29 Modifying and Customizing the TEI DTD

These Guidelines provide an encoding scheme suitable for encoding a wide range of texts, and capable of being extended in well-defined and documented ways for texts that cannot be conveniently or appropriately encoded using what is provided. This chapter describes how the TEI encoding scheme may be modified and extended. The modification scheme discussed here, like the rest of the TEI scheme, is independent of whether SGML or XML encoding is used. Except for the different syntax of empty elements, a valid Unicode SGML document conforming to the SGML version of the DTD should also be a valid XML document conforming to the XML version of the DTD. Conversion of SGML versions of TEI documents to equivalent XML versions is not discussed here; in general, the only modification required between TEI Interchange Format (as defined in chapter 30 *Rules for Interchange*) and XML is in the syntax of empty elements and the character encoding used.

Formally speaking, these Guidelines provide both syntactic information about how elements and attributes may be used in valid documents and semantic information about what interpretation should be attached to given syntactic constructs. In this sense, they are both a *document type definition* and a *document type declaration*. In the present version of the Guidelines, this semantic information is provided only as informal descriptive prose. Although the descriptions have been written with care, there will inevitably be cases where the intention of the contributors has not been conveyed with sufficient clarity to prevent users of the Guidelines from 'extending' them in the sense of attaching slightly variant semantics to them.

Beyond this unintentional semantic extension, some of the elements described can intentionally be used in a variety of ways; for example, the element `<note>` has an attribute type which can take on arbitrary string values, depending on how it is used in a document. A new type of annotation, therefore, requires no extension to the TEI document type definition.

Furthermore, there are several ways for combining and extending the existing syntactic mechanisms themselves. Earlier chapters have identified these:

- combining the supplied tag sets — the core with one or more base tag sets and additional tag sets — as described in chapter 3 *Structure of the TEI Document Type Definition*;
- documenting how languages are represented using character sets in a document by providing one or more writing system declarations, as described in chapter 25 *Writing System Declaration*;
- extending the intentionally open-ended feature structure mechanism by providing one or more feature system declarations, as described in chapter 26 *Feature System Declaration*.

This chapter explains how the supplied tag sets can be customized by suppressing elements, renaming elements, extending syntactic classes, and adding elements. The different techniques described here have different effects on the level of TEI conformance to be ascribed to a text, as described in chapter 28 *Conformance*.

The TEI DTD is designed to support modification of the tag sets in a documented way that can be validated by a parser. Those wishing to modify the tag sets must do so using *only* the mechanisms described in this chapter if the resulting documents are to be TEI-conformant.

To make possible this range of modifications, the TEI DTD fragments are supplied in a *parameterized* form, in which some content models, element names, and entity values are represented not by fixed form literals but by parameter entity references. In other parts of these Guidelines, the text of the DTD fragments has been presented in a partially *resolved* form, in which the names of elements are given as absolute literals (e.g. 'name', rather than '%n.name;') for the sake of clarity of discussion. Content models presented elsewhere, however, are given in parameterized form throughout (e.g. '%paraContent;'). This is because, in some cases, the actual value of the parameter entity used will differ depending on which tagsets are in use, and also because the modification mechanisms discussed here depend on the ability to redefine these entities. Software tools to automatically derive a fully resolved DTD from a parameterized version of it are widely available, and form a component of any validating SGML or XML parser.[174]

[174] For an example of such a tool, see the TEI pizzachef at `http://www.tei-c.org/pizza.html`.

In the absence of any modification, the TEI core DTD and the additions to it embodied in the base and additional tag sets behave as follows:

1. All the elements described in the relevant sections of these Guidelines are defined.
2. The names of these elements are as given by these Guidelines.
3. The content model of each element is as given by these Guidelines.
4. The attributes of each element and the names and types of these attributes are as given in these Guidelines.
5. The membership of element classes, and the resultant inheritance of attributes are as given in these Guidelines.

The modification mechanisms allow the user to override these defaults in the following ways, while retaining conformance to the TEI Guidelines:

1. The definition of elements may be suppressed, so deleting the associated elements from the modified DTD.
2. The name (generic identifier) associated with an element may be changed, while preserving the semantics of the element. Note, however, that the new name may not clash with the default name of any element defined in these Guidelines.
3. The global teiform attribute should be used to specify the TEI name for such renamed elements.
4. Those parts of the content model of an element which are specified by classes may be extended by adding members to the classes.
5. Further attributes, together with their names and types, may be specified for an individual element and existing attributes for an individual element may be renamed. Note that the new names may not clash with names of existing attributes for the element.
6. Further attributes, together with their names and types, may be specified for the elements in a particular class and those inheriting characteristics from that class.

The modification mechanisms presented in this chapter are quite general, and may be used to make all the types of changes just listed. They can also be used to make more complete modifications of the DTD; if changes other than those listed above are made, however, the resulting text will no longer be TEI conformant.

These Guidelines provide for user modification of the TEI DTD largely by using parameter entities at appropriate points in the DTD. It is not absolutely essential to understand them in detail to modify the TEI DTD (the examples later in this chapter can be followed cookbook fashion), but it will probably prove helpful. An example set of modification files is provided as part of the current release: see further section 29.3 *TEI Lite: an example Customization* below.

Parameter entities are a mechanism for allowing string substitution within markup declarations; they can thus can be used to effect changes in declarations. A parameter entity, unlike an element, may be declared more than once in a DTD; if more than one declaration is given, the parser uses the first one it encounters. Since the declaration subset within the document is read before the external file containing the predefined DTD, an entity declaration in the DTD subset will take precedence over one in the external file. In the TEI DTD, the literal string which defines the model group for some elements, say <p>, is made the value of a parameter entity; the actual element declaration for <p> contains not the literal string itself, but a reference to the parameter entity (in this case, paraContent). If the document's DTD subset contains a declaration for paraContent, this will be used in preference to the standard definition within the external TEI DTD files. The redefinability of parameter entities accounts both for the TEI's use of parameter entities as the vehicle for effecting extensions, and for the separation of entity definitions from other material (to be defined below) that might be needed for certain modifications of the TEI DTD.

Local modifications are most conveniently grouped into two files, one containing modifications to the TEI parameter entities, and the other containing new or changed declarations of elements and their

attributes. Names for these files should be specified by the parameter entities TEI.extensions.ent and TEI.extensions.dtd, by declarations such as the following:

```
<!ENTITY % TEI.extensions.ent SYSTEM 'project.ent' >
<!ENTITY % TEI.extensions.dtd SYSTEM 'project.dtd' >
```

These declarations must be given in the document's DTD subset. The first will be needed for all of the modifications discussed below. The second is only required for the last type of extension described. The parameter entities thus defined are then referenced at an appropriate point in the compiling of the TEI DTD, as further described in section 3.3 *Invocation of the TEI DTD*.

29.1 Kinds of Modification

There are several kinds of modification that can be made to the TEI DTD as follows:

1. deletion of elements;
2. renaming of elements;
3. extension of classes;
4. modification of content models or attribute lists.

These are described in the remaining sections of this chapter.

Each kind of modification changes the set of documents that can be parsed using the DTD. Each combination of the original TEI DTD fragments may be thought of as defining a certain set of documents. Each DTD resulting from a modified set of TEI DTD fragments allows a different set of documents to be parsed. The set of documents parsed by the original DTD may be properly contained in the set of documents parsed by a modified DTD, or vice versa. Modifications that have either of these two results are called *clean modifications* in the remainder of the chapter. Alternatively, the set of documents parsed by the original DTD might overlap the set of documents parsed by the modified DTD with neither being properly contained in the other. Modifications that have this result are called *unclean modifications* in the remainder of the chapter.

29.1.1 Suppressing Elements

The simplest way to modify the supplied tag sets is to suppress one or more of the supplied elements. In the modifiable version of the DTDs, every element declaration is enclosed by a marked section. The marked section is governed by one of the keywords IGNORE or INCLUDE, which is provided indirectly using a parameter entity. This parameter entity has the same name as the generic identifier of the element. Thus, the declaration for the paragraph element, <p>, occurs within a marked section 'guarded' by the parameter entity p:

```
<![ %p; [
<!-- element and attlist declaration for p goes here   -->
]]>
```

The declaration given for these *guard entities* in the modifiable version is INCLUDE in all cases. The construct above is interpreted thus: the first of the three lines is the opening of a marked section; when the parser encounters the section and sees the keyword INCLUDE as its guard (more precisely, sees a parameter entity the value of which is the keyword INCLUDE), the content of the marked section is parsed; the second line of the three is the content of the marked section; and the third line of the three is the closing of a marked section. If the guard is changed to IGNORE, the parser will ignore the content of the marked section.

Thus, to delete the declaration of a generic identifier and thus suppress the element entirely, the entity that provides the guard on the marked section wherein the element declaration appears must simply be set to IGNORE. For example, if the <note> element is not to be used in a particular application, the line

```
<!ENTITY % note 'IGNORE'>
```

should appear somewhere in the DTD prior to its reference in the guard around the declaration for the <note> element. This is achieved by inserting the above declaration in the file which has its name given by the entity TEI.extensions.ent.

Two different cases of deleting one or more elements from the TEI DTD can be identified. The first case involves deleting only elements that are optional wherever they appear in TEI documents. Deleting these is clean in the sense that documents that are parseable with the modified DTD can also be parsed according to the original TEI DTD. To say this another way, the set of documents matching the new DTD is contained in the set of documents matching the original DTD.

The second case involves deleting elements that are required in one or more of their appearances in TEI documents. Deleting these is unclean in that some documents that can be parsed according to the new DTD could not be parsed according to the original TEI DTD. To say this another way, the set of documents matching the new DTD neither contains nor is contained in the set of documents matching the original DTD.

29.1.2 Renaming Elements

In the modifiable version of the TEI DTD, elements are not referred to directly by their generic identifiers; instead, the modifiable version of the DTD makes use of parameter entities that expand to the standard generic identifiers. This allows renaming of elements by redefining the appropriate parameter entities. The names of parameter entities used for naming are formed by taking the standard generic identifier of the element and attaching the string "n." (for "name") as a prefix. Thus, the standard generic identifiers for paragraphs, notes and quotations, <p>, <note>, and <q>, are defined by declarations of the following form:

```
<!ENTITY % n.p        'p'>
<!ENTITY % n.note     'note'>
<!ENTITY % n.soCalled 'soCalled'>
```

These parameter entities are all contained within a file (teigis.ent) which is embedded during the compilation of a TEI DTD. To change the name of an element therefore, all that is needed is to provide an overriding declaration for the appropriate parameter entity. For instance, the following declaration converts <note> to <annotation>:

```
<!ENTITY % n.note 'annotation'>
```

This declaration must be inserted in the file which has its name given by the entity TEI.extensions.ent.

Two different cases of renaming can be identified. The first case involves replacing existing names with names that are otherwise unused in the TEI scheme. (This can be easily checked by looking in the index of the Guidelines.) Such a modification is clean in that the new DTD would still accepts any document accepted by the publication DTD (given the appropriate renaming of elements). The new name cannot possibly conflict with the generic identifier of any other element, since there can be no other occurrences. To say this another way, the set of documents matching the new DTD is isomorphic to the set of documents matching the old DTD. The example given results in a clean modification because there is no element <annotation> specified in these Guidelines. It is also true that any document not using the renamed element which parses under the unmodified DTD will also parse under the modified DTD.

The second case involves introducing a name already used somewhere in a TEI tag set. This is unclean in that it changes what an existing generic identifier means. The name in question could not be declared by any tag set that is used in the document, as it is syntactically invalid to provide two declarations for the same element. The new generic identifier might occur in some TEI tag set not currently included in the DTD used to parse the document. For example, if in some setting the element <note> were assigned the new name <fs> (because, say, notes are used in some technical document to record functional specifications) there might be no immediate problem. If however it was later decided to add the feature structure analysis tag set into the DTD used to parse the document, though, a name clash would occur. There would also be problems in interchanging the resulting documents without confusion.

As a special case, consider translating all of the generic identifiers for all elements into some other language, L. It may be, for example, that the word for "paragraph" in language L begins with the letter "s" and that thus the paragraph element is renamed as <s>. By the definition just given, this would be an unclean modification because an element <s> already exists in the TEI DTD. However, this is clearly not a problem so long as all of the names are redefined at once and that no collisions occur in the new name space: that is, provided that the TEI element <s> is renamed as some other string. This can be

done by a total replacement of the file that contains the entity declarations for the names of the elements. This systematic replacement of names in the DTD must be followed by a systematic use (or replacement) of the new names in the document. To think about this in the terms used earlier, the set of documents matching the new DTD (with all names systematically changed in both the DTD and the documents) is isomorphic to the set of documents matching the original DTD with no names changed (in either the DTD or the documents).

The formal declarations of the parameter entities used for generic identifiers are contained in the file teigis.ent; since their names and replacement texts are fully predictable, these parameter entities are not individually documented in the reference section of these Guidelines. The parameter entity tei.elementNames is used to embed the file teigis.ent in the DTD. A re-declaration for this parameter entity may therefore be used to embed a different version of this file:

```
<!ENTITY % tei.elementNames SYSTEM 'OTAgis.ent' >
```

If an element is renamed using the techniques described here, its declaration for the global TEIform attribute will be left undisturbed; the default value will therefore still be the standard TEI name for the element. TEI-aware application programs can thus process TEI-conformant documents which rename TEI elements, since by consulting the TEIform attribute value the application can learn the standard name for the element and process it accordingly.

In the normal course of events, the value of this attribute will never be specified in a TEI-conformant document; all occurrences will have the default value. In some special circumstances, it can be useful to specify a non-default value on some instances of an element; this allows application programs to process correctly a locally defined element which usually corresponds to one TEI element (which would be expressed by the default value) but sometimes to another TEI element (which would be expressed by explicit values attached to the element instance).

29.1.3 Class Extension

In 3.7.2 *Classes Used in Content Models*, the concept of a class of elements that can appear in the same kinds of structural locations in a document was introduced. An entity is associated with each class named by prefixing the string "m." (for "model") to the name of the class. For example, the value of the parameter entity m.bibl is a list of the members of the class bibl.

In the modifiable version of the TEI DTD, an additional entity is defined for each model class. This additional entity also takes the name of the class, this time prefixed by the string "x." (for "extension"). The default value of these *x-dot entities* is always the empty string. A reference to the corresponding x-dot entity is always included within the replacement string for each m-dot entity. This enables an encoder to add new members to a class simply by declaring a new value for its associated x-dot entity.

For example, the class bibl has the three members <bibl>, <biblFull>, and <biblStruct>. Its content-model entity is defined thus:

```
<!ENTITY % x.bibl '' >
<!ENTITY % m.bibl '%x.bibl; bibl | biblFull | biblStruct' >
```

With the default value of the x-dot entity, this is the same as defining m.bibl with the replacement text `bibl | biblFull | biblStruct`.

An encoder can add an element to the class by providing a new declaration for the x-dot entity. For example, to add a new element called <my.bib>, this definition would be used:

```
<!ENTITY % x.bibl 'my.bib |' >
```

Note that the specification of an x-dot entity must always end with the vertical bar character (for alternation). The definition would be inserted at the appropriate place in the file associated with the entity TEI.extensions.ent. This changes the replacement text of m.bibl from its default value to `my.bib | bibl | biblFull | biblStruct`. If more than one element is to be added to a class, the x-dot entity for the class should be redefined as a list of the new generic identifiers, each one (*including the last*) followed by a vertical bar.

Class extension is always clean in that the set of documents matching the DTD containing the extended class contains all of the documents matching the original DTD. Class extension can imply either the

addition of an existing element to a pre-defined class, or the addition of a new element (as described in the next section) to one.

29.1.4 New content models

Encoders can modify the content models that specify what is contained in an element or set of elements defined by the TEI DTD, modify the attributes of existing elements, or add new elements to the DTD.

Content models or attributes for existing elements are modified in two stages. First, the existing declaration of the element must be deleted in the manner described in the first section of this chapter. Second, a new declaration for the element is given. This new declaration must be inserted in the file associated with the entity TEI.extensions.dtd.

For example, suppose that symbolic designations to be marked with the element <term> can always be associated with a particular source. While the content model of the publication version of the TEI DTD is acceptable, the attribute list needs to be extended. To perform this modification, the following steps must be taken. The declaration

```
<!ENTITY % term 'IGNORE' >
```

must be inserted into the file associated with the entity TEI.extensions.ent. Then a new definition must be inserted into the file associated with the entity TEI.extensions.dtd. In this example, the definition will be the same as that given in 6.3.4 *Terms, Glosses, and Cited Words*, save for the addition of a new attribute.

```
<!ELEMENT term     %om.RR;  (%phrase.seq;)                    >
<!ATTLIST term               %a.global;
          type               CDATA              #IMPLIED
          source             CDATA              #IMPLIED       >
```

New elements are defined by inserting their definitions into the file associated with the entity TEI.extensions.dtd. To be usable, they must somehow be included in the model for some existing element. This can be done either by class extension (which can now be seen to be a restricted, special case of the process defined here) or by redefining the element(s) within which the new element is to be included.

The set of documents matched by the modified DTD and the set of documents matched by the original DTD may be related in several different ways. It is certainly possible that the former could properly include the latter or vice versa; either of these could be said to be clean modifications because the set to be matched has become strictly larger or strictly smaller.

It is also possible that the set of documents matched by the modified DTD is different from the set matched by the original DTD and they may either contain some common documents or have no documents in common; either of these is said to be an unclean modification.

Radical revision is possible. It would be possible to remodel so that the <teiHeader> is not required, or so that it is required but the minimal components described in chapter 5 *The TEI Header* are no longer required, or so that no <text> element is required. In fact, the mechanism, if used in an extreme way, permits deletion of the entire set of TEI definitions and their replacement by an entirely different DTD! Such revisions would result in documents that are not TEI conformant in even the broadest sense, and it is not intended that encoders use the mechanism in this way.

29.2 Documenting the Modifications

When the modification mechanisms are used, their use must be documented. There are two ways in which information about the modifications is recorded.

The first record of the modifications is in the use of the extension files. The file associated with the entity TEI.extensions.ent contains the specifications of the parameter entities that are redefined to accomplish the modifications. This file should be structured in such a way that readers can easily identify any modifications that have been made. The following structure is recommended.

```
<!--  The following elements are deleted     -->
<!--  The following elements are renamed     -->
<!--  The following classes are extended     -->
<!--  The following elements are revised     -->
```

The appropriate parameter entity specifications should be entered after each comment in this file. The order of the comments corresponds to the order of the discussion in this chapter (roughly, from simple to complex). Setting the appropriate entity to IGNORE for each revised element is done in the last section of the file.

The file associated with the entity TEI.extensions.dtd should contain the DTD fragments for new and changed element definitions. The following structure is recommended.

```
<!--  The following declarations define new extensions    -->
<!--    (element and attlist specifications for new tags  -->
<!--    introduced in part 3 of the ent file go here)     -->
<!--  ...                                                 -->
<!--  The following declarations define revised tags      -->
<!--    (element and attlist specifications for tags      -->
<!--    mentioned in part 4 of the ent file go here       -->
<!--  ...                                                 -->
```

These files give a parser sufficient information to implement the modifications and are also useful in providing human readers with some indication of the changes made in the TEI DTD. Full documentation of any additional or modified elements should also be provided, using the ancillary tag set described in chapter 27 *Tag Set Documentation*.

29.3 TEI Lite: an example Customization

Shortly after publication of the first edition of these Guidelines, as a demonstration of how the TEI encoding scheme might be adopted to meet 90% of the needs of 90% of the TEI user community, the TEI editors produced a brief tutorial defining one specific 'clean' modification of the TEI scheme, which they called TEI Lite. This tutorial and its associated DTD became very popular and are available from the TEI web site at http://www.tei-c.org/Lite/. The modification files used to define this view of the TEI are also included among the TEI DTD fragments and may be used as a model for other customizations.

The introductory tutorial on TEI Lite describes in more detail how the tagset was defined, and the rationale underlying its selection of available TEI features. For the present purposes, we note that it required elements taken from the TEI prose base, and also from the TEI additional tag sets for linking, for analysis, and for figures and tables. A small number of additional phrase-level elements not defined in the main TEI scheme were also needed. Finally, large number of the elements made available by this combination of tagsets were not needed, and could be cleanly discarded from the DTD. To effect this, the following doctype declaration (available in the file teilite.dec) was appropriate:

```
<!DOCTYPE TEI.2 PUBLIC "-//TEI P4//DTD Main DTD Driver File//EN" 'tei2.dtd' [
<!ENTITY % TEI.prose    'INCLUDE' >
<!ENTITY % TEI.linking  'INCLUDE' >
<!ENTITY % TEI.analysis 'INCLUDE' >
<!ENTITY % TEI.figures  'INCLUDE' >
<!ENTITY % TEI.XML      'INCLUDE' >
<!ENTITY % TEI.extensions.ent SYSTEM 'teilitex.ent' >
<!ENTITY % TEI.extensions.dtd SYSTEM 'teilitex.dtd' >
]>
```

This declaration is appropriate for an XML customization. If an SGML version is required, the parameter entity TEI.XML should be redefined, by replacing its declaration above with a declaration like the following:

```
<!ENTITY % TEI.XML  "IGNORE">
```

The file teilitex.ent consists largely of declarations like the following:

```
<!ENTITY % TEI.2      'INCLUDE' >
<!ENTITY % teiCorpus.2  'IGNORE' >
```

There is one line for each element potentially available in the tagsets selected, indicating whether it is to be included in the resulting DTD or not. By default, elements are always included, so the first line above is not strictly necessary. However, its inclusion makes it easier for the user of this extension file to see at a glance which elements from the original DTD have been included and which have not.

This file also contains, at its start, declarations for four parameter entities used in defining content models. The first three are needed to add the new (non-TEI) elements declared for this customization into existing model classes:

```
<!ENTITY % x.data      'ident | code | kw |'                  >
<!ENTITY % x.inter     'eg |'                                >
<!ENTITY % x.common    'eg |'                                >
```

As further discussed in section 3.7 *Element Classes*, these declarations have the effect of adding the elements <ident>, <code>, and <kw> to the data model class, and adding the element <eg> to the inter and common classes. Without these declarations, the elements named here would not appear anywhere within the structure of the TEI DTD.

The entity file also redefines the linking attribute class, which is used to add linking attributes to all elements when the TEI.linking tag set is enabled. In TEI lite only a subset of the linking attributes, given by the following definition, is required (compare this declaration with that in the reference section for this class):

```
<!ENTITY % a.linking '
          corresp           IDREFS              #IMPLIED
          next              IDREF               #IMPLIED
          prev              IDREF               #IMPLIED'     >
```

In addition, this entity file contains declarations for a number of commonly used character entity sets (see further chapter 4 *Languages and Character Sets*) and graphic notations (see further section 22.3 *Specific Elements for Graphic Images*). Different sets will be declared, depending on whether the parameter entity TEI.XML is specified as INCLUDE or IGNORE. The XML version of TEI Lite supplies Unicode definitions for these character entities; the SGML version supplies SDATA declarations.

As supplied, the following is typical of the character entity set declarations included for XML:

```
<!ENTITY % ISOlat1
    PUBLIC "-//TEI//ENTITIES Unicode values for ISO 8879 Added Latin 1//EN"
   "iso-lat1.ent">
```

This declaration associates the parameter entity ISOlat1 with a public entity defined by the TEI with the formal identifier "-//TEI//ENTITIES Unicode values for ISO 8879 Added Latin 1//EN. The user may specify the actual location of this entity in a number of ways: the default is to seek a file with the name iso-lat1.ent. This may be over-ridden by supplying an alternative location for the entity set, either by means of another declaration in the DTD subset, or by means of an alternative entry in an associated SGML Open Catalog file. Sample copies of the standard entity sets are available from the TEI web site at http://www.tei-c.org/XML_entities, so one way of modifying the above declaration for an XML DTD might be

```
<!ENTITY % ISOlat1
    PUBLIC "-//TEI//ENTITIES Unicode values for ISO 8879 Added Latin 1//EN"
   "http://www.tei-c.org/XML_Entities/iso-lat1.ent">
```

SGML versions for the same entity sets are also available. These have a different formal public identifier, but the same default system identifier: thus, if the parameter entity TEI.XML has the value IGNORE, the declaration for ISOlat1 will be as follows:

```
<!ENTITY % ISOlat1
    PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN"
   "iso-lat1.ent">
```

Again, the user can override this association by specifying a different system identifier in the DTD subset, or in a local catalog file; sample entity sets for SGML are also available from the TEI web site at http://www.tei-c.org/ISO_Entities, so one way of modifying the above declaration for an SGML DTD might be

```
<!ENTITY % ISOlat1
    PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN"
   "http://www.tei-c.org/ISO_Entities/iso-lat1.ent">
```

Following their declaration, the parameter identifiers for these four entity sets are referenced:

```
%ISOlat1;
%ISOlat2;
%ISOpub; %ISOnum;
```

The remainder of the TEI Lite extension entity file contains declarations for the following commonly used notations:

```
<!NOTATION png PUBLIC
  '-//TEI//NOTATION IETF RFC2083 Portable Network Graphics//EN' >
<!NOTATION jpeg PUBLIC
  'ISO DIS 10918//NOTATION JPEG Graphics Format//EN' >
<!NOTATION tiff PUBLIC
   '-//TEI//NOTATION Aldus Tagged Image File Format//EN'>
<!NOTATION gif PUBLIC
  '-//TEI//NOTATION Compuserve Graphics Interchange Format//EN' >
<!NOTATION sgml PUBLIC
  'ISO 8879:1986//NOTATION Standard Generalized Markup Language//EN' >
<!NOTATION wsd  PUBLIC
  '-//TEI P3-1994//NOTATION Writing System Declaration//EN' >
```

With these declarations in force, the TEI Lite user may embed graphics in PNG, JPEG, TIFF, or GIF format (as further discussed in section 22.3 *Specific Elements for Graphic Images*; the declarations for SGML and for WSD are required to allow for reference to external SGML or WSD documents as further discussed in sections 14 *Linking, Segmentation, and Alignment* and 25 *Writing System Declaration* respectively.

The file teilitex.dtd contains the following declarations for the new elements listed above:

```
<!ELEMENT %n.gi;     %om.RO; (#PCDATA)                          >
<!ATTLIST %n.gi;             %a.global;
          TEI                (yes | no)          'yes'
          TEIform            CDATA               'gi'           >

<!ELEMENT %n.eg;     %om.RR; (#PCDATA)                          >
<!ATTLIST %n.eg;             %a.global;
          TEIform            CDATA               'eg'           >

<!ELEMENT ident      %om.RR; (#PCDATA)                          >
<!ATTLIST ident              %a.global;
                             type CDATA #IMPLIED                >

<!ELEMENT code       %om.RR; (#PCDATA)                          >
<!ATTLIST code               %a.global;                         >

<!ELEMENT kw         %om.RR; (#PCDATA)                          >
<!ATTLIST kw                 %a.global;
                             type CDATA #IMPLIED               >
```

Note that these declarations use the same parameter entities as other parts of the TEI DTD, in particular the parameter entities om.RR and om.RO which make the same extension file usable in both XML or SGML contexts, as further explained in section 3.8.4 *Generation of an XML DTD*, and a.global, which supplies the standard definition for the global attributes.

Two of the elements listed above as 'new' are in fact already defined in the auxiliary tag set for tagset documentation discussed in chapter 27 *Tag Set Documentation*; since, however, the extension mechanism defined here does not allow us to include auxiliary tagsets as such, we have simply copied the definitions for those elements (<gi> and <eg>) from the DTD into our extension file, thus allowing for them to be renamable. The other new elements are simply defined in the same way as any others

To complete the job, full tag descriptions for the new elements added should be provided. Here is a sample description for the <ident> element:

```
<tagDoc><gi>ident</gi>
<rs>identifier</rs>
<desc>contains an identifier in some formal language
(e.g. a variable name); also used for <soCalled>syntactic
variables</soCalled> in syntax diagrams and the like.</desc>
```

```
<attList>
<attDef><attName>type</attName><desc>
indicates the type of identifier.</desc><datatype>CDATA</datatype>
<valList>
<val>fpi</val><desc>formal public identifier</desc>
<val>file</val><desc>operating system filename</desc>
<val>pe</val><desc>parameter entity</desc>
<val>gi</val><desc>generic identifier in SGML or XML</desc>
</valList>
<default>#IMPLIED</default></attDef></attList>
<exemplum>
<eg><![CDATA[<ident type="gi">ident</ident>]]></eg></exemplum><remarks/>
<elemDecl><![CDATA[<!ELEMENT ident %om.RO; (#PCDATA) >]]></elemDecl>
<attlDecl><![CDATA[<!ATTLIST ident %a.global;
                type CDATA #IMPLIED      >]]></attlDecl>
</tagDoc>
```

Further examples of such descriptions are provided in chapter 38 *Sample Tag Set Documentation*.