<ptr>** elements within a document is, in general, determined by their meaning: the elements occur where the cross-reference or hypertext link occurs in the text.

The element **<link>** may be used to implement general purpose links among elements in a TEI document; it is intended for use where none of the more specific kinds of linkage discussed in the remainder of this chapter and the two subsequent ones is appropriate. The targets of a link are represented by its *targets* attribute, which takes a list of SGML identifiers as its value, much like the *target* attribute of **<ref>** and **<ptr>**, but with slightly different meaning, as described below. Unlike that of **<ptr>** and **<ref>**, the location of **<link>** elements within the SGML document is arbitrary.

The element **<linkGrp>** may be used to group links together in a single part of the document; such a collection represents what is often referred to in the hypertextual literature as a *web*, a term introduced by the Brown University FRESS project in 1969. Typical software might hide a web entirely and merely use it as a source of information about links, which are displayed independently at their referenced locations. Alternatively, software might provide a direct view of the link collection, along with added functions for manipulating the collection, as by filtering, sorting, and so on. (Such processing is not specified by these guidelines.) The **<linkGrp>** element also provides a convenient way of establishing a default for the *type* attribute on a group of links of the same type: by default, the *type* attribute on a **<link>** element has the same value as that given for *type* on the enclosing **<linkGrp>**.

All elements pointed to or linked by these elements must be identifiable using the global *id* attribute. This implies that they must be present in the same document, and that they must bear unique *id* values. Pointing or linking to external documents and pointing or linking where SGML identifiers are not available is implemented by the external pointing mechanisms discussed in section 16.3, “External Pointers and References,” on page 13. Using those mechanisms, pointers may be represented using the **<xptr>** and **<xref>** elements. External links and links to elements without identifiers do not require a special element; they may be represented using the standard **<link>** element, but an intermediate **<xptr>** element must be provided within the current document, to bear the *id* attribute used in the target of the link. The syntax and semantics of the **<link>** and **<linkGrp>** elements are identical, whether their targets are within the current document or not.

As an example of the use of these elements, consider the practice (common in 18th century English verse and elsewhere) of providing footnotes citing parallel passages from classical authors. In the usual case, such footnotes might of course simply be encoded using the general purpose **<note>** element, embedded within the text, as follows:[2]

```
<l>(Diff'rent our parties, but with equal grace</l>
<l>The Goddess smiles on Whig and Tory race,</l>
<note type='imitation' place=foot anchored=no><bibl>
    Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
<l>'Tis the same rope at sev'ral ends they twist,</l>
<l>To Dulness, Ridpath is as dear as Mist)</l>
```

For this simple type of text-note link, it is unlikely that any mechanism more complex than that shown in the example above would be needed. For completeness however, and to facilitate comparison of the various techniques proposed by these Guidelines, we now present three alternative methods of linking the note with the text annotated: pointing from text to note, pointing from note to text, and linking note and text with a **<link>**.

All of these methods allow the annotation to be transcribed separately from the text itself. If for example the annotation occurs as an end-note (or in a separate commentary) rather than a footnote, it may be desired to transcribe it where it occurs rather than embedding it within the text. In some cases, especially when the annotations are numerous, it might be felt more convenient to encode them separately even if they do appear as footnotes. The techniques used in the following examples may thus be useful in some cases where the simple tagging shown above cannot be used.

First, a **<ptr>** element might be placed at an appropriate point within the text to link it with the annotation:

---

[2] The *type* attribute on the note is used to classify the notes using the typology established in the Advertisement to the work: “The *Imitations* of the Ancients are added, to gratify those who either never read, or may have forgotten them; together with some of the Parodies, and Allusions to the most excellent of the Moderns.” In the source text, the text of the poem shares the page with two sets of notes, one headed “Remarks” and the other “Imitations”.

```
<l>(Diff'rent our parties, but with equal grace</l>
<l>The Goddess smiles on Whig and Tory race,<ptr
   rend='unmarked' target=N3.284></l>
<l>'Tis the same rope at sev'ral ends they twist,</l>
<l>To Dulness, Ridpath is as dear as Mist)</l>
 ...
<!-- elsewhere in the document ... -->
<note id=N3.284 type='imitation' place=foot anchored=no><bibl>
    Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
```

The **<note>** element has been given an arbitrary identifier (N3.284) to enable it to be specified as the target of the pointer element. Because there is no marker in the text to signal the existence of the annotation, the *rend* attribute has been given the value “unmarked”.

Secondly, the *target* attribute of the **<note>** element can be used to point at its associated text:

```
<l id=L3.283>(Diff'rent our parties, but with equal grace</l>
<l id=L3.284>The Goddess smiles on Whig and Tory race,</l>
<l id=L3.285>'Tis the same rope at sev'ral ends they twist,</l>
<l id=L3.286>To Dulness, Ridpath is as dear as Mist)</l>
 ...
<!-- elsewhere in the document ... -->
<note target=L3.284
    type='imitation' place=foot anchored=no><bibl>
    Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
```

An SGML identifier has been given to each line, for convenience in transcribing other commentaries, even though in this example the note points only at line L3.284.

If the note, as frequently occurs, begins with an explicit reference to the line annotated, a **<ref>** element may be used to encode the reference. In order to facilitate hypertext jumps from text to annotation, one might also prefer to encode both pointers from note to text and a pointer from text to note:

```
<l id=L3.283>(Diff'rent our parties, but with equal grace</l>
<l id=L3.284>The Goddess smiles on Whig and Tory race,<ptr
   rend='unmarked' target=N3.284></l>
<l id=L3.285>'Tis the same rope at sev'ral ends they twist,</l>
<l id=L3.286>To Dulness, Ridpath is as dear as Mist)</l>
 ...
```

```
<!-- elsewhere in the document ... -->
<note id=N3.284 target=L3.284
    type='imitation' place=foot anchored=no><ref
    target='L3.284' rend='sc'>
    Verse 283-84.
    <quote><l>&mdash;&mdash;. With equal grace</l>
           <l>Our Goddess smiles on Whig and Tory race.</l>]</ref>
    <bibl>Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
```

In the third method, identifiers are supplied for both verse line and annotation, and a **<link>** element is used to associate the two:

```
<l id=L3.283>(Diff'rent our parties, but with equal grace</l>
<l id=L3.284>The Goddess smiles on Whig and Tory race,<ptr
   rend='unmarked' target=N3.284></l>
<l id=L3.285>'Tis the same rope at sev'ral ends they twist,</l>
<l id=L3.286>To Dulness, Ridpath is as dear as Mist)</l>
 ...
<!-- elsewhere in the document ... -->
<note id=N3.284 target=L3.284
    type='imitation' place=foot anchored=no><ref
    target='L3.284' rend='sc'>
    Verse 283-84.
    <quote><l>&mdash;&mdash;. With equal grace</l>
           <l>Our Goddess smiles on Whig and Tory race.</l>]</ref>
    <bibl>Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
 ...
<!-- ... and yet elsewhere in the document ... -->
<link type=imitation targets='N3.284 L3.284'>
```

The *targets* attribute of the **<link>** element here bears two identifiers. In general, it is the responsibility of an application to determine how the two indicated elements are associated, although encoders may provide a hint, as here, by the use of the *type* attribute. The values for this attribute are not specified in these Guidelines, but should be documented in the TEI Header, in the **<encodingDesc>** element. These Guidelines do however provide for more specific encoding of some very commonly occurring kinds of association, for example, correspondence (see section 16.4, “Correspondence,” on page 33) or aggregation (see section 16.5, “Aggregation and Virtual Elements,” on page 45); when these more specialized tags apply, they should normally be preferred to the more general purpose **<link>** element.

These elements are formally defined as follows:

```
<!-- 16.1:  Links                                                  -->
<!-- Elements PTR and REF are defined in the Core                  -->

<!ELEMENT link          - o  EMPTY                                  >
<!ATTLIST link               %a.global;
          resp               CDATA                #IMPLIED
          crdate             CDATA                #IMPLIED
          targType           NAMES                #IMPLIED
          targOrder          (Y | N | U)          U
          evaluate           (all | one | none)   #IMPLIED
          targets            IDREFS               #REQUIRED
          type               CDATA                %INHERITED;     >
<!ELEMENT linkGrp       - -  (link)+               +(ptr | xptr) >
<!ATTLIST linkGrp            %a.global;
          type               CDATA                #IMPLIED        >
<!-- This fragment is used in sec. 16                              -->
```

16.2 Multi-headed Pointers

The sample encodings given in the previous section do not represent the fact that the annotation is associated with both lines of the couplet rather than an arbitrary point within the second line. This is partly because there is no single element in the text containing the two lines. One simple method of overcoming the problem would be to introduce such an element, either explicitly — for example, a **<lg>** element, as defined in section “REFID=COve not defined” — or implicitly, using the **<join>** virtual element discussed in section 16.5, “Aggregation and Virtual Elements,” on page 45. In either case the new element's *id* attribute could be used to provide the value for the *target* attribute of the **<note>** element or of the **<ref>** element, embedded within the **<note>**. Alternatively, if the number of elements to be combined in this way is not too large, it may be simpler to use a “multi-headed” pointer from the note into the text, supplying identifiers for each of the two lines as the target for the note and for the cross-reference.

```
<l id=L3.283>(Diff'rent our parties, but with equal grace</l>
<l id=L3.284>The Goddess smiles on Whig and Tory race,</l>
 ...
<!-- elsewhere in the document ... -->
<note id=N3.284 target='L3.283 L3.284'
    type='imitation' place=foot anchored=no><ref
    target='L3.283 L3.284' targOrder='Y' rend='sc'>
    Verse 283-84.
    <quote><l>&mdash;&mdash;. With equal grace</l>
           <l>Our Goddess smiles on Whig and Tory race.</l>]</ref>
    <bibl>Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
```

When the *target* attribute of a **<ptr>** or **<ref>** element specifies more than one element, the indicated elements are always understood to be combined or aggregated in some way to produce the object of the pointer. The attribute *targOrder* should be used, as here, to indicate whether or not the sequence in which constituents of the target list are specified should be followed when combining them. An additional check on the constituents of the list may be specified by the *targType* attribute; if the above example were:

```
<ref target='L3.283 L3.284' targOrder='Y' targType='L PTR'>
Verse 283-84. ... </ref>
```

then an application could require that the elements indicated by identifiers L3.283 and L3.284 were instances of element types **<l>** or **<ptr>** only.

This use of “multiheaded pointing” should be carefully distinguished from that used by the *targets* attribute of the **<link>** element. As the attribute name suggests, in the latter case, the indicated elements are not simply combined together as if they were a single target, but associated in some respect, specifically, that indicated by the *type* attribute. Crucially, the intended relationship between every item in a list of identifiers must be identical. To represent the link discussed above as follows would be erroneous:

```
<!-- ... and yet elsewhere in the document ... -->
<link type=imitation targets='N3.284 L3.283 L3.284'>
```

This is not correct, because the association between N3.284 and L3.284 (the note and one line of text) is not of the same kind as that between L3.283 and L3.284 (the two lines of text). A processor cannot be expected to process this kind of target list correctly. To represent the link correctly, an intermediate pointer or a virtual element must be introduced:

```
<!-- ... and yet elsewhere in the document ... -->
<ptr id=X3.284 target='L3.283 L3.284'>
<!-- or, synonymously              ,           -->
<!-- <join id=X3.284 parts='L3.283 L3.284'>    -->
<link type=imitation targets='N3.284 X3.284' evaluate=all>
```

The link now associates the note with a pointer element, rather than with the lines of verse directly. This pointer element in turn points to the two lines. Indirect pointing of this kind is of particular importance when the extended pointer mechanism is used (see section 16.3, “External Pointers and References,” on page 13) but is essential wherever links are made among targets, some of which are themselves multi-headed. The attribute *evaluate* should be used to require that any pointer encountered as the target of a pointer is itself evaluated, where this is the intended action. (If *evaluate* has the value *none*, the link target will be the pointer itself, rather than the objects it points to; this makes it possible to make links between links, and to point at pointers.)

Finally, where a number of links share the same *type* attribute value, we can avoid the need to specify the *type* attribute on each **<link>** by defining a **<linkGrp>** element to contain all such links. A link group might be used to hold all the links of a particular type (e.g. “imitation”). For example, assuming the existing elsewhere in the document of the following other notes, of a kind similar to that already discussed,

```
<l id=L2.79>A place there is, betwixt earth, air and seas</l>
<l id=L2.80>Where from Ambrosia, Jove retires for ease.</l>
<l> ...
<l id=L2.88>Sign'd with that Ichor which from Gods distills.</l>
<l> ...
<l id=L3.283>(Diff'rent our parties, but with equal grace</l>
<l id=L3.284>The Goddess smiles on Whig and Tory race,</l>
<l id=L3.285>'Tis the same rope at sev'ral ends they twist,</l>
<l id=L3.286>To Dulness, Ridpath is as dear as Mist)</l>

<!-- ... -->
<!-- elsewhere in the document ... -->
<note id=N2.79 type=imitation><bibl>Ovid Met. 12.</bibl>
    <quote lang=la>
        <l>Orbe locus media est, inter terrasq; fretumq;
        <l>C&oelig;lestesq; plagas &mdash;
    </quote>
</note>
<note id=N2.88 type=imitation>
    Alludes to <bibl>Homer, Iliad 5</bibl>....
</note>
<!-- ... -->
<note id=N3.284 type='imitation'><bibl>Virg. &AElig;n. 10.</bibl>
    <quote>
    <l>Tros Rutulusve fuat; nullo discrimine habebo.</l>
    <l>&mdash;&mdash; Rex Jupiter omnibus idem.</l>
    </quote>
</note>
 ...
```

then it might be convenient to group all the links involving notes of type “imitation” at a single place in the encoded text, as follows:

```
<linkGrp type=imitation>
    <link targets='N2.79 L2.79'>
    <link targets='N2.88 L2.88'>
    <!-- ... -->
    <ptr id=X3.284 target='L3.283 L3.284'>
    <link targets='N3.284 X3.284' evaluate=all>
</linkGrp>
```

16.3 External Pointers and References

Where the object of a link or pointer element is not contained within the current document, or where it does not bear an *id* attribute, it is not possible to point at it with a **<ptr>** or **<ref>** element, nor to link it directly with a **<link>** element, because no IDREF value can be supplied for the *target* or *targets* attribute of these elements. In such cases, the encoder must indicate the intended element indirectly by means of the elements discussed in this section.

- **<xptr>** : defines a pointer to another location in the current document or an external document. Attributes include:
  - *doc* : specifies the document within which the required location is to be found.
  - *from* : specifies the start of the destination of the pointer as an expression in the TEI extended pointer notation.
  - *to* : specifies the endpoint of the destination of the pointer as an expression in the TEI extended pointer notation.
- **<xref>** : defines a reference to another location in the current document, or an external document, using an extended pointer notation, possibly modified by additional text or comment. Attributes include:
  - *doc* : specifies the document within which the required location is to be found.
  - *from* : specifies the start of the destination of the pointer as an expression in the TEI extended pointer notation.
  - *to* : specifies the endpoint of the destination of the pointer as an expression in the TEI extended pointer notation.

As members of the element class *pointer*, these elements share the following set of attributes:

- *type* : categorizes the pointer in some respect, using any convenient set of categories.
- *resp* : specifies the creator of the pointer.
- *crdate* : specifies when the pointer was created.
- *targType* : specifies the kinds of elements to which this pointer may point.
- *targOrder* : where more than one identifier is supplied as the value of the *target* attribute, this attribute specifies whether the order in which they are supplied is significant. Legal values include:
  - *Y* : Yes: the order in which IDREFs are specified as the value of a *target* attribute should be followed when combining the targeted elements.
  - *N* : No: the order in which IDREFs are specified as the value of a *target* attribute has no significance when combining the targeted elements.
  - *U* : Unspecified: the order in which IDREFs are specified as the value of a *target* attribute may or may not be significant.
- *evaluate* : specifies the intended meaning when the target of a pointer is itself a pointer. Legal values include:
  - *all* : if the element pointed to is itself a pointer, then the target of that pointer will be taken, and so on, until an element is found which is not a pointer.
  - *one* : if the element pointed to is itself a pointer, then its target (whether a pointer or not) is taken as the target of this pointer.

none : no further evaluation of targets is carried out beyond that needed to find the element specified in the pointer's target.

Unlike the pointer elements discussed in the previous section, these elements do not specify their target by means of a *target* attribute. Instead they use one or both of the attributes *from* and *to* to delimit a portion of some document. In other respects, these elements correspond with the elements **<ptr>** and **<ref>** discussed in sections 6.6, “Simple Cross References,” [in separate fascicle], and 16.1, “Pointers and Links,” on page 5. Note that there is no element **<xlink>** corresponding with the **<link>** element; this makes it possible to make links both within and between documents using the same syntax, as further discussed below.

The values of the *from* and *to* attributes on the **<xptr>** and **<xref>** elements indicate the point or passage being referred to by showing how to locate it, using one or more special keywords, as defined below in section 16.3.1, “TEI Extended Pointer Syntax.” Examples are given there.

The **<xptr>** and **<xref>** elements are formally defined as follows:

```
<!-- 16.3:  Extended pointers                                  -->
<!ELEMENT xref         - o  EMPTY                                >
<!ATTLIST xref              %a.global;
                            %a.pointer;
          doc               ENTITY              #IMPLIED
          from              CDATA               #IMPLIED
          to                CDATA               #IMPLIED         >
<!ELEMENT xptr         - o  EMPTY                                >
<!ATTLIST xptr              %a.global;
                            %a.pointer;
          doc               ENTITY              #IMPLIED
          from              CDATA               #IMPLIED
          to                CDATA               #IMPLIED         >
<!-- This fragment is used in sec. 16                          -->
```

16.3.1 TEI Extended Pointer Syntax

The elements **<xptr>** and **<xref>** are used to represent a link between their own location (the “link origin”) and some other location (the “destination”), which may or may not be in the same document. Software supporting intra- and inter-document links (e.g. hypertext systems) should provide access from the location of such an element to the destination.

This section defines the allowable values for the target attributes (*from*, *to*, and *doc*) of the **<xptr>** and **<xref>** elements:

doc: names the external entity, if any, within which the destination appears
from: identifies the starting point of the destination, using the keywords and syntax described below
to: identifies the ending point of the destination, using the keywords and syntax described below

An **<xptr>** or **<xref>** element with *no* attributes at all is, by definition, a link to the root (i.e. the *document element* — by default, this is the **<TEI.2>** element) of the document in which it appears.

The *doc* attribute value must be the name of an entity declared in the SGML document type declaration. If only the *doc* attribute is given a value, then by definition the destination is the entire entity named by the *doc* value. A more specific location within another entity must be specified with the *from* and the *to* attributes, as described below.

The *from* and the *to* attributes indicate the specific location pointed at, within the entity named by the *doc* attribute (or within the current document, if no *doc* value is given). Their values are referred to below as *location pointer specifications*. When both attributes are specified, the span pointed at by the element runs from the starting point of the span indicated by *from* to the ending point of the string specified by *to*. If the latter precedes the former in the document, then the pointer is in error and fails. If only the *from* attribute is specified, the *to* attribute defaults to the same value; the effect is that the element as a whole points to the span indicated by the *from* attribute. It is a semantic error to specify a value for *to* but not for *from*.

16.3.1.1 Location Ladders

Each location pointer specification consists of a sequence of *location terms*, each of which consists of a keyword specifying a *location type* followed by one or more parenthesized parameter lists, each of which specifies a *location value* via a list of parameters. Location types and values, and the parameters within a location value, must be separated by white space characters.

Using terms borrowed from *HyTime*,[3] we say that each TEI location term in a specification provides the *location source* for the next, and the entire specification is equivalent to a *location ladder*. By specifying the entire ladder in a single attribute value, the TEI extended pointer mechanism greatly reduces the syntactic and processing complexity of hypertextual pointers.

In formal terms:[4]

---

[3] HyTime is an international standard (ISO 10744) built on SGML. It provides facilities for representing both static and dynamic information for processing and interchange by hypertext and multimedia applications. See *ISO/IEC 10744 Information Technology - Hypermedia/Time-based Structuring Language (HyTime)* (Geneva: International Organization for Standardization, 1992). For discussion of the relation between TEI proposals and Hytime, see chapter 35, “Relationship of TEI to Other Standards,” [in separate fascicle].

[4] The formal grammar specified in this section is reproduced as a unit in the appendix. The notation used for this formal grammar is that defined in chapter 42, “Formal Grammar for the TEI-Interchange-Format Subset of SGML,” [in separate fascicle].

```
ladder   ::=  locterm
         |    ladder locterm
```

16.3.1.2 Location Terms

The keywords used in location terms are these; references to “the tree” mean the tree representing the SGML document hierarchy.

root: points at the root of the target document
here: points at the location of the pointer
id: points at an ID within the target document
ref: gives a “canonical reference” to a location in the target document
child: indicates an element found by descending one level in the tree
descendant: indicates an element found by descending one or more levels in the tree
ancestor: indicates an element found by ascending one or more levels in the tree
previous: indicates an element found by traversing the older siblings of the current location source
next: indicates an element found by traversing the younger siblings of the current location source
pattern: specifies a regular expression to be located within the existing location source
token: points at one or more *tokens* in the character content of the location source
str: points at one or more characters in the character content of the location source
space: points at a location using coordinates in some (application-defined) n-dimensional space
foreign: points at a location using some non-SGML method, and gives the name of the method
HyQ: points at a location using the HyQ query language defined by ISO 10744 (HyTime)
ditto: (in the *to* attribute only) points at the same span as was indicated by the *from* attribute

In formal terms:

```
locterm  ::=  'ROOT'                         // default first location
         |    'HERE'                         // location of the xptr
         |    'ID' '(' NAME ')'              // only one ID allowed.
         |    'REF' '(' characters ')'       // only one ref allowed
         |    'CHILD' steps
         |    'DESCENDANT' steps
         |    'ANCESTOR' steps
         |    'PREVIOUS' steps
         |    'NEXT' steps
         |    'PATTERN' regs                 // mult patterns allowed
         |    'TOKEN' '(' range ')'
         |    'STR' '(' range ')'
         |    'SPACE' '(' NAME ')' pointpair
         |    'FOREIGN' parms
```

```
            |    'HYQ' parms
            |    'DITTO'                       // valid only in TO att.
            ;
```

Note that the keywords, though shown here quoted in uppercase, are not case sensitive.

Each location term specifies a location in the target document; this location may be a single point, more often a span of text (often the span of a single element) within the target document. The location ladder as a whole is interpreted from left to right, and each location term specifies a location relative to the location specified by the sequence prior to that point (i.e. to its location source). Unless *here* or *id* is specified as the first location term, the beginning location source is always *root*. An empty location sequence thus is the same as *root* and specifies the entire destination entity.

In general, the search for the location specified by a location term will be conducted only within its location source (i.e. within the location already identified by preceding location terms). There are however several exceptions. The terms *root*, *here*, and *id* all ignore the location source defined by any preceding terms and therefore make sense only as the first items in the ladder. The term *ancestor*, *next*, and *previous* do not ignore the location source, but select a new span from the adjacent or enclosing portions of the text, and not from within the location source. Finally the location terms *foreign*, *space*, and *HyQ* are not defined fully here; they may or may not ignore the existing location source.

Some of the location terms make sense only in SGML documents; these are *id*, *child*, *ancestor*, *descendant*, *previous*, and *next*. The latter four involve traversing the tree representing the SGML document hierarchy and are most easily understood when their location source is a single SGML element. If the location source is not a single SGML element, the tree-traversal keywords operate upon its beginning end-point, its “front end” (in English, this will be the leftmost point of the location source; in Arabic or Hebrew it will be the rightmost point). In this case *child* and *descendant* have no meaning, since character data has no descendants in the document tree; the first *ancestor* of such a location source is the element immediately containing the character data in question, and the siblings referred to by *next* and *previous* are the other children of that immediately containing element.

The details of each keyword are given below, along with definitions of their syntax and semantics of their results. Examples are also provided. It is strongly recommended that when IDs are available, they should be used in preference to the other methods for pointing defined here.

For all keywords, the description assumes that the target document does in fact contain a span or element which matches the description; otherwise, the location term has no referent and is said to “fail”. If any location term fails, the entire pointer fails. No backtracking or retrying is performed (and indeed for the most part the location terms are defined as having only one matching location, so backtracking would in most cases lead to no better result).

16.3.1.3 The ROOT Keyword

The location term *root* selects the root of the destination document tree; in SGML terms, this is the "document element". Since it ignores any existing location source, the *root* keyword makes sense only as the first location term in the ladder. Since *root* is assumed as the implicit first term in any ladder, the following two location ladders have the same meaning:

```
ROOT DESCENDANT (2 DIV1)
DESCENDANT (2 DIV1)
```

16.3.1.4 The HERE Keyword

The keyword *here* designates the location at which the pointer element itself is situated; it allows extended pointers to select items like "the paragraph immediately preceding the one within which this pointer occurs". Since it ignores any existing location source, this keyword typically makes sense only as the first location term in a location specification.

To designate "the paragraph preceding the current one", the following location ladder could be used:

```
HERE ANCESTOR (1 P) PREVIOUS (1 P)
```

(See below for descriptions of the keywords *ancestor* and *previous*.)

16.3.1.5 The ID Keyword

The resulting location is the element within the destination entity whose ID attribute has the value specified as the location value. The ID location type typically makes sense only as the *first* location pair in a location specification, but there is no syntactic requirement that it be so.

For example,the location specification

```
ID (a27)
```

chooses the necessarily unique element of the destination entity which has an attribute of declared value of type ID, whose value is "a27".

16.3.1.6 The REF Keyword

The resulting location is an element which can be found by interpreting the location value in accordance with document-specific rules for a *canonical reference.* Such reference systems, particularly common in documents of interest to classical and biblical scholars, must also be defined in the TEI header, using the **<refsDecl>** element (see section 5.3.4, "The Reference System Declaration," [in separate fascicle]). If more than one element matches the canonical reference, the first one encountered is chosen.

For example, the location specification

```
REF (MT.2.1)
```

chooses the first element of the destination entity which is identified by the canonical reference “MT.2.1”

16.3.1.7 The CHILD Keyword

The *child* location type specifies an element or span of character data in the document hierarchy using a location value which functions as a *domain-style* address. The value is a series of parenthesized steps, separated by white space. Each such step represents one level of the hierarchy within the location source. Each step may contain up to four parameters separated by white space and interpreted as follows:

1. a signed or unsigned instance number
2. an expression matching an SGML generic identifier
3. an expression matching an SGML attribute name
4. an expression matching an SGML attribute value

In formal terms, the location value of *child* is a series of *steps*:

```
steps    ::=  '(' step ')'
         |    steps '(' step ')'
         ;
step     ::=  signed
         |    signed element
         |    signed element attribute
         |    signed element attribute value
         ;
```

Location values of the same form are also used by the keywords *descendant*, *ancestor*, *previous*, and *next*; details of the interpretation may vary from keyword to keyword.

If an instance number alone is specified, it selects the nth child of the location source. If specified with following parameters, it selects the nth among those children of the location source which satisfy the other parameters. If a negative number is given, the nth child is counted from the last child of the location source to the first. The location source must contain at least n children;[5] if it does not, the *child* term fails.

In formal terms, the first parameter of a *step* is a signed integer:

[5] Strictly speaking, |n| (absolute value of n) children.

```
signed   ::=  NUMBER                        // default sign is +
         |    '+' NUMBER
         |    '-' NUMBER
         ;
```

If a second parameter is given, it is interpreted as an SGML generic identifier, and only elements of the type indicated will be selected. For example, the location specification

```
CHILD (3 DIV1) (4 DIV2) (29 P)
```

chooses the 29th paragraph of the fourth sub-division of the third major division of the initial location source. The location specification

```
CHILD (3 DIV1) (4 DIV2) (-2 P)
```

chooses the next-to-last paragraph of the fourth **<div2>** of the third **<div1>** in the location source.

Constraint by generic identifier is strongly recommended, because it makes links more perspicuous and more robust. It is perspicuous because humans typically refer to things by type: as “the second section”, “the third paragraph”, etc. It is robust because it increases the chance of detecting breakage if (due to document editing) the target originally pointed at no longer exists.

The generic identifier may be specified as a normal SGML name, as a (parenthesized) regular expression, or using the reserved values “#CDATA” or “*”. Regular expressions take the form described below; the location term

```
CHILD (3 (DIV[123])
```

matches the third element which has a generic identifier of “div1”, “div2”, or “div3”. If the generic identifier is specified as “*”, any generic identifier is matched; this means that “CHILD (2 *)” is synonymous with “CHILD (2)”. If the second parameter is “#CDATA”, the location term selects only untagged subportions of an element having SGML mixed content.

The location ladder

```
CHILD (3 #CDATA)
```

thus chooses the third span of character data directly contained by the current location source. If the location source is a paragraph containing

1. a sentence (A)
2. an embedded quotation, marked as a **<q>**
3. another sentence (B)
4. an embedded note, marked as a **<note>**
5. another sentence (C)
6. a second embedded quotation, marked as a **<q>**

where the three sentences A, B, and C are character data enclosed by no element smaller than the paragraph itself, then “CHILD (3 #PCDATA)” selects sentence C, while “CHILD (3)” selects sentence B.

If specified as a name (i.e. without parentheses), the generic identifier is case sensitive if and only if the SGML declaration specifies that generic identifiers are case sensitive (by default they are not). If specified as a regular expression, the expression given is always case sensitive; in the usual case this means the regular expression should be in uppercase, as in the examples here.

In formal terms the second parameter of a *step* is defined thus:[6]

```
element  ::=  NAME
         |    '#CDATA'
         |    '*'
         |    '(' regular ')'
         ;
```

If a third parameter is specified, it is interpreted as an attribute name, and only elements bearing that attribute and satisfying the other constraints will be selected. An element is held to “bear” an attribute if (a) the attribute is defined for that element type and (b) the value of the attribute is not “IMPLIED”. Like the preceding, this parameter may be specified as “*” in the (unlikely) event that an attribute value constitutes a constraint regardless of what attribute name it is a value for. The parameter may also be specified as a parenthesized regular expression.

For example, the location term

```
CHILD (1 * TARGET)
```

selects the first child of the location source for which the attribute *target* has a value. The location term

```
CHILD (1 * (TARGET(S?))
```

will select the first child of the location source for which an attribute called either *target* or *targets* has a value.

As with generic identifiers, attribute names are case sensitive if and only if the SGML declaration says they are; regular expressions are always case sensitive and should usually be uppercased, as shown here.

In formal terms, the third parameter of a tree-traversal step is defined thus:

[6] If regular expressions need not be parenthesized, then this grammar and the accompanying text must be changed. Remove this note before publication.-MSM

```
attribute ::= NAME
          |    '*'
          |    '(' regular ')'
          ;
```

If a fourth parameter is specified, it is interpreted as an attribute value, and only elements satisfying the other constraints and also bearing an attribute of the specified name and value will be selected. The attribute value may be specified exactly as in an SGML document; as a consequence, if the attribute value to be specified contains white space characters, it must be enclosed in quotation marks. The attribute value may also be specified as a regular expression, enclosed in parentheses.

For example, the location specification

```
CHILD (1 * N 2) (1 * N 1)
```

chooses an element using the global *n* attribute. Beginning at the location source, the first child (whatever kind of element it is) with an *n* attribute having the value “2” is chosen; then that element's first direct sub-element having the value “1” for the same attribute is chosen.

The location specification

```
CHILD (1 FS RESP ((lanc|LANC)(s|S|ashire|ASHIRE)))
```

selects the first child of the location source which is an **<fs>** element bearing a *resp* attribute with the value “lancs”, “lancashire”, “LANCS”, or “LANCASHIRE” (as well as other possible combinations which are left to the reader's ingenuity). If specified with quotation marks or as a regular expression, the attribute-value parameter is case-sensitive; otherwise not.

In formal terms, the fourth parameter of a tree-traversal step is defined thus:

```
value    ::=  LITERAL                 // i.e. quoted string.
         |    NAME                    // As for attribute values in
         |    NUMBER                  // document, NMTOKENs need not
         |    NUMTOKEN                // be quoted
         |    '#IMPLIED'
         |    '(' regular ')'
         ;
```

16.3.1.8 The DESCENDANT Keyword

If the *descendant* keyword is used, the location term selects an element or character-data string which is a descendant of the current location source. Like *child*, *descendant* takes as a value a series of one or more parenthesized steps, which may contain the same four parameters described above. The set of elements and strings which may be selected, however, is the set of all descendants of the location source (i.e. the set of all elements contained by it), rather than only the set of immediate children.

The location specification

```
ID (a23) DESCENDANT (2 TERM LANG DE)
```

thus selects the second **<term>** element with a *lang* of “de” occurring within the element with an *id* of “a23”. The search for matching elements occurs in the same order as the SGML data stream; in terms of the document tree, this amounts to a depth-first left-to-right search.

If the instance number is negative, the search is a depth-first left-to-right search, in which the right-most, deepest matching element is numbered -1, etc. The location specification

```
DESCENDANT (-1 NOTE)
```

thus chooses the last **<note>** element in the document.

16.3.1.9 The ANCESTOR Keyword

The *ancestor* location term selects an element from among the direct ancestors of the location source in the document hierarchy. The location value is of the same form as defined for the *child* and *descendant* location types. However, the *ancestor* keyword selects elements from the list of containing elements or “ancestors” of the location source, counting upwards from the parent of the location source (which is ancestor number 1) to the root of the document instance (which is ancestor number -1).

The location source must have at least as many ancestors as the absolute value of the instance number specified as the first parameter of the step. The *ancestor* type thus may not be specified as the first component of a location specification, because the initial location source in effect at that point is the root, which has no ancestors.

For example, the location term

```
ANCESTOR (1 * N 1) (1 DIV)
```

first chooses the smallest element properly containing the location source and having attribute *n* with value “1”; and then the smallest **<div>** element properly containing it. The location term

```
ANCESTOR (1)
```

chooses the immediate parent of the location source, regardless of its type or attributes. The location term

```
ANCESTOR (1 * LANG fr)
```

selects the smallest ancestor for which the *lang* attribute has the value “fr”. The term

```
ANCESTOR (-1 * LANG fr)
```

selects the *largest* ancestor for which the *lang* attribute has the value “fr”. Finally, the term

```
ANCESTOR (1 (DIV[0123456789]?))
```

chooses the smallest **<div>** element of any level which contains the location source.

16.3.1.10 The PREVIOUS Keyword

The *previous* keyword selects an element or character-data string from among those which precede the location source within the same containing element. We speak of the elements and character-data strings contained by the same *parent* element as *siblings*; those which precede a given element or string in the document are its *elder siblings*; those which follow it are its *younger siblings*.

The instance number in the location value of a *previous* term designates the nth elder sibling of the location source, counting from most recent to less recent. The location ladder

```
ID (a23) PREVIOUS (1)
```

thus designates the element immediately preceding the element with an *id* of “a23”. Negative instance numbers also designate elder siblings, counting from the eldest sibling to the youngest. The location source must have at least as many elder siblings as the absolute value of the instance number. If the location source has at least one elder sibling, then the location term

```
PREVIOUS (-1)
```

designates its eldest sibling and is thus synonymous with the ladder

```
ANCESTOR (1) CHILD (1)
```

16.3.1.11 The NEXT Keyword

The keyword *next* behaves like *previous*, but selects from the younger siblings of the location source, not the elder siblings. The location ladder

```
ID (a23) NEXT (1)
```

thus designates the element or string immediately following the element which has an *id* of “a23”. Negative instance numbers also designate younger siblings, counting from the location source to the youngest sibling. The location source must have at least as many younger siblings as the absolute value of the instance number. If the location source has at least one younger sibling, then the location term

```
NEXT (-1)
```

designates its youngest sibling and is thus synonymous with the ladder

```
ANCESTOR (1) CHILD (-1)
```

16.3.1.12 The PATTERN Keyword

The *pattern* keyword selects the *first* place within the location source which matches a pattern-matching expression included as the location value. If more than one location matches that expression, there is no error, but the second and later matches are ignored.

Matching is defined to be case-sensitive, i.e. “abc” is *not* the same as “ABC”. The pattern is expressed as a regular expression in which the following characters have special meanings, similar to those of many Unix programs (such as *grep*) which handle regular expressions:

. match any single character (including white space characters).

[...] match any character from the set enclosed within the brackets. If, however, the first enclosed character is “^”, then match any character *not* from the set enclosed within the brackets. For example, “[^aeiou]” would match any character except a, e, i, o, or u.

**** If the next character is a, d, n, or s, the expression matches any character from a pre-defined group, as shown below; otherwise, the next character is to be taken literally, even if it would otherwise have a special meaning. The special character classes are:

- **\a** any alphabetic character (as defined in the writing system declaration)
- **\d** any digit (0 through 9)
- **\n** any line boundary
- **\s** any white-space character (space, tab, record end, record start)

Note that although \n for newline is provided, its use is discouraged.

***** match zero or more occurrences of the previous regular expression.

+ match one or more occurrences of the preceding regular expression.

? match zero or one occurrences of the preceding regular expression.

^ match the following regular expression only at the beginning of the location source.

$ match the preceding regular expression only at the end of the location source.

| match either the regular expression on the left, or the one on the right.

(...) match the regular expression within the parentheses. (Parentheses are used to control application of the *, ?, +, and | operators, etc.)

For example, the location specification

```
PATTERN (Chapter.8)
```

chooses the first instance of the content string “Chapter” which is followed by any single character and then the digit 8, within the location source. Various elements which contain that location could be selected by following the *pattern* location term with one or more of other types such as *ancestor* (see above).

It is recommended practice to use structure-oriented location types to specify the destination element as narrowly as possible, and then to specify a pattern only within that element context. If element boundaries are encountered within the location source, however, they are ignored and have no effect on the pattern matching operation.

In formal terms, the location value of the *pattern* keyword is defined thus:

```
regs     ::= '(' regular ')'
         |    regs '(' regular ')'
         ;
regular  ::=  character
         |    '.'                            // match any character
         |    '[^' || characters || ']'      // match any char not in list
         |    '[' || characters || ']'       // match any char in list
         |    '\a'                      // match any alphabetic
         |    '\d'                      // match any digit 0-9
         |    '\n'                      // match newline (&#RE;&#RS;)
         |    '\s'                      // match any whitespace character
         |    '\\'                      // match backslash (rev. solidus)
         |    '\' || nonspecial         // match nonspecial character
         |    regular || '*'            // match 0-n of 'regular'
         |    regular || '+'            // match 1-n of 'regular'
         |    regular || '?'            // match 0-1 of 'regular'
         |    '^' || regular            // match at start of loc source
         |    regular || '$'            // match at end of loc source
         |    regular || regular        // match 1st, then 2d regular exp.
         |    regular || '|' || regular // match either 1st or 2d
         |    '(' || regular || ')'     // use parentheses for grouping
         ;

characters ::= /* empty string */
         |    characters character
         ;
nonspecial ::= /* any character except a, d, n, or s */
```

16.3.1.13 The TOKEN Keyword

The *token* keyword selects a sequence of one or more *tokens* chosen from within the character content of the location source, where tokens are counted exactly as for the corresponding HyTime *tokenloc* form. The location value must be either a single positive integer, or a pair of positive integers separated by white space, representing the first and the last token numbers to be included in the resulting location. If two integers are specified, the second must not be less than the first. The location source must contain at least as many tokens as are specified in the location value.

This location type should not be used to count across element boundaries. It is recommended practice to use structure-oriented location types to specify the destination element as narrowly as possible, and then to specify a token location only within that element context. If element boundaries are encountered within the location source, they are ignored.

This location type behaves intuitively only for strings containing an alternating sequence of SGML name-characters and white space; this is the type of string found, for example, in SGML attribute values of type IDREFS, such as “a21 z a13”. For compatibility with the HyTime standard, all characters not included in the class of name characters by the current SGML declaration (by default this includes all punctuation other than the hyphen and full stop) are treated as white space characters.

For example, the location specification

```
ID (a27) TOKEN (3 5)
```

chooses the 3rd, 4th, and 5th tokens from the content of the element whose identifier is “a27”. If this element contained the string “This is _not_ a very good idea”, the target selected would be “not_ a very”.

In formal terms the location value of the *token* and *str* keywords is defined as a range:

```
range    ::=  NUMBER
         |    NUMBER NUMBER
         ;
```

16.3.1.14 The STR Keyword

The *str* keyword identifies a sequence of one or more *characters* chosen from within the character content of the location source, where characters are counted exactly as for the HyTime *strloc* form, which has a corresponding meaning and usage. The location value must be either a single positive integer, or a pair of positive integers separated by white space, indicating the first and the last characters to be included in the resulting location. If two integers are specified, the second must not be less than the first. The location source must have at least as many characters as are specified in the larger of the integers.

This location type should not be used to count across element boundaries. The recommended practice is to use structure-oriented location types to specify the destination element, and then to specify a character location only within that element context. If element boundaries are encountered, however, within the location source, they have no effect.

Character offsets in an SGML document must be counted not from the original source file, but from the output of the SGML parser, (the *element structure information set* or *ESIS*). This is because the rules of SGML allow certain characters to be deleted or expanded transparently.

For example, the location specification

```
ID (a27) STRLOC (3 5)
```

chooses the 3rd 4th and 5th characters of the content of the element having identifier a27. If this element contained the string “This is an even worse idea”, the result would be the string “is ” (i, s and a space).

In multi-byte character sets it is characters which are counted, not bytes. However, in the case of diacritics coded by sequences of bit combinations rather than having separate code points for every combination of letter and diacritic, the diacritics are counted. This means that the following location ladder may retrieve different strings, depending on the system character set in use and on the entity declarations in effect:

```
PATTERN (Wagner's\sG&ouml;tterd&aum;mmerung) STR (10 24)
```

In some character sets, where ö and ä are encoded as single characters, it will select the string “Götterdämmerung”; in others, where they are encoded with distinct characters for umlaut, a, and o, it will select the string “Götterdämmeru”, truncating the last two letters. If a system-dependent definition is used (containing e.g. a printer escape sequence), the results are even less predictable. For this reason, the *str* keyword must be used with caution and should be avoided where possible.

16.3.1.15 The SPACE Keyword

The *space* location term applies to entities which represent graphical or spatio-temporal data; typically such entities are not encoded in SGML, but in one of many specialized graphical formats. SGML provides standard mechanisms (the NOTATION declaration and related constructs) for specifying what format such an entity uses.

The location value for *space* consists of two or three parenthesized parameter lists. The first contains the name of the co-ordinate space in use. The second and third each consist of any number of signed integers. The numbers in a parameter list represent locations along each dimension of a Cartesian co-ordinate space with all axes orthogonal; the length of the list equals the number of dimensions/axes of the space (usually, but not inevitably, 2, 3, or 4).

If the third parameter list is not specified, the location is the single point in the co-ordinate space specified by the second parameter list. If all three parameter lists are specified, the location is the rectangular prism defined by treating corresponding items of the second and third lists as inclusive bounds along each dimension in turn.

The mapping from co-ordinates to physical or display space, and the meaning and ordering of the axes, are not defined by these guidelines. They should be specified in the TEI header unless they can be determined by definition from the format in which the referenced entity is known to be encoded (for example, many graphics formats can only encode locations in units of pixels, counted in a 3 dimensional left-handed co-ordinate space).

Time may be construed as an axis in addition to any others; when it is, it is TEI recommended practice that it be positioned last. The units used must be defined in the TEI header; it is acceptable in certain media (such as videodiscs) to use frame numbers as a surrogate axis for time.

For example,

```
SPACE (2D) (0 0) (1 1)
```

specifies the location of the unit square tangent to the origin in quadrant 1 of a common graph.

The location value for a *space* location term is a NAME enclosed in parentheses, followed by a point pair:

```
pointpair ::= '(' numbers ')'
          |    '(' numbers ')' '(' numbers ')'
          ;
numbers   ::=  NUMBER
          |    numbers NUMBER
          ;
```

16.3.1.16 The FOREIGN Keyword

The *foreign* keyword takes any number of parenthesized parameter lists, and is terminated by the end of the attribute value, or by the next non-parenthesized token, whichever comes first.

The meaning of the *foreign* location term is not defined by these Guidelines. It is intended for use in pointing to special kinds of non-SGML, non-coordinate space data. That is, it should be used for making links to data which cannot be specified using the other mechanisms. The meaning of any *foreign* location types must be specified in the TEI header, in the **<encodingDesc>** element. If more than one such type is used, it is TEI recommended practice that the first parameter list to *foreign* be a name associated with the particular type by documentation in the TEI header.

For example, assume that some program uses a proprietary data format called XFORM, and that the program has supplied an identifier “06286208998” for some piece of data it owns. Then the location specification

```
FOREIGN (XFORM) (06286208998)
```

would be one way of expressing a link to that piece of data.

16.3.1.17 The HYQ Keyword

The *HyQ* keyword takes a single parenthesized parameter lists, which contains an expression in the *HyQ* query language defined by the HyTime standard. See documentation on HyTime and HyQ for definitions of HyQ expressions.

16.3.1.18 The DITTO Keyword

The *ditto* keyword is valid only as the first location term in a ladder, and only within the *to* attribute of an extended pointer element. It designates the location result of the *from* attribute on the same element. Thus in the pointer

```
<xptr from='ID (a23) ANCESTOR (1 DIV[0123]) PATTERN (Wagnerian)'
      to='DITTO PATTERN (Liebestod)'>
```

the *from* attribute designates the first occurrence of the string “Wagnerian” in the **<div>** containing the element with an *id* of “a23”. The *to* attribute designates the first occurrence of the string “Liebestod” which occurs *after* “Wagnerian”, within the same **<div>**. Without the *ditto* keyword, it would be necessary to repeat the entire location ladder of the *from* attribute in the *to* attribute, which would be error-prone for complex expressions.

16.3.2 Using Extended Pointers

As noted above, when only the *from* attribute is specified, the **<xref>** or **<xptr>** element points at the span indicated by *from*. When both *from* and *to* are specified, the element points at the span running from the beginning of the span indicated by the former to the end of the span indicated by the latter. To point at the second, third, and fourth paragraphs of the second chapter (**<div1>**) in the body of the current document, therefore, one may specify either of the following:

```
<xptr from='DESCENDANT (1 body) CHILD (2 div1) (2 p)'
      to = 'DESCENDANT (1 body) CHILD (2 div1) (4 p)' >
<!-- or equivalently: -->
<xptr from='DESCENDANT (1 body) CHILD (2 div1) (2 p)'
      to = 'DITTO NEXT (2 p)'>
```

To point to “the **<term>** occurring in the current **<termEntry>** with attribute $n = 2$”, only the *from* attribute would be required:

```
<xptr from='HERE ANCESTOR (1 TERMENTRY) DESCENDANT (1 TERM N 2)'>
```

The following example demonstrates how elements from two different documents may be combined

```
<xptr id=x1 doc=doc1 from='id (d1.1)'>
<xptr id=x2 doc=doc2 from='id (d2.1) tree (2 *)'>

<ptr id=p1 target='x1 x2'>

<link target='p1 s1 s2'>
```

The first **<xptr>** indicates the element in doc1 which has identifier “d1.1”. The second indicates the second subelement of the element in doc2 which has identifier “d2.1”. These two elements are pointed to as a single item by the **<ptr>** element and given the identifier “p1”. This aggregation, finally, is linked with two other elements both in the current document, with identifiers “s1” and “s2”.

An extended pointer, as described above, may specify as its target only a single destination. Where the intended destination of a link is an aggregation or alignment of destinations, possibly in separate documents, a **<ptr>** or **<join>** element must be used to combine them into a single pointer, as described elsewhere in this chapter. Like any other element, an **<xref>** and **<xptr>** may be given a unique id within the document that contains them. This id value can then be supplied as one of the *target* values for a multi-headed **<ptr>** or **<link>** element, to represent aggregation or linkage respectively.

For example, a modern commentary on an older text must frequently refer to that text, which might well be encoded in a separate SGML document. Some discussions will refer to set of discrete passages in the older text, and will thus require multi-headed pointers. In such a case, the document type declaration must contain a declaration for an SGML entity containing the older text, which might look something like this:

```
<!-- the 1729 Dunciad Variorum -->
<!ENTITY dunciad system 'dunc1729.tei' >
```

In the commentary itself, reference will be made to this external document, using **<xptr>** and **<xref>** elements. When the commentary refers to aggregates of discontiguous passages, **<xptr>** elements are used to point to the individual passage, and a **<ref>** element may refer to these passages as a group by pointing to the **<xptr>**s:

```
<xptr id='xL2.5'   doc=dunciad from='ID (L2.5)'  >
<xptr id='xN1.48'  doc=dunciad from='ID (N1.48)' >
<xptr id='xN1.68'  doc=dunciad from='ID (N1.68)' >
<xptr id='xN1.104' doc=dunciad from='ID (N1.104)'>
<xptr id='xN1.106' doc=dunciad from='ID (N1.106)'>
 ...
<p>In <ref target='xL2.5 xN1.48 xN1.68 xN1.104 xN1.106'
evaluate=all>the references to Theobald</ref>, Pope's satire
characteristically ...</p>
```

If the same discontiguous target is to be referred to repeatedly, it may be convenient to give it a single identifier, thus:

```
<xptr id='xL2.5'   doc=dunciad from='ID (L2.5)'  >
<xptr id='xN1.48'  doc=dunciad from='ID (N1.48)' >
<xptr id='xN1.68'  doc=dunciad from='ID (N1.68)' >
<xptr id='xN1.104' doc=dunciad from='ID (N1.104)'>
<xptr id='xN1.106' doc=dunciad from='ID (N1.106)'>
<ptr id='Theobald' target='xL2.5 xN1.48 xN1.68 xN1.104 xN1.106'>
 ...
<p>In <ref target='Theobald' evaluate=all>the references to
Theobald</ref>, Pope's satire characteristically ...</p>
```

A hypertext web might associate passages of the text and notes with the individuals mentioned, the ancient authors imitated, or thematic content, thus:

```
<xptr id='xL2.5'   doc=dunciad from='ID (L2.5)'  >
<xptr id='xN1.48'  doc=dunciad from='ID (N1.48)' >
<xptr id='xN1.68'  doc=dunciad from='ID (N1.68)' >
<xptr id='xN1.104' doc=dunciad from='ID (N1.104)'>
<xptr id='xN1.106' doc=dunciad from='ID (N1.106)'>
<ptr id='Theorefs' target='xL2.5 xN1.48 xN1.68 xN1.104 xN1.106'>

<xptr id='xN2.3'   doc=dunciad from='ID (N2.3)'  >
<xptr id='xN2.46'  doc=dunciad from='ID (N2.46)' >
<xptr id='xN2.54'  doc=dunciad from='ID (N2.54)' >
<xptr id='xN2.66'  doc=dunciad from='ID (N2.66')'>
<xptr id='xN2.78'  doc=dunciad from='ID (N2.78')'>
<xptr id='xN1.104' doc=dunciad from='ID (N1.104)'>
<ptr id='Curlrefs' target='xN2.3 xN2.46 xN2.54 xN2.66 xN2.78 xN1.104'>

 ...
<div><head>Individuals Named in the Text</head>
<list type=gloss>
<label id=curlname>Curll, Edmund</label>
<item  id=curldesc>A bookseller and publisher ...</item>
 ...
<label id=Theoname>Theobald, Lewis.</label>
<item  id=Theodesc>Attorney, active also as editor and reviewer ...
</item>
 ...
</list>

 ...
<div><head>Ancient Authors Imitated in the Text</head>
<list type=bulleted>
<item id=virgil>Virgil
<item id=homer >Homer
<item id=ovid  >Ovid
 ...
</list>

<ptr id=Curll    target='curlname curldesc'>
<ptr id=Theobald target='theoname theodesc'>
 ...
<linkGrp type=bio>
```

```
<link targets='theobald theorefs'>
<link targets='curl     curlrefs'>
 ...
</linkGrp>
<linkGrp type=imitations>
<link targets='virgil   virgrefs'>
<link targets='homer    homerefs'>
 ...
</linkGrp>
```

16.4 Correspondence

A common problem of text encoding is that of indicating that two or more passages correspond to each other in some way. Provided that corresponding elements bear an *id* attribute, their alignment may be expressed using either the *corresp* attribute described at the beginning of this chapter, or the **<corresp>** and **<correspGrp>** elements:

<correspGrp> : groups a number of **<corresp>** elements which have common properties or function. Attributes include:
  how : indicates how the elements are aligned. Legal values include:
    start : The beginnings of the elements are aligned.
    end : The ends of the elements are aligned.
    mid : The middles of the elements are aligned.
    whole : The whole elements are aligned.
    overlap : The aligned elements overlap.
<corresp> : represents an alignment or correspondence among a group of elements or passages. Attributes include:
  targets : specifies the elements or passages to be aligned, by giving a set of ID values associated with them.
  how : indicates how the passages are aligned. Legal values include:
    start : The beginnings of the passages are aligned.
    end : The ends of the passages are aligned.
    mid : The middles of the passages are aligned.
    whole : The whole passages are aligned.
    overlap : The aligned passages overlap.

Like other pointing elements, these elements are members of the class *pointer* and share the following additional attributes:

type : categorizes the pointer in some respect, using any convenient set of categories.
resp : specifies the creator of the pointer.
crdate : specifies when the pointer was created.
targType : specifies the kinds of elements to which this pointer may point.
targOrder : where more than one identifier is supplied as the value of the *target* attribute, this attribute specifies whether the order in which they are supplied is significant. Legal values include:
  Y : Yes: the order in which IDREFs are specified as the value of a *target* attribute should be followed when combining the targeted elements.

N : No: the order in which IDREFs are specified as the value of a *target* attribute has no significance when combining the targeted elements.

U : Unspecified: the order in which IDREFs are specified as the value of a *target* attribute may or may not be significant.

evaluate : specifies the intended meaning when the target of a pointer is itself a pointer. Legal values include:

all : if the element pointed to is itself a pointer, then the target of that pointer will be taken, and so on, until an element is found which is not a pointer.

one : if the element pointed to is itself a pointer, then its target (whether a pointer or not) is taken as the target of this pointer.

none : no further evaluation of targets is carried out beyond that needed to find the element specified in the pointer's target.

The *corresp* attribute may be specified for any element, provided that the analytic attribute set has been enabled, as described in section 16, "Segmentation and Alignment," on page 3. It takes as its value a string of SGML identifiers, indicating the element or elements which are regarded as corresponding in some sense. As a simple example of its use, consider the following sentence:

```
<title id=S1>Shirley</title>, which made its Friday
night debut only a month ago, was not listed on
<org id=N1>NBC</org>'s new schedule, although
<s id=s2 corresp=N1>the network</s> says
<s id=s3 corresp=S1>the show</s> still is being considered.
```

Here the anaphoric phrases "the network" and "the show" have been associated directly with the elements to which they refer by means of a *corresp* attribute. This mechanism is simple to apply, but has two drawbacks: it is to a large extent arbitrary as to which of the associated elements should bear the *corresp* attribute, and it is not possible to specify more exactly what kind of correspondence is intended. Where this attribute is used, therefore, encoders are encouraged to specify their intent in the associated encoding declarations in the TEI Header.

These drawbacks do not apply to the empty **<corresp>** element, and its associated grouping element **<correspGrp>**. The same example could be encoded using the **<corresp>** element as follows:

```
<title id=S1>Shirley</title> which made its Friday
night debut only a month ago was not listed on
<org id=N1>NBC</org>'s new schedule, although
<s id=s2>the network</s> says
<s id=s3>the show</s> still is being considered.
 ...
<correspGrp type=anaphor>
  <corresp targets='S1 s3'>
  <corresp targets='N1 s2'>
</correspGrp>
```

We consider correspondence to be an equivalence relation, so that all of the following encodings, and some others as well, are semantically equivalent. The first four of these are redundancy free; the fifth one is maximally redundant.

This example shows even more clearly the indeterminate placement of the *corresp* attribute.

```
<!-- Encoding 1 -->
<s id=s1 corresp='s2 s3'> ...
<s id=s2> ...
<s id=s3> ...

<!-- Encoding 2 -->
<s id=s1> ...
<s id=s2 corresp='s1 s3'> ...
<s id=s3> ...

<!-- Encoding 3 -->
<s id=s1> ...
<s id=s2> ...
<s id=s3 corresp='s1 s2'> ...

<!-- Encoding 4 -->
<s id=s1 corresp=s2> ...
<s id=s2 corresp=s3> ...
<s id=s3> ...

<!-- Encoding 5 -->
<s id=s1 corresp='s2 s3'> ...
<s id=s2 corresp='s1 s3'> ...
<s id=s3 corresp='s1 s2'> ...
```

The redundancy and indeterminacy allowed by the *corresp* attribute are eliminated by the **<corresp>** element. All of the hypothetical encodings given above using the *corresp* attribute are equivalent to the following single encoding using the **<corresp>** and **<correspGrp>** elements.

```
<s id=s1> ...
<s id=s2> ...
<s id=s3> ...

<correspGrp>
  <corresp targets='s1 s2 s3'>
</correspGrp>
```

The *corresp* attribute also has the drawback of providing no method of indicating whether the correspondence indicated is true of the whole of the elements being aligned, their starting points, or their ending points. By contrast, the **<corresp>** element has a *how* attribute to specify whether the beginnings of the affected elements are aligned, the ends, the middles, or the whole elements; or whether the alignments merely overlap in some unspecified way.

The element **<correspGrp>** may be used to group together a number of **<corresp>** elements having common values for their *type* or *how* values, to avoid the need to specify them repeatedly.

Here are the formal declarations of the **<correspGrp>** and **<corresp>** elements.

```
<!-- 16.4:  Correspondence                                        -->
<!ELEMENT correspGrp    - -  (corresp)+          +(ptr | xptr) >
<!ATTLIST correspGrp         %a.global;
          how                (start | end | mid | whole |
                             overlap)           whole         >
<!ELEMENT corresp       - O  EMPTY                            >
<!ATTLIST corresp            %a.global;
                             %a.pointer;
          targets            IDREFS             #REQUIRED
          how                (start | end | mid | whole |
                             overlap)           %INHERITED;   >
<!-- This fragment is used in sec. 16                         -->
```

16.4.1 A Detailed Example

The **<corresp>** element may be used to align alternative or parallel versions or translations of the same underlying material. We illustrate by considering three English versions of St. John's Gospel: those in *The Holy Bible: Revised Standard Version* (RSV), in J.B. Phillips, *The New Testament in Modern English* (PNT), and in *The New English Bible* (NEB).

We assume that the text of RSV is broken down into **<div0>** segments containing the Old and New Testaments, **<div1>** segments containing the various books of the Bible, and **<div2>** segments containing the chapters of those books. We assume that **<s>** tags mark verses and orthographic sentences, that all elements other than headings are provided with *id* attributes, and that chapter and verse numbers are recorded as values of the *n* attribute on appropriate elements. Our markup of this version of St. John's Gospel appears schematically as follows (white space has been added liberally to aid legibility).

```
<!-- ... -->
<div0 id=NT><head>The New Testament</head>
<!-- ... -->
<div1 id=JN><head>The Gospel According to John</head>
<div2 id=JN1>
<p id=JN1p1>
        <s id=JN1.1 n=c1v1 type=sent>
        In the beginning was the Word, and the Word
        was with God, and the Word was God.
        </s>
<s id=JN1.2-1.3 type=sent>
        <s id=JN1.2 n=c1v2 type=sent-part>
        He was in the beginning with God:
        </s>
        <s id=JN1.3 n=c1v3 type=sent-part>
        all things were made through him, and
        without him was not anything made
        that was made.
        </s>
```

```
</s>
<!-- ... -->
```

PNT is divided into four parts; we assume that these are encoded as **<div0>** segments. These are made up of one or more books, which we represent as **<div1>** segments. Books are divided into sections, for which PNT provides descriptive heads. Chapter and verse numbers are recorded at the beginnings of those sections; these are the only chapter and verse numbers indicated in PNT. We represent sections as **<div2>** segments, and assign the chapter and verse numbers as the value of the *n* attribute of those elements. Sections are made up of paragraphs, which are in turn made up of orthographic sentences. Our schematic markup of this version of St. John's Gospel is as follows.

```
<div0 id=pgsp><head>The Gospels</head>
<!-- ... -->
<div1 id=pJN><head>The Gospel of John</head>
<div2 id=pJN1 n=c1v1><head>Prologue</head>
<p id=pJN1.1>
        <s id=pJN1.1.1>
        At the beginning God expressed himself.
        </s>
        <s id=pJN1.1.2>
        That personal expression, that word, was with
        God and was God, and he existed with God
        from the beginning.
        </s>
        <s id=pJN1.1.3>
        All creation took place through him,
        and none took place without him.
        </s>
<!-- ... -->
```

Finally, NEB is divided into four parts corresponding exactly to those in PNT; we suppose that these are also encoded as **<div0>** segments. They are also made up of one or more books, which which correspond to those in PNT, and we represent them also as **<div1>** segments. Again, like PNT, books are divided into headed sections, which we represent as **<div2>** segments. However, these are further divided into subsections (without heads), which we represent as **<div3>** segments; these, in turn, are divided into paragraphs and orthographic sentences. Chapter and verse numbers are recorded in the margin; we indicate these as values of the *n* attributes on selected elements. Our schematic markup of this version of St. John's Gospel is as follows.

```
<div0 id=ngsp><head>The Gospel</head>
<!-- ... -->
<div1 id=nJN><head>The Gospel According to John</head>
<div2 id=nJN1 n=c1><head>The Coming of Christ</head>
<div3 id=nJN1.1 n=c1v1>
<p id=nJN1.1.1>
        <s id=nJN1.1.1.1>
        When all things began, the Word already was.
        </s>
        <s id=nJN1.1.1.2>
```

```
            The Word dwelt with God, and what God was,
            the Word was.
            </s>
            <s id=nJN1.1.1.3 n=c1v2>
            The Word, then, was with God at the beginning,
            and through him all things came to be; no single
            thing was created without him.
            </s>
    <!-- ... -->
```

Although the identity of the *n* values specified here indicates (implicitly) the correspondences between these three versions, it suffers from the twin defects that it cannot be validated or processed by an SGML processor and that it is restricted in its granularity. Although all the elements for which the *n* attribute has the value “c1v1” begin at the same relative points in their respective versions, they all end in different places.

To represent correspondences among these three versions explicitly, the analytic *corresp* attribute may be used. For example, to show that the **<div1>** element with identifier “JN” corresponds with both that with identifier “pJN” tag and that with identifier “nJN”, its start-tag can be rewritten as

```
<div1 id=JN corresp='pJN nJN'>
```

In the same way, to show the relation between the first segment in the RSV fragment and corresponding segments in the PNT and NEB fragments, we might rewrite the tag for the RSV segment as follows:

```
<s id=JN1.1 type=sent corresp='pJN1.1.1 nJN1.1.1.1' >
```

However, the correspondence indicated by this encoding differs slightly from the others in that it is only at their *starting* points that the three segments are properly aligned. The content of the element with identifier “JN.1” in fact corresponds not only to the elements indicated, but to one additional elements in each of the other two versions (in PNT to “pjn.1.1.2”, in NEB to “njn1.1.1.2”). As noted above, the *corresp* attribute provides no simple way of indicating whether the correspondence it indicates is true of the whole of the elements being aligned, their starts, or their ends. Thus, the two alignments discussed above may be encoded as follows, to indicate more precisely exactly what aligns:

```
<corresp how=whole targets='JN pJN nJN'>
<corresp how=start targets='JN1.1 pJN1.1.1 nJN1.1.1.1' >
```

These alignments, however, are far from capturing the parallels among the three texts. Because of the structural differences among the translations, there are only a few low-level structural units which correspond as wholes to each other:

- PNT PJN1.1.1.1 “At the beginning, God expressed himself” corresponds to NEB NJN1.1.1.1 “When all things began, the Word already was.”
- RSV JN1.1 “In the beginning ... was God” corresponds to NEB NJN1.1.1.1 and NJN1.1.1.2 “When all things began ... the Word was.”

- RSV JN1.2 and JN1.3 "He was ... was made" correspond to NEB NJN1.1.1.3 "The Word, then, ... without him."

These may be recorded using **<corresp>** thus; note that the RSV / NEB alignments requires two elements of NEB to be joined into a virtual unit; this can be done with **<ptr>**, as here, or with **<join>**, as described in section 16.5, "Aggregation and Virtual Elements," on page 45.

```
<correspGrp how='wholes'>
    <corresp targets='pjn1.1.1 njn1.1.1.1'>
</correspGrp>
<correspGrp how='wholes'>
    <ptr id=njn1v1 target='njn1.1.1.1 njn1.1.1.2'>
    <corresp targets='jn1.1     njn1v1'>
    <corresp targets='jn1.2-1.3 njn1.1.1.3'>
</correspGrp>
```

To capture all the overlaps of the three versions, the **<correspGrp>** element can set *how* to "overlap":

```
<correspGrp how='overlap'>
    <corresp targets='jn1.1  pjn1.1.1  njn1.1.1.1'>
    <corresp targets='jn1.1  pjn1.1.2  njn1.1.1.2'>
    <corresp targets='jn1.2  pjn1.1.2  njn1.1.1.3'>
    <corresp targets='jn1.3  pjn1.1.3  njn1.1.1.3'>
</correspGrp>
```

16.4.2 Alignment Using External Pointers

The preceding encoding of the alignment of parallel passages from three texts requires that those texts and the alignment all be part of the same SGML document. If the texts are in separate documents, then additional **<xptr>** elements must be supplied, as discussed in section 16.3, "External Pointers and References," on page 13. These external pointers may appear anywhere within the document, but if they are created solely for use in encoding correspondences, they may for convenience be grouped within the **<correspGrp>** element that uses them.

To demonstrate this facility, we consider how we might encode the alignments in an extract from Comenius' *Orbis Sensualium Pictus*. Each topic covered in this work has three parts: a picture, a prose text in Latin describing the topic, and a carefully-aligned translation of the Latin into English, German or some other vernacular. Key terms in the two texts are typographically distinct, and are linked to the picture by numbers, which appear in the two texts and within the picture as well.[7]

[7] Our example uses the English translation of Charles Hoole (1659), and is taken from John E. Sadler, ed., *John Amos Comenius Orbis Pictus: a facsimile of the first English edition of 1659* (Oxford: Oxford University Press, 1968) (The Juvenile Library).

First, we present the text portions. The English and Latin portions have been encoded as distinct **<div>** elements. Identifiers have been attached to each typographic line, but no other encoding added, to simplify the example.

```
<!-- English text -->

<div id=E98 lang=EN><head>The Study</head>
<s id=E9801>The Study</s>
<s id=E9802>is a place</s>
<s id=E9803>where a Student,</s>
<s id=E9804>a part from men,</s>
<s id=E9805>sitteth alone,</s>
<s id=E9806>addicted to his Studies,</s>
<s id=E9807>whilst he readeth</s>
<s id=E9808>Books,</s>
 ... </div>

<!-- Latin text -->

<div id=L98 lang=LA><head>Mus&eacute;um</head>
<s id=L9801>Museum</s>
<s id=L9802>est locus</s>
<s id=L9803>ubi Studiosus,</s>
<s id=L9804>secretus ab hominibus,</s>
<s id=L9805>solus sedet,</s>
<s id=L9806>Studiis deditus,</s>
<s id=L9807>dum lectitat</s>
<s id=L9808>Libros,</s>
 ... </div>
```

Next we assume that we have stored a digitized image of the picture itself in some external entity we will call “com98” (for further discussion of the handling of external images and graphics, see section 24, “Graphics, Figures, and Illustrations,” [in separate fascicle]). We further assume that we can address portions of this image as a two-dimensional co-ordinate space. The “SPACE” location method of the **<xptr>** element (discussed in section 6.6, “Simple Cross References,” [in separate fascicle] above) can now be used to point to the whole picture and to two portions of it, one containing the picture of a student and the other of a book, as follows:

```
<xptr n='1' id=p981 doc=com98>
<xptr n='2' id=p982 doc=com98 from='space (2d) (75 5) (133 75)'>
<xptr n='3' id=p983 doc=com98 from='space (2d) (55 42) (90 60)'>
```

Note that each external pointer has its own unique identifier, in addition to the *n* attribute, which last holds the visible label (or “explainer”) used for this image portion in the original.

As printed, the text exhibits three kinds of alignment.

1. The English and Latin portions are printed in two parallel columns, with corresponding phrases, (represented above by **<s>** elements), more or less next to each other.

2. Particular words or phrases are marked as terms in the two languages by a change of rendition: the English text, which otherwise uses black letter type throughout, has the words “The Study”, “a Student” “Studies” and “Books” in a roman font; in the Latin text, which is printed in roman, the corresponding words (Museum, Studiosus, Studiis and Libros) are all in italic.
3. Numbered labels appear within the text portions, linking keywords to each other and to sections of the picture. These labels, which have been left out of the above encoding, are attached to the first third and last segment in each language quoted below, and also appear (rather indistinctly) within the picture itself. If it is desired to transcribe them in the text, they might be encoded using as **<ref>** elements, **<anchor>** elements, or **<xptr>**s to the picture; the number itself would be transcribed as the value of the *n* attribute (or as the content of the **<ref>**).

The first kind of alignment might be represented by using the *corresp* attribute on the **<s>** element. The second kind might be represented by using the **<gloss>** and **<term>** mechanism described in section 6.3.3, “Terms, Cited Words, and Glosses,” [in separate fascicle]. The third kind of alignment might be represented using pointers embedded within the texts, although this would involve some duplication. We choose however to use the **<corresp>** element, since this provides an efficient way of representing the three-way alignment between English, Latin and picture without redundancy.

```
<correspGrp>
        <corresp targets='E9801 L9801 p981'>
        <corresp targets='E9802 L9802     '>
        <corresp targets='E9803 L9803 p982'>
        <corresp targets='E9804 L9804     '>
        <corresp targets='E9805 L9805     '>
        <corresp targets='E9806 L9806     '>
        <corresp targets='E9807 L9807     '>
        <corresp targets='E9808 L9808 p983'>
</corresGrp>
```

This map, of course, only aligns whole segments and image portions, since these are the only parts of our encoding which bear identifiers and can therefore be pointed to. To add to it the correspondence between the typographically distinct words mentioned above, new elements must be defined, either within the text itself or externally by using the extended pointer mechanism. Encoding these word pairs as **<term>** and **<gloss>**, although intuitively obvious, requires a non-trivial decision as to whether the Latin text is glossing the English, or vice-versa. Tagging all the marked words as **<term>** avoids the difficult decision, but might be thought by some encoders to convey the wrong information about the words in question. Simply tagging them as additional embedded **<s>** elements with identifiers that can be aligned like the others is also a possibility. All of these require the addition of further markup to the text. This may pose no problems, or it may be infeasible (e.g. if the text is held on a read-only medium). If it is not feasible to add more markup to the original text, the extended pointer mechanism is likely to be the best choice. For example, to indicate that the words “Studies” and “Studiis” correspond, two external pointers might be defined and aligned as follows:

```
<xptr id=xt981  from='id (s E9806) token (4)'>
<xptr id=xt982  from='id (s L9806) token (1)'>
<corresp target='xt981 xt982'>
```

With the extended-pointer methods just demonstrated, we can provide an even more precise clause-by-clause alignment of the example in section 16.4.1, “A Detailed Example,” on page 36. We can use **<xptr>** elements to identify the relevant portions of each **<s>**, thus providing SGML identifiers for the fine-grained parallels. In the following example, we assume that the three versions are transcribed in documents of with system identifiers 'rsv.tei', 'pnt.tei', and 'neb.tei'. The document type declaration should declare entities with these system identifiers:

```
<!ENTITY RSV system 'rsv.tei'>
<!ENTITY PNT system 'pnt.tei'>
<!ENTITY NEB system 'neb.tei'>
```

Next, we declare extended pointers to the sub-parts of the structural units of the text; the text of RSV is given in comments, to aid the reader in following the example.

```
<!-- First we declare the needed pointers -->
<!-- a. In the beginning was the Word -->
<xptr id=xjn1a doc=RSV
      from='ID (jn1.1) PATTERN (In the .+ the Word)'>

<!-- b. and the Word was with God -->
<xptr id=xjn1b doc=RSV
      from='ID (jn1.1) PATTERN (and the .+ with God)'>
<xptr id=pjn1b doc=PNT
      from='ID (pjn1.1.2) PATTERN (That .+ with God)'>
<xptr id=njn1b doc=NEB
      from='ID (njn1.1.1.2) PATTERN (The .+ with God,)'>

<!-- c. and the Word was God. -->
<xptr id=xjn1c doc=RSV
      from='ID (jn1.1) PATTERN (and the Word was God\.)'>
<xptr id=pjn1c doc=PNT
      from='ID (pjn1.1.2) PATTERN (and was God)'>
<xptr id=njn1c doc=NEB
      from='ID (njn1.1.1.2) PATTERN (and .+ was.)'>

<!-- d. He was in the beginning with God: -->
<xptr id=pjn1d doc=PNT
      from='ID (pjn1.1.2) PATTERN (and he .+ the beginning\.)'>
<xptr id=njn1d doc=NEB
      from='ID (njn1.1.1.3) PATTERN (The .+ beginning;)'>

<!-- e. all things were made through him, -->
<xptr id=xjn1e doc=RSV
      from='ID (jn1.3) PATTERN (all things .+ through him,)'>
<xptr id=pjn1e doc=PNT
      from='ID (pjn1.1.3) PATTERN (All .+ through him,)'>
```

```
<xptr id=njn1e doc=NEB
      from='ID (njn1.1.1.3) PATTERN (and through .+ came to be;)'>

<!-- f. and without him was not anything made that was made. -->
<xptr id=xjn1f doc=RSV
      from='ID (jn1.3) PATTERN (and without .+ was made\.)'>
<xptr id=pjn1f doc=PNT
      from='ID (pjn1.1.3) PATTERN (and .+ without him\.)'>
<xptr id=njn1f doc=NEB
      from='ID (njn1.1.1.3) PATTERN (no single .+ without him\.)'>
```

With the pointers defined, we are in a position to record the correspondences at a fairly fine granularity:

```
<correspGrp how='wholes'>
   <corresp targets='xjn1a  pjn1.1.1 njn1.1.1.1'>
   <corresp targets='xjn1b  pjn1b    njn1b     '>
   <corresp targets='jn1.2  pjn1c    njn1c     '>
   <corresp targets='xjn1d  pjn1d    njn1d     '>
   <corresp targets='xjn1e  pjn1e    njn1e     '>
   <corresp targets='xjn1f  pjn1f    njn1f     '>
</correspGrp>
```

16.4.3 Further Example

As a final example, here is an alignment by wholes of three English versions of the same biblical passage (John 4:6-8.). First the RSV fragment:

```
<!-- RSV fragment -->
<xptr id=JN4.6-4.8 from='id (JN4.6)' to='id (JN4.8)'>
<s id=JN4.6 n=c4v6 type=sent-group>
   <s id=JN4.6.1 type=sent>
      <s id=JN4.6.1.1 type=sent-part>
         Jacob's well was there</s>,
      <s id=JN4.6.1.2 type=sent-part>
         and so Jesus, wearied as he was with
         his journey, sat down beside the well</s>.</s>
   <s id=JN4.6.2 type=sent>
         It was about the sixth hour.</s></s>
<p id=JN4.p2>
<s id=JN4.7 n=c4v7 type=sent-group>
   <s id=JN4.7.1 type=sent>
         <!-- Entity sm indicates a stress mark.  -->
         There came a woman of Samar&sm;ia
         to draw water.</s>
   <s id=JN4.7.2 type=sent>
         Jesus said to her,
         <q>Give me a drink.</q></s></s>
<s id=JN4.8 type=sent>
         For his disciples had gone away into the city
         to buy food.</s>
```

Next, the corresponding fragment from PNT:

```
<xptr id=pJN9.1.3-9.2.1
     from='ID (pJN9.1.3)'
     to = 'ID (pJN9.2.1)'>
<s id=pJN9.1.3 type=sent>
        Jesus, tired with the journey, sat down
        beside it, just as he was.</s>
<s id=pJN9.1.4 type=sent>
        The time was about midday.</s>
<s id=pJN9.1.5 type=sent>
        Presently a Samaritan woman arrived to draw
        some water.</s>
<p id=pJN9.2>
<s id=pJN9.2.1 type=sent>
   <s id=pJN9.2.1.1 type=sent-part>
      <q>Please give me a drink,</q>
      Jesus said to her</s>,
   <s id=pJN9.2.1.2 type=sent-part>
      for his disciples had gone away to
      the town to buy food</s>.</s>
```

The corresponding fragment from NEB:

```
<!-- NEB fragment -->
<xptr id=nJN2.6.1.4-2.6.2.3
      from='ID (nJN2.6.1.4)'
      to = 'ID (nJN2.6.2.3)'>
<s id=nJN2.6.1.4 type=sent>
   <s id=nJN2.6.1.4.1 type=sent-part>
        It was about noon</s>,
   <s id=nJN2.6.1.4.2 type=sent-part>
        and Jesus, tired after his journey,
        sat down by the well</s>.</s>
<p id=nJN2.6.2>
<s id=nJN2.6.2.1 type=sent>
        The disciples had gone away to the town to
        buy food.</s>
<s id=nJN2.6.2.2 type=sent>
        Meanwhile a Samaritan woman came to draw
        water.</s>
<s id=nJN2.6.2.3 type=sent>
        Jesus said to her, <q>Give me a drink.</q></s>
```

And finally, the list of correspondences:

```
<!-- alignment -->
<correspGrp how=whole>
   <corresp targets='JN4.6.1.2 pJN9.1.3   nJN2.6.1.4.2'>
   <corresp targets='JN4.6.2   pJN9.1.4   nJN2.6.1.4.1'>
   <corresp targets='JN4.7.1   pJN9.1.5   nJN2.6.2.2'  >
   <corresp targets='JN4.7.2   pJN9.2.1.1 nJN2.6.2.3'  >
   <corresp targets='JN4.8     pJN9.2.1.2 nJN2.6.2.1'  >
```

```
</correspGrp>
```

If, as is probably the case, the three separate versions are kept in three distinct SGML documents, the alignment cannot be done directly in terms of the identifiers used in each document, since these may not be unique. A set of **<xptr>** elements may be used to address the required elements indirectly as follows, again embedded within the **<correspGrp>** element for convenience:

```
 <correspGrp how=whole>
   <corresp targets='x1 x2 x3'>
   <corresp targets='x4 x5 x6'>
   <corresp targets='x7 x8 x9'>
   <corresp targets='xa xb xc'>
   <corresp targets='xd xe xf'>
      <xptr id=x1 doc=rsv  from='ID (JN.4.6.1.2)'>
      <xptr id=x2 doc=pnt  from='ID (pJN.9.1.3)'>
      <xptr id=x3 doc=neb  from='ID (nJN.2.6.1.4.2)'>
      <xptr id=x4 doc=rsv  from='ID (JN.4.6.2)'>
      <xptr id=x5 doc=pnt  from='ID (pJN.9.1.4)'>
      <xptr id=x6 doc=neb  from='ID (nJN.2.6.1.4.1)'>
      <xptr id=x7 doc=rsv  from='ID (JN.4.7.1)'>
      <xptr id=x8 doc=pnt  from='ID (pJN.9.1.5)'>
      <xptr id=x9 doc=neb  from='ID (nJN.2.6.2.2)'>
      <xptr id=xa doc=rsv  from='ID (JN.4.7.2)'>
      <xptr id=xb doc=pnt  from='ID (pJN.9.2.1.1)'>
      <xptr id=xc doc=neb  from='ID (nJN.2.6.2.3)'>
      <xptr id=xd doc=rsv  from='ID (JN.4.8)'>
      <xptr id=xe doc=pnt  from='ID (pJN.9.2.1.2)'>
      <xptr id=xf doc=neb  from='ID (nJN.2.6.2.1)'>
</correspGrp>
```

16.5 Aggregation and Virtual Elements

Because of the strict hierarchical organization of an SGML document, it is not always possible to enclose all and only all of the parts of a fragmented text segment within a single element. In section 16.1, “Pointers and Links,” on page 5 we introduced the notion of the multi- headed pointer as a general purpose method of pointing to discontinuous segments of this kind. In this section we discuss two methods of representing specific kinds of aggregation, using either a set of special-purpose analytic attributes (enabled as described in section 16, “Segmentation and Alignment,” on page 3), or the special purpose **<join>** element. Both mechanisms require that the elements to be aggregated all bear an *id* attribute, or can be accessed indirectly using the extended pointer mechanism described in section 16.3, “External Pointers and References,” on page 13.

The analytic attributes used for aggregation are *next*, *prev*, *link*, and *partof*:

next : specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately follows it in the aggregate.

prev : specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately precedes it in the aggregate.
link : specifies the identifiers of elements that may be linked with this one to form an aggregate.
partof : specifies the identifier of a **<join>** element in the current document representing an aggregate of which this element is a component part.

The **<join>** element is also used to specify aggregates; it can of course carry more information than can the attributes just named.

<join> : identifies a possibly fragmented segment. Attributes include:
parts : specifies the SGML identifiers of the elements or passages to be aggregated.
type : specifies the kind of element which this aggregation may be understood to represent.

As a simple example of the use of these mechanisms, consider the following short passage:

```
<q id=q1>Owl,</q> said Rabbit shortly,
<q id=q2>You and I have brains. The others have fluff.
If there is any thinking to be done in this
forest &dash and when I say thinking I mean
<emph>thinking</emph> &dash; you and I must do it.</q>
```

This example exhibits at least two discontinuities. The two halves of Rabbit's speech are interrupted by the reporting phrase ("said Rabbit shortly") and his final sentence contains an embedded sentence. The simplest method of resolving the first of these is to use the *link* attribute as follows:

```
<q id=q1 link=q2>Owl,</q> said Rabbit shortly,
<q id=q2>You and I have brains. The others have fluff.
 ....</q>
```

Alternatively, to avoid the ambiguity as to whether the first half of the speech should be linked to the first or the second to the first, we might use the paired *next* and *prev* attributes, as follows:

```
<q id=q1 next=q2>Owl,</q> said Rabbit shortly,
<q id=q2 prev=q1>You and I have brains. The others have fluff.
 ...</q>
```

The same mechanism could be used to indicate a simple segmentation of the rest of the speech:

```
<s id=s1 next=s3>
    If there is any thinking to be done in this forest</s>
<s id=s2>
    &dash and when I say thinking I mean <emph>thinking</emph>
    &dash;</s>
<s id=s3 prev=s1>
    you and I must do it.</s>
```

A second method of indicating that the **<s>** elements with identifiers "s1" and "s3" in the above example may be regarded as forming a single "virtual element" would be to use the **<join>** element as follows:

```
<s id=s1>If there is any thinking to be done in this forest</s>
<s id=s2>&dash and when I say thinking
I mean <emph>thinking</emph> &dash;</s>
<s id=s3>you and I must do it.</s>
 ...
<join type=s parts='s1 s3'>
```

The same mechanism may, *mutatis mutandis*, be used to indicate that the two **<q>** elements in the first example above may be regarded as forming a single "virtual q".

Here is the formal declaration of the **<join>** element.

```
<!-- 16.5:  Virtual elements                                  -->
<!ELEMENT join          - O  EMPTY                              >
<!ATTLIST join               %a.global;
          parts              IDREFS              #REQUIRED
          type               NAME                #IMPLIED       >
<!-- This fragment is used in sec. 16                         -->
```

16.5.1 Extended Example

We now present a more detailed example of the use of these mechanisms to represent an analysis of a passage from J.D. Salinger, *Franny and Zooey*, in which fragments of the two voices of the character Zui-Gan are aggregated in a number of different ways:

```
<p>Zui-Gan called out to himself every day,
<q id=zuiq1 next=zuiq6>Master.</q></p>

<p>Then he answered himself,
<q id=zuiq2 next=zuiq3>Yes, sir.</q></p>

<p>And then he added,
<q id=zuiq3 prev=zuiq2 next=zuiq4>Become sober.</q></p>

<p><q id=zuiq4 prev=zuiq3 next=zuiq5>And after that,</q>
he continued,
<q id=zuiq5 prev=zuiq4>do not be deceived by others.</q>

<p><q id=zuiq6 prev=zuiq1>Yes, sir; yes, sir,</q>
he replied.</p>
```

The *link* attribute can also be used for this purpose; for example, the beginning of the passage can be marked up as follows.

```
<p>Zui-Gan called out to himself every day,
<q id=zuiq1 link=zuiq6>Master.</q></p>
<p>Then he answered himself,
<q id=zuiq2 link='zuiq3 zuiq4 zuiq5'>Yes, sir.</q></p>
```

To create a virtual element that unites the fragmented segments, the **<join>** tag may be used. This element requires a *parts* attribute, which points to the elements which make up such a segment, and may also have a *type* attribute, whose value is the name (in SGML terms, the generic identifier) of the element which the **<join>** can be construed to represent. The value of this attribute may be considered to represent the type of element which the **<join>** would be if it could be encoded directly.

The next example is a re-analysis, using the **<join>** tag, of the preceding markups of the two sets of disconnected quoted elements.

```
<p>Zui-Gan called out to himself every day,
<q id=zuiq1>Master.</q></p>
<p>Then he answered himself, <q id=zuiq2>Yes,
sir.</q></p>
<p>And then he added, <q id=zuiq3>Become sober.
</q></p>
<p><q id=zuiq4>And after that,</q> he continued,
<q id=zuiq5>do not be
deceived by others.</q></p>
<p><q id=zuiq6>Yes, sir; yes, sir,</q> he replied.</p>

<join id=zuiqj1 type=q parts='zuiq1 zuiq6'>
<join id=zuiqj2 type=q parts='zuiq2 zuiq3 zuiq4 zuiq5'>
```

The markup in the preceding example sets pointers from the **<join>** tags to the various **<q>** tags which are its parts. One may also set pointers from the individual fragments to the **<join>** tags by means of the *partof* attribute, thus providing for two-way pointing, as follows.

```
<p>Zui-Gan called out to himself every day,
<q id=zuiq1 partof=zuiqj1>Master.</q></p>
<p>Then he answered himself,
<q id=zuiq2 partof=zuiqj2>Yes, sir.</q></p>
<p>And then he added,
<q id=zuiq3 partof=zuiqj2>Become sober.</q></p>
<p><q id=zuiq4 partof=zuiqj2>
And after that,</q> he continued,
<q id=zuiq5 partof=zuiqj2>do
not be deceived by others.</q></p>
<p><q id=zuiq6 partof=zuiqj1>
Yes, sir; yes, sir,</q> he replied.</p>

<join id=zuiqj1 type=q parts='zuiq1 zuiq6'>
<join id=zuiqj2 type=q parts='zuiq2 zuiq3 zuiq4 zuiq5'>
```

Next, suppose that *id* attributes, for whatever reasons, are not provided for text elements. Then **<xptr>** elements may be created using any of the methods described in section 16.3, “External Pointers and References,” on page 13. These tags, in turn, may be provided with *id* attributes which the *parts* attribute on the **<join>** elements can point to, as in the following example.

```
<p>Zui-Gan called out to himself every day, <q>Master.</q>
<p>Then he answered himself, <q>Yes, sir.</q>
<p>And then he added, <q>Become sober.</q>
<p><q>And after that,</q> he continued,
<q>do not be deceived by others.</q>
<p><q>Yes, sir; yes, sir,</q> he replied.

<join id=zuiqj1 type=q parts='rzuiq1 rzuiq6'>
<join id=zuiqj2 type=q parts='rzuiq2a rzuiq2b rzuiq2c rzuiq2d'>

<xptr id=rzuiq1 from='CHILD (2 div0) (6 div1) (10 text) (1 p) (1 q)'>
<xptr id=rzuiq2 from='CHILD (2 div0) (6 div1) (10 text) (2 p) (1 q)'>
<xptr id=rzuiq3 from='CHILD (2 div0) (6 div1) (10 text) (3 p) (1 q)'>
<xptr id=rzuiq4 from='CHILD (2 div0) (6 div1) (10 text) (4 p) (1 q)'>
<xptr id=rzuiq5 from='CHILD (2 div0) (6 div1) (10 text) (4 p) (2 q)'>
<xptr id=rzuiq6 from='CHILD (2 div0) (6 div1) (10 text) (5 p) (1 q)'>
```

The **<join>** and **<xptr>** elements need not, of course, be in the same SGML document as the text; indeed, if for example, the text is held on a read-only medium this may not be possible. In such a case, they may be stored into some other document, and point into the document containing the text simply by specifying its entity name as the value of the *doc* attribute on each of the **<xptr>** elements:

```
 ...
<!ENTITY fazdoc SYSTEM 'C:\faz.doc'>
 ...

<xptr id=rzuiq1
      doc=fazdoc
      from='CHILD (2 div0) (6 div1) (10 text)
      (1 p) (1 q)'>
 ...
```

The next example is a possible reconstruction of the fragment of the diary that the character Winston Smith is writing throughout the first chapter of *Nineteen-Eighty-Four*, by George Orwell.

```
<div1 id=c1><head rendition='large caps centered'>one</head>
<div2 id=c1s1>
<p>
<s type=orth-sent>It was a bright cold day in April,
and the clocks were striking thirteen. </s>
<!-- ... -->
<s type=orth-sent>In small clumsy letters he wrote:</s>
<q rendition='centered italic' id=d1>
<s type=orth-sent id=d1.1>April 4th, 1984.</s>
```

```
</q>
<p>
<s type=orth-sent>He sat back.</s>
<s type=orth-sent>A sense of complete helplessness had
descended upon him.</s>
<!-- ... -->
<p>
<s type=orth-sent>Suddenly he began writing in sheer
panic, only imperfectly aware of what he was setting
down.</s>
<s type=orth-sent>His small but childish handwriting
straggled up and down the page, shedding first its
capital letters and finally even its full stops:</s>
<q rendition=italic id=d2>
<s type=orth-sent id=d2.1>April 4th, 1984.</s>
<s type=orth-sent id=d2.2>Last night to the flicks. </s>
<!-- ... -->
<s type=phrase id=d2.11.1>then there was a wonderful
shot of a childs arm going up up up right up into
the air </s>
<!-- ... -->
<s type=phrase id=d2.11.9>typical prole reaction</s>
<s type=phrase id=d2.11.10>they never</s>&mdash;
</q>
<p>
<s type=orth-sent>Winston stopped writing, partly
because he was suffering from cramp. </s>
<!-- ... -->
<p><s type=orth-sent>For a moment he was seized
by a kind of hysteria. </s>
<s type=orth-sent>He began writing in
a hurried untidy scrawl: </s>
<q rendition=italic id=d3>
<s type=phrase id=d3.1>theyll shoot me </s>
<s type=phrase id=d3.2>i dont care </s>
<s type=phrase id=d3.3>theyll shoot me
in the back of the neck </s>
<s type=phrase id=d3.4>i dont care </s>
<s type=phrase id=d3.5>down with big brother </s>
<s type=phrase id=d3.6>they always shoot you
in the back of the neck </s>
<s type=phrase id=d3.7>i dont care </s>
<s type=phrase id=d3.8>down with big brother</s>&mdash;
</q>
<p>
<s type=orth-sent>He sat back in his chair,
slightly ashamed of himself, and laid down the pen. </s>
<!-- ... -->
<s type=orth-sent>With the feeling that he was speaking
to O'Brien, and also that he was setting forth an
important axiom, he wrote: </s>
<q rendition=italic id=d15>
<s type=orth-sent id=d15.1>Freedom is the
```

```
freedom to say that two plus two make four. </s>
<s type=orth-sent id=d15.2>If that is granted,
all else follows.</s>
</q>
<div2 id=c1s8><head rendition='large caps centered'>viii
<!-- ... -->

<join n="Winston Smith's diary" id=wsd type=p
      parts='d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14 d15'>

<div1 id=c2><head rendition='large caps centered'>two</head>
<!-- ... -->
```

Part 7

Alphabetic Reference List of Tags

Class Name: **ana**
Description: defines additional analytic attributes which may be enabled for all elements in the TEI encoding scheme.
Attributes:

corresp (i.e. correspondents): specifies the identifiers of corresponding elements.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

link (i.e. link): specifies the identifiers of elements that may be linked with this one to form an aggregate.
Data type: IDREFS
Value: one or more valid SGML identifiers for elements in the current document separated by white space.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

partof: specifies the identifier of a **<join>** element in the current document representing an aggregate of which this element is a component part.
Data type: IDREF
Value: a valid SGML identifier attached to a **<join>** element.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

prev (i.e. previous): specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately precedes it in the aggregate.
Data type: IDREF

Value: a valid SGML identifier for an element in the current document.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

next (i.e. next): specifies the identifier of an element which may be linked with the current element to form an aggregate, and which immediately follows it in the aggregate.
Data type: IDREF
Value: a valid SGML identifier for an element in the current document.
Default value: #IMPLIED
Example:

Remarks:
This attribute is optional.

Part: additional tag set for analysis and interpretation.
Member of classes:
DTD file: teiana2 16, “Segmentation and Alignment,” on page 3

<corresp>

Description: represents an alignment or correspondence among a group of elements or passages.
Attributes:

targets: specifies the elements or passages to be aligned, by giving a set of ID values associated with them.
Data type: IDREFS
Value: Each value specified must be the same as that specified as value for an *ID* attribute for some other element in the current SGML document.
Default value: #REQUIRED
Remarks: If passages to be aligned do not have ID values, an **<xptr>** element may be defined to point at them, and its ID may be specified as one of the targets of **<corresp>**.
This attribute is required.

how: indicates how the passages are aligned.
Data type: (start | end | mid | whole | overlap)
Legal values are:
start : The beginnings of the passages are aligned.
end : The ends of the passages are aligned.
mid : The middles of the passages are aligned.
whole : The whole passages are aligned.
overlap : The aligned passages overlap.

Default value: %INHERITED;
Remarks: The **<corresp>** element inherits its *how* attribute value from that of the **<correspGrp>** containing it (if any). Specify *how* only to override that value.
This attribute is optional.

Example:

```
<corresp targets='jn4.6.1.2 pjn9.1.3 njn2.6.1.4.2'>
```

Part: additional tag set for analysis and interpretation.
Member of classes: loc pointer
DTD file:
Data description: Empty tag.
May occur within: correspGrp
May contain: EMPTY
Element Declaration:

```
<!ELEMENT corresp       - O  EMPTY                         >
```

Attribute List Declaration:

```
<!ATTLIST corresp            %a.global;
                             %a.pointer;
          targets            IDREFS            #REQUIRED
          how                (start | end | mid | whole |
                             overlap)          %INHERITED;    >
```

16.4, “Correspondence,” on page 33
This tag is recommended when applicable.

<correspGrp> (i.e. correspondence group)

Description: groups a number of **<corresp>** elements which have common properties or function.
Attributes:

how: indicates how the elements are aligned.
Data type: (start | end | mid | whole | overlap)
Legal values are:
start : The beginnings of the elements are aligned.
end : The ends of the elements are aligned.
mid : The middles of the elements are aligned.
whole : The whole elements are aligned.
overlap : The aligned elements overlap.
Default value: whole
This attribute is optional.

Example:

```
<correspGrp how=whole>
   <corresp targets='jn4.6.1.2 pjn9.1.3   njn2.6.1.4.2'>
   <corresp targets='jn4.6.2   pjn9.1.4   njn2.6.1.4.1'>
   <corresp targets='jn4.7.1   pjn9.1.5   njn2.6.2.2'  >
   <corresp targets='jn4.7.2   pjn9.2.1.1 njn2.6.2.3'  >
   <corresp targets='jn4.8     pjn9.2.1.2 njn2.6.2.1'  >
</correspGrp>
```

Part: additional tag set for analysis and interpretation.
Member of classes: loc
DTD file:
Data description: One or more corresp elements.
May occur within:
May contain: corresp ptr xptr
Element Declaration:

```
<!ELEMENT correspGrp    - -  (corresp)+           +(ptr | xptr) >
```

Attribute List Declaration:

```
<!ATTLIST correspGrp         %a.global;
          how                (start | end | mid | whole |
                             overlap)            whole          >
```

16.4, “Correspondence,” on page 33
This tag is optional.

<join>

Description: identifies a possibly fragmented segment.
Attributes:

parts: specifies the SGML identifiers of the elements or passages to be aggregated.
Data type: IDREFS
Value: Each value specified must be the same as that specified as value for an *ID* attribute for some other element in the current SGML document.
Default value: #REQUIRED
This attribute is required.

type: specifies the kind of element which this aggregation may be understood to represent.
Data type: NAME
Value: The name of an element in the current DTD.
Default value: #IMPLIED
This attribute is optional.

Example:

```
<s>Southern dialect (my own variety, at least) has only
<list id=lp1>
<item><s>I done gone</s></item>
<item><s>I done went</s></item>
</list>
whereas Negro Non-Standard basilect has both these and
<list id=lp2>
<item><s>I done go</s></item>.
</list></s>
<s>White Southern dialect also has
<list id=lp3>
<item><s>I've done gone</s></item>
<item><s>I've done went</s></item>
</list>
which, when they occur in Negro dialect, should probably be considered
as borrowings from other varieties of English.</s>
<join type=list id=lst1 parts='lp1 lp2 lp3'>
```

Part: additional tag set for analysis and interpretation.
Member of classes:
DTD file:
Data description: Empty element.
May occur within:
May contain: ptr xptr EMPTY
Element Declaration:

```
<!ELEMENT join          - O  EMPTY                             >
```

Attribute List Declaration:

```
<!ATTLIST join               %a.global;
          parts              IDREFS              #REQUIRED
          type               NAME                #IMPLIED       >
```

16.5, “Aggregation and Virtual Elements,” on page 45
This tag is optional.

<link>

Description: defines an association or hypertextual link among elements or passages, of some type not more precisely specifiable by other elements.
Attributes:

targets: specifies the SGML identifiers of the elements or passages to be linked or associated.
Data type: IDREFS
Value: Each value specified must be the same as that specified as value for an *ID* attribute for some other element in the current SGML document.

Default value: #REQUIRED

This attribute is required.

type: categorizes the group of associations in some respect, using any convenient set of categories.
Data type: CDATA
Value: Where type is specified on a **<linkGrp>** element, it supplies the default value for the *type* attribute on each **<link>** included within the group.
Default value: %INHERITED;

This attribute is optional.

Example:

Remarks: This element should only be used to encode associations not otherwise provided for by more specific elements.

The location of this element within a document has no significance, unless it is included within a **<linkGrp>**, in which case it may inherit the value of the *type* attribute from the value given on the **<linkGrp>**.
Part: additional tag set for hypertext and hypermedia.
Member of classes: loc pointer
DTD file:
Data description:
May occur within: linkGrp
May contain: ptr xptr EMPTY
Element Declaration:

```
<!ELEMENT link          - o  EMPTY                              >
```

Attribute List Declaration:

```
<!ATTLIST link               %a.global;
          resp               CDATA              #IMPLIED
          crdate             CDATA              #IMPLIED
          targType           NAMES              #IMPLIED
          targOrder          (Y | N | U)        U
          evaluate           (all | one | none) #IMPLIED
          targets            IDREFS             #REQUIRED
          type               CDATA              %INHERITED;    >
```

16.1, “Pointers and Links,” on page 5
This tag is optional.

<linkGrp>

Description: defines a collection of associations or hypertextual links.

Attributes:

type: categorizes the group of associations in some respect, using any convenient set of categories.
Data type: CDATA
Value: Where type is specified on a **<linkGrp>** element, it supplies the default value for the *type* attribute on each **<link>** included within the group.
Default value: #IMPLIED

This attribute is optional.

Example:

```
<linkGrp type='translation'>
<link target='s1 z1'>
<link target='s2 z2'>
 ...
<link target='s99 z99' type='paraphrase'>
</linkGrp>
```

Remarks: A web or link group is an administrative convenience, which may be used to collect a set of links together for any purpose, not simply to supply a default value for the *type* attribute.
Part: additional tag set for hypertext and hypermedia.
Member of classes:
DTD file:
Data description: May contain one or more **<link>** elements only, optionally with interspersed pointer elements.
May occur within:
May contain: link ptr xptr
Element Declaration:

```
<!ELEMENT linkGrp       - -  (link)+             +(ptr | xptr) >
```

Attribute List Declaration:

```
<!ATTLIST linkGrp            %a.global;
          type               CDATA               #IMPLIED       >
```

16.1, “Pointers and Links,” on page 5
This tag is optional.

Class Name: **loc**
Description: elements used for purposes of location and reference
Attributes: [None: global and inherited attributes only.]

Remarks:
Part: base tag set for common core features.
Member of classes: phrase
DTD file: 6.6, “Simple Cross References,” [in separate fascicle]

Class Name: **phrase**
Description: includes elements which can occur at the level of individual words or phrases.
Attributes: [None: global and inherited attributes only.]
Remarks: This class of elements can occur only within larger elements of the class *inter* or *chunk*. In prose, this means these elements can occur within paragraphs, list items, lines of verse, etc.
Part: base tag set for common core features.
Member of classes:
DTD file: 3.4, “Element Classes and Other Parameter Entities,” [in separate fascicle]

Class Name: **pointer**
Description: elements which point to other elements, using IDREFs.

Attributes:

type: categorizes the pointer in some respect, using any convenient set of categories.
Data type: CDATA
Value: The type should indicate the intended function of the pointer, or the rhetorical relationship between its source and target.
Default value: #IMPLIED

This attribute is optional.

resp : specifies the creator of the pointer.
Data type: CDATA
Value: any string of characters, usually the initials or name of the creator.
Default value: #IMPLIED
This attribute is optional.

crdate : specifies when the pointer was created.
Data type: CDATA
Value: any string representing a date.
Default value: #IMPLIED
This attribute is optional.

targType : specifies the kinds of elements to which this pointer may point.

Data type: NAMES
Value: A list of names of SGML elements defined in the DTD of the current document.
Default value: #IMPLIED
Remarks: If this attribute is supplied, every element specified as a target must be of one or the other of the types specified. An application may choose whether or not to report failures to satisfy this constraint as errors, but may not access an element of the right identifier but the wrong type.

This attribute is optional.

targOrder : where more than one identifier is supplied as the value of the *target* attribute, this attribute specifies whether the order in which they are supplied is significant.
Data type: (Y | N | U)
Legal values are:
- Y : Yes: the order in which IDREFs are specified as the value of a *target* attribute should be followed when combining the targeted elements.
- N : No: the order in which IDREFs are specified as the value of a *target* attribute has no significance when combining the targeted elements.
- U : Unspecified: the order in which IDREFs are specified as the value of a *target* attribute may or may not be significant.

Default value: U

This attribute is optional.

evaluate : specifies the intended meaning when the target of a pointer is itself a pointer.
Data type: (all | one | none)

- all : if the element pointed to is itself a pointer, then the target of that pointer will be taken, and so on, until an element is found which is not a pointer.
- one : if the element pointed to is itself a pointer, then its target (whether a pointer or not) is taken as the target of this pointer.
- none : no further evaluation of targets is carried out beyond that needed to find the element specified in the pointer's target.

Default value: #IMPLIED

Remarks: If no value is given, the application program is responsible for deciding (possibly on the basis of user input) how far to trace a chain of pointers.

This attribute is optional.

Remarks:
Part: base tag set for common core features.
Member of classes:
DTD file: 6.6, “Simple Cross References,” [in separate fascicle]

<ptr>

Description: defines a pointer to another location in the current document.

Attributes:

target: specifies the destination of the pointer as one or more SGML identifiers
Data type: IDREFS
Value: Each value specified must be the same as that specified as value for an *ID* attribute for some other element in the current SGML document.
Default value: #REQUIRED

This attribute is required.

Example:

```
<ptr target="P2 P5" resp=LB type=s>
<ptr target='D1' type='defn'>
```

Remarks:
Part: base tag set for common core features.
Member of classes: loc pointer
DTD file:
May occur within: correspGrp join link linkGrp ptr
May contain: ptr xptr EMPTY
Element Declaration:

```
<!ELEMENT ptr           - o  EMPTY                          >
```

Attribute List Declaration:

```
<!ATTLIST ptr                %a.global;
                             %a.pointer;
          target             IDREFS             #REQUIRED      >
```

6.6, “Simple Cross References,” [in separate fascicle]

This tag is optional.

<ref>

Description: defines a reference to another location in the current document, in terms of one or more identifiable elements, possibly modified by additional text or comment.

Attributes:

target: specifies the destination of the reference as one or more SGML identifiers
Data type: IDREFS
Value: Each value specified must be the same as that specified as value for an *ID* attribute for some other element in the current SGML document.
Default value: #IMPLIED

This attribute is optional.

Example:

```
<ref target="P2 P5" resp=LB type=s>
   See especially the second sentence</ref>.

   See also <ref>s.v. <term>locution</term></ref>.
```

Remarks:
Part: base tag set for common core features.
Member of classes: loc pointer
DTD file:
May occur within:
May contain: %phrase.seq
Element Declaration:

```
<!ELEMENT ref           - -  (%phrase.seq)                      >
```

Attribute List Declaration:

```
<!ATTLIST ref                %a.global;
                             %a.pointer;
          target             IDREFS             #IMPLIED        >
```

6.6, “Simple Cross References,” [in separate fascicle]
This tag is optional.

<xptr> (i.e. extended pointer)

Description: defines a pointer to another location in the current document or an external document.

Attributes:

doc: specifies the document within which the required location is to be found.
Data type: ENTITY
Value: The name of a system entity identifying the document within which the target of this pointer is to be found, by default the current document.
Default value: #IMPLIED

This attribute is required.

from: specifies the start of the destination of the pointer as an expression in the TEI extended pointer notation.
Data type: CDATA
Value: The value specified must be a valid expression in the TEI extended pointer notation defined in section 16.3, "External Pointers and References," on page 13.

Default value: #IMPLIED

Remarks: If no value is specified, the target is the whole of the document identified by the *doc* attribute.

This attribute is optional.

to: specifies the endpoint of the destination of the pointer as an expression in the TEI extended pointer notation.
Data type: CDATA
Value: The value specified must be a valid expression in the TEI extended pointer notation defined in section 16.3, "External Pointers and References," on page 13.

Default value: #IMPLIED

Remarks: This attribute may only be supplied if the *from* attribute is also supplied, in which case the destination is defined to extend from the beginning of the location specified by the *from* attribute, up to the end of the location specified by the *to* attribute. It is an error for *to* to specify a location whose end precedes the beginning of the location specified by *from*; it is not an error for the scopes to overlap.

This attribute is optional.

Example:

Remarks:
Part: additional tag set for analysis and interpretation.
Member of classes: loc pointer
DTD file:
May occur within: correspGrp join link linkGrp ptr
May contain: EMPTY
Element Declaration:

```
<!ELEMENT xptr          - o  EMPTY                         >
```

Attribute List Declaration:

```
<!ATTLIST xptr              %a.global;
                            %a.pointer;
          doc               ENTITY              #IMPLIED
          from              CDATA               #IMPLIED
          to                CDATA               #IMPLIED       >
```

16.3, “External Pointers and References,” on page 13
This tag is optional.

<xref> (i.e. extended reference)

Description: defines a reference to another location in the current document, or an external document, using an extended pointer notation, possibly modified by additional text or comment.

Attributes:

doc: specifies the document within which the required location is to be found.
Data type: ENTITY
Value: The name of a system entity identifying the document within which the target of this pointer is to be found, by default the current document.
Default value: #IMPLIED

This attribute is required.

from: specifies the start of the destination of the pointer as an expression in the TEI extended pointer notation.
Data type: CDATA
Value: The value specified must be a valid expression in the TEI extended pointer notation defined in section 16.3, “External Pointers and References,” on page 13.

Default value: #IMPLIED

Remarks: If no value is specified, the target is the whole of the document identified by the *doc* attribute.

This attribute is optional.

to: specifies the endpoint of the destination of the pointer as an expression in the TEI extended pointer notation.
Data type: CDATA
Value: The value specified must be a valid expression in the TEI extended pointer notation defined in section 16.3, "External Pointers and References," on page 13.

Default value: #IMPLIED

Remarks: This attribute may only be supplied if the *from* attribute is also supplied, in which case the destination is defined to extend from the beginning of the location specified by the *from* attribute, up to the end of the location specified by the *to* attribute. It is an error for *to* to specify a location whose end precedes the beginning of the location specified by *from*; it is not an error for the scopes to overlap.

This attribute is optional.

Example:

Remarks:
Part: additional tag set for analysis and interpretation.
Member of classes: loc pointer
DTD file:
May occur within:
May contain: EMPTY
Element Declaration:

```
<!ELEMENT xref          - o  EMPTY                              >
```

Attribute List Declaration:

```
<!ATTLIST xref               %a.global;
                             %a.pointer;
          doc                ENTITY              #IMPLIED
          from               CDATA               #IMPLIED
          to                 CDATA               #IMPLIED       >
```

16.3, "External Pointers and References," on page 13
This tag is optional.

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