P5 Fascicule

Lou Burnard, Sebastian Rahtz

ODD SUBSET

1 Documentation Elements

This chapter describes a module which may be used for the documentation of the XML elements and element classes which make up any markup scheme, in particular that described by the TEI Guidelines, and also for the automatic generation of schemas or DTDs conforming to that documentation. It should be used also by those wishing to customize or modify these Guidelines in a conformant manner, as further described in chapters «MD» and «CF» and may also be useful in the documentation of any other comparable encoding scheme, even though it contains some aspects which are specific to the TEI and may not be generally applicable.

An overview of the kind of processing environment envisaged for the module described by this chapter may be helpful. In the remainder of this chapter we refer to software which provides such as a processing environment as an ODD processor[1]. Like any other piece of XML software, an ODD processor may be instantiated in many ways: the current system uses a number of XSLT stylesheets which are freely available from the TEI, but this specification makes no particular assumptions about the tools which will be used to provide an ODD processing environment.

As the name suggests, an ODD processor uses a single XML document to generate multiple outputs. These outputs will include:

- formal reference documentation for elements, attributes, element classes, patterns etc. like those provided in *1.4. Element catalogue*;
- detailed descriptive documentation, embedding some parts of the formal reference documentation, such as the tag description lists provided in this and other chapters of these Guidelines;
- declarative code for one or more XML schema languages, specifically Relax NG or W3C Schema.
- declarative code for fragments which can be assembled to make up an XML Document Type Declaration.

The input required to generate these outputs consists of running prose, and special purpose elements documenting the components (elements, classes, etc.) which are to be documented and also declared in the chosen schema language. All of this input is encoded in XML using the module defined by this chapter. In order to support more than one schema language, this module uses a comparatively high level model which can then be mapped by an ODD processor to the specific constructs appropriate to the schema language in use. Although some modern schema languages such as RelaxNG or W3Schema natively support self-documentary features of this kind, we have chosen to retain the ODD model, if only for reasons of compatibility with earlier versions of these Guidelines. We do however use the ISO standard XML schema language RelaxNG (`http://www.relaxng.org`) as a means of declaring content models, rather than inventing a completely new XML-based representation for them.

In the TEI abstract model, a markup scheme (a schema) consists of a number of discrete modules, which can be combined more or less as required. Each major chapter of these Guidelines defines a distinct module. Each module declares a number of elements specific to that module, and may also populate particular classes. All classes are declared globally, but particular modules extend the range of elements and attributes available by adding new members to existing classes of elements, or by defining new classes. Modules can also declare particular patterns, which act as short-cuts for commonly used content models or class references.

In the present chapter, we discuss the elements needed to support this system. In addition, section *1.1.1. Phrase level documentary elements* discusses some general purpose elements which may be useful in any kind of technical documentation, wherever there is need to talk about technical features of an XML encoding such as element names and attributes. Section *1.1.2. Modules and schemas* discusses the elements which are used to document XML modules and their high level components. Section *1.1.3. Specification elements* discusses the elements which document specific XML elements and their attributes, element classes, and generic patterns or macros. Finally, section *1.1.5. Formal declaration* gives an overview of the whole module.

1.1 Phrase level documentary elements

1.1.1 Phrase level terms

In any kind of technical documentation, the following phrase-level elements may be found useful for marking up parts of a markup language or other formal language when these occur within the body of running text.

[1] ODD is short for 'One Document Does it all', and was the name invented by the original TEI Editors for the predecessor of the system currently used for this purpose. See further Rahtz et al 2002

- **<att>** contains the name of an attribute appearing within running text.

  scheme supplies the name of the scheme in which this name is defined. Sample values include:

  TEI this element is part of the TEI scheme.

  DBK this element is part of the Docbook scheme.

- **<code>** contains literal code

  type the language of the code

- **<eg>** contains a single example demonstrating the use of an element or attribute.

  – No attributes other than those globally available (see definition for tei.global.attributes)

- **<egXML>** contains a single example demonstrating the use of an XML element or attribute.

  – No attributes other than those globally available (see definition for tei.global.attributes)

- **<gi>** contains the name (generic identifier) of an element.

  scheme supplies the name of the scheme in which this name is defined. Sample values include:

  TEI this element is part of the TEI scheme.

  DBK this element is part of the Docbook scheme.

- **<tag>** contains text of a complete start- or end-tag, possibly including attribute specifications, but excluding the opening and closing markup delimiter characters.

  scheme supplies the name of the scheme in which this name is defined. Legal values are:

  TEI this element is part of the TEI scheme.

  DBK this element is part of the Docbook scheme.

- **<val>** contains a single attribute value.

  – No attributes other than those globally available (see definition for tei.global.attributes)

As an example of the recommended use of these elements, we quote from an imaginary TEI working paper:

Example 1.1

```
<p>The <gi>gi</gi> element is used to tag
 element names when they appear in the text; the
 <gi>tag</gi> element however is used to show how a tag as
 such might appear. So one might talk of an occurrence of the
 <gi>blort</gi> element which had been tagged
 <tag>blort type='runcible'</tag>. The
 <att>type</att> attribute may take any name token as
 value; the default value is <val>spqr</val>, in memory of
 its creator.</p>
```

The <code> and <egXML> elements are used to mark strings of formal code, or passages of markup. The first of these is intended for use in citing brief passages in some formal language such as a programming language, as in the following example:

Example 1.2

```
 <p>If the variable <ident>z</ident> has a value of zero, a statement
  such as <code>x=y/z</code> will usually cause a fatal error.</p>
```

If the cited phrase is a mathematical or chemical formula, the more specific <formula> element defined by the figures module («FTFOR») may be more appropriate.

The <eg> element may be used to enclose any kind of example, which will typically be rendered as a distinct block, possibly using particular formatting conventions, when the document is processed. It is a specialised form of the more general <q> element provided by the TEI core module. In documents containing examples of XML markup, the <egXML> element should be used for preference, as further discussed below in «TDEG», since the content of this element can be checked for well-formedness.

These elements are all members of the class tei.oddPhr, which is turn a member of the general phrase class. When this module is included in a schema, this class is expanded to include these elements, and they are therefore available at any point that a phrase level element is available.

1.1.2 Element and attribute descriptions

Within the body of a document using this module, the following elements may be used to reference parts of the specification elements discussed in section *1.1.3. Specification elements*, in particular the brief prose descriptions these provide for elements and attributes.

- **<specList>** marks where a list of descriptions is to be inserted into the prose documentation.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<specDesc>** indicates that a description of the specified element should be included at this point within a tagList.

  key (identifier) supplies the identifier of the documentary element or class for which a description is to be obtained.

  atts (attributes) supplies attribute names for which descriptions should be obtained.

TEI practice requires that a <specList> listing the elements under discussion introduce each subsection of a module's documentation. The source for for the present section, for example, begins as follows:

Example 1.3

```
<div3>
 <head>Element and attribute descriptions</head>
 <p>Within the body of a document using this module, the following
  elements may be used to reference parts of the specification elements
  discussed in section <ptr target="TDcrystals"/>, in particular the
  brief prose descriptions these provide for elements and attributes.
  <specList>
   <specDesc key="specList"/>
   <specDesc key="specDesc"/>
  </specList>
 </p>
 <p>TEI practice requires that a <gi>specList</gi> listing the elements
  ...
  </p><!-- ... -->
</div3>
```

When formatting the <ptr> element in this example, an ODD processor might simply generate the section number and title of the section referred to, perhaps additionally inserting a link to the section. In a similar way, when processing the <specDesc> elements, an ODD processor must recover relevant details of the element being specified (<specList> and <specDesc> in this case) from their associated declaration elements: typically, the details recovered will include a brief description of the element and its attributes. These, and other data, will be stored in a specification element elsewhere within the current document, or they may be supplied by the ODD processor in some other way, for example from a database. For this reason, the link to the required specification element is always made using a TEI-defined key rather than an XML IDREF value. The ODD processor uses this key as a means of accessing the specification element required. There is no requirement that this be performed using the XML ID/IDREF mechanism, but there is an assumption that the identifier be unique.

A <specDesc> generates in the documentation the identifier, and also the contents of the <desc> child of whatever specification element is indicated by its *key* attribute, together with any associated attribute list, as in the example above.

1.1.3 Formal definitions

The elements discussed in this section are formally defined as follows: [1.1] **att**:

```
att = element att { att.content, att.attributes }

att.content = text

att.attributes =
   tei.global.attributes,
   att.attributes.scheme,
   [ a:defaultValue = "att" ] attribute TEIform { text }?,
   empty

att.attributes.scheme =
```

```
   [ a:defaultValue = "TEI" ]
   attribute scheme { att.attributes.scheme.content }?

att.attributes.scheme.content = text

tei.oddPhr |= att
```

code:

```
code = element code { code.content, code.attributes }

code.content = text

code.attributes =
   tei.global.attributes,
   code.attributes.type,
   [ a:defaultValue = "code" ] attribute TEIform { text }?,
   empty

code.attributes.type = attribute type { code.attributes.type.content }?

code.attributes.type.content = text

tei.oddPhr |= code
```

eg:

```
eg = element eg { eg.content, eg.attributes }

eg.content = text

eg.attributes =
   tei.global.attributes,
   [ a:defaultValue = "eg" ] attribute TEIform { text }?,
   empty

tei.common |= eg
tei.data |= eg
```

egXML:

```
egXML = element id2290837:egXML { egXML.content, egXML.attributes }

egXML.content = text

egXML.attributes =
   tei.global.attributes,
   [ a:defaultValue = "egXML" ] attribute TEIform { text }?,
   empty

tei.common |= egXML
tei.data |= egXML
```

gi:

```
gi = element gi { gi.content, gi.attributes }

gi.content = text

gi.attributes =
   tei.global.attributes,
```

```
   gi.attributes.scheme,
   [ a:defaultValue = "gi" ] attribute TEIform { text }?,
   empty

gi.attributes.scheme =
   [ a:defaultValue = "TEI" ] attribute scheme { gi.attributes.scheme.content }?

gi.attributes.scheme.content = text

tei.oddPhr |= gi
```

ident:

```
ident = element ident { ident.content, ident.attributes }

ident.content = text

ident.attributes =
   tei.global.attributes,
   ident.attributes.type,
   [ a:defaultValue = "ident" ] attribute TEIform { text }?,
   empty

ident.attributes.type = attribute type { ident.attributes.type.content }?

ident.attributes.type.content = text

tei.oddPhr |= ident
```

tag:

```
tag = element tag { tag.content, tag.attributes }

tag.content = text

tag.attributes =
   tei.global.attributes,
   tag.attributes.scheme,
   [ a:defaultValue = "tag" ] attribute TEIform { text }?,
   empty

tag.attributes.scheme =
   [ a:defaultValue = "TEI" ]
   attribute scheme { tag.attributes.scheme.content }?

tag.attributes.scheme.content = text

tei.oddPhr |= tag
```

val:

```
val = element val { val.content, val.attributes }

val.content = text

val.attributes =
   tei.global.attributes,
   [ a:defaultValue = "val" ] attribute TEIform { text }?,
   empty

tei.oddPhr |= val
```

specList:

```
specList = element specList { specList.content, specList.attributes }

specList.content = specDesc+

specList.attributes =
   tei.global.attributes,
   [ a:defaultValue = "specList" ] attribute TEIform { text }?,
   empty

tei.oddPhr |= specList
```

specDesc:

```
specDesc = element specDesc { specDesc.content, specDesc.attributes }

specDesc.content = empty

specDesc.attributes =
   tei.global.attributes,
   specDesc.attributes.key,
   specDesc.attributes.atts,
   [ a:defaultValue = "specDesc" ] attribute TEIform { text }?,
   empty

specDesc.attributes.key = attribute key { specDesc.attributes.key.content }?

specDesc.attributes.key.content = xsd:NCName

specDesc.attributes.atts = attribute atts { specDesc.attributes.atts.content }?

specDesc.attributes.atts.content = text

tei.oddPhr |= specDesc
```

1.2 Modules and schemas

As mentioned above, the primary purpose of this module is to facilitate the documentation of an XML schema derived from the TEI Guidelines. The following elements are provided for this purpose:

- **<schemaSpec>** generates a TEI-conformant schema and documentation for it.

  type type of schema

  start specifies entry points to the schema, i.e. which elements are allowed to be used as the root of documents conforming to it.

  namespace specifies the default namespace (if any) applicable to components of the schema.

- **<moduleSpec>** documents the structure, content, and purpose of a single module, i.e. a named and externally visible group of declarations.

  type type of module to be generated

- **<moduleRef>** references a module which is to be incorporated into a schema.

- **<specGrp>** contains any convenient grouping of specifications for use within the current module.

  – No attributes other than those globally available (see definition for tei.global.attributes)

- **<specGrpRef>** indicates that the declarations contained by the specGrp referenced should be inserted at this point.

  target points at the specification group which logically belongs here.

A module is a convenient way of grouping together element and other declarations and associating an externally-visible name with the group. A specification group performs essentially the same function, but the resulting group is not accessible outside the scope of the ODD document in which it is defined, whereas a module can be accessed by name from any TEI schema. Modules, elements and their attributes, element classes, and patterns are all individually documented using further elements described in section *1.1.3. Specification elements* below; part of that specification includes the name of a module to which the component belongs. An ODD processor generating XML DTD or schema fragments from a document marked up according to the recommendations of this chapter will generate such fragments for each <module> element found. For example, the chapter documenting the TEI module for names and dates contains a module specification like the following:

Example 1.14

```
<moduleSpec id="DND" ident="namesdates">
 <gloss>Additional tags for names and dates</gloss>
</moduleSpec>
```

together with specifications for all the elements, classes, and patterns which make up that module, expressed using <elementSpec>, <classSpec> or <patternSpec> elements as appropriate. (These elements are discussed in section *1.1.3. Specification elements* below) Each of those specifications carries a module attribute, the value of which is namesdates. An ODD processor encountering the <moduleSpec> element above can thus generate a schema fragment for the TEI namesdates module that includes declarations for all the elements (etc.) which reference it.

In most realistic applications, it will be desirable to combine more than one module together to form a complete schema. A schema consists of references to one or more modules or specification groups, and may also contain explicit declarations or redeclarations of elements (see further *1.1.4. Building a schema*). Any combination of modules can be used to create a schema: the distinction between base and additional tagsets in earlier versions of the TEI scheme has not been carried forward into P5.

A schema can combine references to TEI modules with references to other (non-TEI) modules using different namespaces, for example to include mathematical markup expressed using MathML in a TEI document. By default, the effect of combining modules is to allow all of the components declared by the constituent modules to coexist (where this is syntactically possible: where it is not — for example, because of name clashes — a schema cannot be generated). It is also possible to over-ride declarations contained by a module, as further discussed in section *1.1.4. Building a schema*

It is often convenient to describe and operate on sets of declarations smaller than the whole, and to document them in a specific order: such collections are called specGrps (specification groups). Individual <specGrp> elements are identified using the global *id* attribute, and may then be referenced from any point in an ODD document using the <specGrpRef> element. This is useful if, for example, it is desired to describe particular groups of elements in a specific sequence. Note however that the order in which element declarations appear within the schema code generated from a <moduleSpec> element cannot be altered, and is not affected by the order of declarations within a <specGrp>.

An ODD processor will generate a piece of schema code corresponding with the declarations contained by a <specGrp> element in the documentation being output, and a cross reference to such a piece of schema code when processing a <specGrpRef>. For example, if the input text reads

Example 1.15

```
<p>This module contains three red elements:
 <specGrp id="RED">
  <elementDecl ident="beetroot"><!-- ... -->
  </elementDecl>
  <elementDecl ident="east"><!-- ... -->
  </elementDecl>
  <elementDecl ident="rose"><!-- ... -->
  </elementDecl>
 </specGrp>
 and two blue ones:
 <specGrp id="BLUE">
  <elementDecl ident="sky"><!-- ... -->
  </elementDecl>
  <elementDecl ident="bayou"><!-- ... -->
  </elementDecl>
 </specGrp>
</p>
```

then the output documentation will replace the two <specGrp> elements above with a representation of the schema code declaring the elements <beetroot>, <east>, <rose> and that declaring the elements <sky> and <bayou> respectively. Similarly, if the input text contains elsewhere a passage such as

Example 1.16

```
<div>
 <head>An overview of the imaginary module</head>
 <p>The imaginary module contains declarations for coloured things:
  <specGrpRef target="RED"/>
  <specGrpRef target="BLUE"/>
 </p>
</div>
```

then the <specGrpref> elements may be replaced by an appropriate piece of reference text such as 'The RED elements were declared in section 4.2 above', or even by a copy of the relevant declarations. As stated above, the order of declarations within the imaginary module described above will not be affected in any way. Indeed, it is possible that the imaginary module will contain declarations not present in any specification group, or that the specification groups will refer to elements that come from different modules. Specification groups are always local to the document in which they are defined, and cannot be referenced externally (unlike modules).

1.2.1 DTD specific elements

The following deprecated elements are also currently defined in P5 ODDs for compatability reasons only. They will be removed as soon as we find out how to do without them when generating SGML dtds.

- **<stringVal>** contains the intended expansion for the entity documented by an patternSpec element, enclosed by quotation marks.
  - No attributes other than those globally available (see definition for tei.global.attributes)

1.2.2 Formal definitions

The elements discussed in this section are formally defined as follows: [1.2] **moduleRef**:

```
moduleRef = element moduleRef { moduleRef.content, moduleRef.attributes }

moduleRef.content = empty

moduleRef.attributes =
   tei.global.attributes,
   ( moduleRef.attributes.key | moduleRef.attributes.url )?,
   [ a:defaultValue = "moduleRef" ] attribute TEIform { text }?,
   empty

moduleRef.attributes.key = attribute key { moduleRef.attributes.key.content }

moduleRef.attributes.key.content = xsd:NCName

moduleRef.attributes.url = attribute url { moduleRef.attributes.url.content }

moduleRef.attributes.url.content = xsd:anyURI

tei.oddRef |= moduleRef
```

moduleSpec:

```
moduleSpec = element moduleSpec { moduleSpec.content, moduleSpec.attributes }

moduleSpec.content = macro.glossSeq, exemplum*, remarks?, listRef*

moduleSpec.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   moduleSpec.attributes.type,
   [ a:defaultValue = "moduleSpec" ] attribute TEIform { text }?,
   empty

moduleSpec.attributes.type =
```

```
   attribute type { moduleSpec.attributes.type.content }?

moduleSpec.attributes.type.content = text

tei.oddDecl |= moduleSpec
tei.identifiable |= moduleSpec
```

schemaSpec:

```
schemaSpec = element schemaSpec { schemaSpec.content, schemaSpec.attributes }

schemaSpec.content = macro.glossSeq, ( moduleRef | specGrpRef | tei.oddDecl )*

schemaSpec.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   schemaSpec.attributes.type,
   schemaSpec.attributes.start,
   schemaSpec.attributes.namespace,
   [ a:defaultValue = "schemaSpec" ] attribute TEIform { text }?,
   empty

schemaSpec.attributes.type =
   attribute type { schemaSpec.attributes.type.content }?

schemaSpec.attributes.type.content = text

schemaSpec.attributes.start =
   [ a:defaultValue = "TEI" ]
   attribute start { schemaSpec.attributes.start.content }?

schemaSpec.attributes.start.content = xsd:NMTOKENS

schemaSpec.attributes.namespace =
   [ a:defaultValue = "http://www.tei-c.org/ns/1.0" ]
   attribute namespace { schemaSpec.attributes.namespace.content }?

schemaSpec.attributes.namespace.content = text

tei.oddPhr |= schemaSpec
tei.identifiable |= schemaSpec
```

specGrp:

```
specGrp = element specGrp { specGrp.content, specGrp.attributes }

specGrp.content = ( tei.oddDecl | tei.oddRef | tei.chunk )*

specGrp.attributes =
   tei.global.attributes,
   [ a:defaultValue = "specGrp" ] attribute TEIform { text }?,
   empty

tei.oddDecl |= specGrp
```

specGrpRef:

```
specGrpRef = element specGrpRef { specGrpRef.content, specGrpRef.attributes }

specGrpRef.content = empty

specGrpRef.attributes =
```

```
   tei.global.attributes,
   specGrpRef.attributes.target,
   [ a:defaultValue = "specGrpRef" ] attribute TEIform { text }?,
   empty

specGrpRef.attributes.target =
   attribute target { specGrpRef.attributes.target.content }

specGrpRef.attributes.target.content = xsd:IDREF

tei.oddRef |= specGrpRef
```

stringVal:

```
stringVal = element stringVal { stringVal.content, stringVal.attributes }

stringVal.content = text

stringVal.attributes =
   tei.global.attributes,
   [ a:defaultValue = "stringVal" ] attribute TEIform { text }?,
   empty
```

1.3 Specification elements

The following elements are used to specify elements, classes, and patterns for inclusion in a given module:

- **<elementSpec>** documents the structure, content, and purpose of a single element type.

  ns (namespace) specifies the namespace to which this element belongs

  usage specifies the optionality of an attribute or element. Legal values are:

  req required
  mwa mandatory when applicable
  rec recommended
  rwa recommended when applicable
  opt optional

- **<classSpec>** contains reference information for a TEI element class; that is a group of elements which appear together in content models, or which share some common attribute, or both.

  type indicates whether this is a model class, an attribute class, or both. Legal values are:

  model members of this class appear in the same content models
  atts members of this class share common attributes
  both members of this class share attributes and also appear in the same content models

- **<macroSpec>** documents the function and implementation of a pattern.

  type indicates which type of entity should be generated, when an ODD processor is generating a module using SGML syntax. Legal values are:

  pe parameter entity
  epe element parameter entity
  ge general entity
  dt datatype entity

Unlike most elements in the TEI scheme, each of these elements has a fairly rigid internal structure consisting of a large number of child elements which are always presented in the same order. For this reason, we refer to them informally as 'crystals'. Furthermore, since these elements all describe markup objects in broadly similar ways, they have several child elements in common. In the remainder of this section, we discuss first the elements which are common to all the specification elements, and then those which are specific to a particular type.

Specification elements may appear at any point in an ODD document, both between and within paragraphs as well as inside a `<specGrp>` element, but the specification element for any particular component may only appear once (except in the case where a modification is being defined; see further *1.1.4. Building a schema*). The order

in which they appear will not affect the order in which they are presented within any schema module generated from the document. In documentation mode, however, an ODD processor will output the schema declarations corresponding with a specification element at the point in the text where they are encountered, provided that they are contained by a <specGrp> element, as discussed in the previous section. An ODD processor will also associate all declarations found with the nominated module, thus including them within the schema code generated for that module, and it will also generate a full reference description for the object concerned in a catalogue of markup objects. These latter two actions always occur irrespective of whether or not the declaration is included in a <specGrp>.

These elements are formally declared as follows:

[1.3] **elementSpec**:

```
elementSpec =
   element elementSpec { elementSpec.content, elementSpec.attributes }

elementSpec.content =
   macro.glossSeq, classes?, content?, attList?, exemplum*, remarks?, listRef*

elementSpec.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   elementSpec.attributes.ns,
   elementSpec.attributes.usage,
   [ a:defaultValue = "elementSpec" ] attribute TEIform { text }?,
   empty

elementSpec.attributes.ns =
   [ a:defaultValue = "http://www.tei-c.org/ns/1.0" ]
   attribute ns { elementSpec.attributes.ns.content }?

elementSpec.attributes.ns.content = text

elementSpec.attributes.usage =
   [ a:defaultValue = "opt" ]
   attribute usage { elementSpec.attributes.usage.content }?

elementSpec.attributes.usage.content = "req" | "mwa" | "rec" | "rwa" | "opt"

tei.oddDecl |= elementSpec
tei.identifiable |= elementSpec
```

classSpec:

```
classSpec = element classSpec { classSpec.content, classSpec.attributes }

classSpec.content =
   macro.glossSeq, classes?, attList?, exemplum*, remarks?, listRef*

classSpec.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   classSpec.attributes.type,
   [ a:defaultValue = "classSpec" ] attribute TEIform { text }?,
   empty

classSpec.attributes.type =
   attribute type { classSpec.attributes.type.content }

classSpec.attributes.type.content = "model" | "atts" | "both"

tei.oddDecl |= classSpec
tei.identifiable |= classSpec
```

macroSpec:

```
macroSpec = element macroSpec { macroSpec.content, macroSpec.attributes }

macroSpec.content =
   macro.glossSeq, ( stringVal | content )+, exemplum*, remarks?, listRef*

macroSpec.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   macroSpec.attributes.type,
   [ a:defaultValue = "macroSpec" ] attribute TEIform { text }?,
   empty

macroSpec.attributes.type =
   attribute type { macroSpec.attributes.type.content }?

macroSpec.attributes.type.content = "pe" | "epe" | "ge" | "dt"

tei.oddDecl |= macroSpec
tei.identifiable |= macroSpec
```

1.3.1 Common elements

This section discusses the child elements common to all of the specification elements. These child elements are used to specify the naming, description, exemplification, and classification of the specification elements.

Description of components

- **<remarks>** contains any commentary or discussion about the usage of an element, attribute, class, or entity not otherwise documented within the containing element.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<listRef>** supplies a list of significant references to places where this element is discussed, in the current document or elsewhere.
  - No attributes other than those globally available (see definition for tei.global.attributes)

A <desc> element is used to provide a brief characterization of the intended function of the element, class, value etc. being documented.

The <remarks> element contains any additional commentary about how the item concerned may be used, details of implementation-related issues, suggestions for other ways of treating related information etc., as in the following example:

Example 1.24

```
<elementSpec module="core" ident="foreign"><!--... --><remarks>
  <p>This element is intended for use only where no other element
   is available to mark the phrase or words concerned.  The global
   <att>xml:lang</att> attribute should be used in preference to this element
   where it is intended to mark the language of the whole of some text
   element.</p>
  <p>The <gi>distinct</gi> element may be used to identify phrases
   belonging to sublanguages or registers not generally regarded as true
   languages.</p>
 </remarks><!--... -->
</elementSpec>
```

A specification element will usually conclude with a list of references, each tagged using the standard <ptr> or <xptr> elements, and grouped together into a <listRef> element: in the case of the <foreign> element discussed above, the list is as follows:

Example 1.25

```
<listRef>
 <ptr target="COHQHF"/>
</listRef>
```

Exemplification of components

- **<exemplum>** contains a single example demonstrating the use of an element, together with optional paragraphs of commentary.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<eg>** contains a single example demonstrating the use of an element or attribute.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<egXML>** contains a single example demonstrating the use of an XML element or attribute.
  - No attributes other than those globally available (see definition for tei.global.attributes)

The <exemplum> element is used to combine a single illustrative example with an optional paragraph of commentary following or preceding it. The illustrative example itself may be marked up using either the <eg> or the <egXML> element.

If an example contains XML markup, it should be marked up using the <egXML> element. In such a case, it will clearly be necessary to distinguish the markup within the example from the markup of the document itself. In an XML schema environment, this is easily done by nominating a different name space for the <egXML> element. For example:

Example 1.26

```
<p>The <gi>term</gi> element may be used
 to mark any technical term, thus :
 <egXML xmlns="http://www.tei-c.org/ns/Examples">
   This <term>recursion</term> is
   giving me a headache.</egXML></p>
```

Alternatively, the XML tagging within an example may be 'escaped', either by using entity references, or by wrapping the whole example in a CDATA marked section:

Example 1.27

```
<p>The <gi>term</gi> element may be used
 to mark any technical term, thus :
 <egXML>
   This &lt;term&gt;recursion&lt;/term&gt; is
   giving me a headache.</egXML></p>
```

or, equivalently:

Example 1.28

```
<p>The <gi>term</gi> element may be used
 to mark any technical term, thus :
 <egXML><![CDATA[
   This <term>recursion</term> is
   giving me a headache.]]><egXML>
```

If the XML contained in an example is not well-formed then it must either be enclosed in a CDATA marked section, or 'escaped' as above: this applies whether the <eg> or <egXML> is used. If it is well-formed but not valid, then it should be enclosed in a CDATA marked section within an <egXML>.

An <egXML> element should not be used to tag non-XML examples: the general purpose <eg> or <q> elements should be used for such purposes.

Classification of components In the TEI scheme elements are assigned to one or more classes, which may themselves have subclasses. The following elements are used to indicate class membership:

- **<classes>** specifies all the classes of which the documented element or class is a member or subclass.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<memberOf>** specifies class membership of the parent element or class. Selected attributes:

key (a name) specifies the identifier for a class of which the documented element or class is a member or subclass

The `<classes>` element appears within either the `<elementSpec>` or `<classSpec>` element. It specifies the classes of which the element or class concerned is a member by means of one or more `<memberOf>` child elements. Each such element references a class by means of its *key* attribute. Classes themselves are defined by the `<classSpec>` element described in section *1.1.3.4. Element Classes* below.

For example, to show that the element `<gi>` is a member of the class tei.oddPhr, the `<elementSpec>` which documents this element contains the following `<classes>` element:

Example 1.29

```
<classes>
   <memberOf key="tei.oddPhr"/>
 </classes>
```

Formal declarations The elements discussed in this section are formally declared as follows: [1.4] **remarks**:

```
remarks = element remarks { remarks.content, remarks.attributes }

remarks.content = macro.componentSeq
remarks.attributes =
   tei.global.attributes,
   [ a:defaultValue = "remarks" ] attribute TEIform { text }?,
   empty
```

listRef:

```
listRef = element listRef { listRef.content, listRef.attributes }

listRef.content = ( ptr | xptr )*

listRef.attributes =
   tei.global.attributes,
   [ a:defaultValue = "listRef" ] attribute TEIform { text }?,
   empty

tei.oddDecl |= listRef
```

exemplum:

```
exemplum = element exemplum { exemplum.content, exemplum.attributes }

exemplum.content = p*, ( egXML | eg ), p*

exemplum.attributes =
   tei.global.attributes,
   [ a:defaultValue = "exemplum" ] attribute TEIform { text }?,
   empty
```

classes:

```
classes = element classes { classes.content, classes.attributes }

classes.content = memberOf*

classes.attributes =
   tei.global.attributes,
   [ a:defaultValue = "classes" ] attribute TEIform { text }?,
   empty
```

memberOf:

```
memberOf = element memberOf { memberOf.content, memberOf.attributes }

memberOf.content = text

memberOf.attributes =
   tei.global.attributes,
   memberOf.attributes.key,
   [ a:defaultValue = "memberOf" ] attribute TEIform { text }?,
   empty

memberOf.attributes.key = attribute key { memberOf.attributes.key.content }?

memberOf.attributes.key.content = xsd:NCName
```

1.3.2 Element Specifications

The <elementSpec> element is used to document an element type, together with its associated attributes. In addition to the elements listed above, it may contain the following subcomponents:

- **<content>** contains the text of a declaration for the schema documented.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<attList>** contains documentation for all the attributes associated with this element, as a series of attDef elements. Selected attributes:

  org specifies whether all the attributes in the list are available (org="group") or only one of them (org="choice")

The content of the element <content> may be expressed in one of two ways. It may use a schema language of some kind, as defined by a pattern called schemapattern, which is declared in the tagdocs-decl module, defined by this chapter. Alternatively, the legal content for an element may be fully specified using the <valList> element, described in *1.1.3.3. Attribute list specification* below.

In the case of the TEI Guidelines, element content models are defined using Relax NG patterns, but the user may over-ride this by redefining this pattern.

Here is a very simple example

Example 1.34

```
<content>
 <text/>
</content>
```

This content model uses the Relax NG namespace, and will be copied unchanged to the output when Relax NG schemas are being generated. When an XML DTD is being generated, an equivalent declaration (in this case (#PCDATA)) will be output.

Here is a more complex example:

Example 1.35

```
<content>
 <group>
  <ref name="fileDesc"/>
  <zeroOrMore>
   <ref name="tei.header"/>
  </zeroOrMore>
  <optional>
   <ref name="revisionDesc"/>
  </optional>
 </group>
</content>
```

This is the content model for the <teiHeader> element, expressed in the Relax NG syntax, which again is copied unchanged to the output during schema generation. The equivalent DTD notation generated from this is (fileDesc, (%tei.header;)*, revisionDesc?).

The Relax NG language does not formally distinguish element names, class names or macro names: all names are patterns which are handled in the same way, as the above example shows. Within the TEI scheme, however, different naming conventions are used to distinguish amongst the objects being named. Unqualified names (`fileDesc`, `revisionDesc`) are always element names. Names prefixed with `tei.` (e.g. `tei.header`) are always class names. In DTD language, element classes are represented by parameter entities (`%tei.header;` in the above example) See further «a section of ST we haven't written yet».

1.3.3 Attribute list specification

The `<attList>` element is used to document information about a collection of attributes, either within a `<elementSpec>`, or within a `<classSpec>`. An attribute list can be organized either as a group of attribute definitions, all of which are understood to be available, or as a choice of attribute definitions, of which only one is understood to be available. An attribute list may also contain nested attribute lists.

The `<attDef>` element is used to document a single attribute, using an appropriate selection from the common elements already mentioned and the following which are specific to attributes:

- **<attDef>** contains the definition of a single attribute. Selected attributes:

  usage specifies the optionality of an attribute or element.

- **<datatype>** specifies the declared value for an attribute, by referring to any datatype defined by the chosen schema language.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<defaultVal>** specifies the default declared value for an attribute.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<valDesc>** specifies any semantic or syntactic constraint on the value that an attribute may take, additional to the information carried by the datatype element.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<valList>** contains one or more valItem elements defining possible values for an attribute.

  type specifies the extensibility of the list of attribute values specified. Legal values are:

  closed only the values specified are permitted.

  semi all the values specified should be supported, but other values are legal and software should have appropriate fallback processing for them.

  open the values specified are sample values only.
- **<valItem>** contains a single value and gloss pair for an attribute.
  - No attributes other than those globally available (see definition for tei.global.attributes)

The `<attList>` within a `<elementSpec>` is used to specify only the attributes which are specific to that particular element. Instances of the element may carry other attributes which are declared by the classes of which the element is a member. These extra attributes, which are shared by other elements, or by all elements, are specified by an `<attList>` contained within a `<classSpec>` element, as described in section *1.1.3.4. Element Classes* below.

Datatypes The `<datatype>` element is used to state what kind of value an attribute may have, using whatever facilities are provided by the underlying schema language. For the TEI scheme, expressed in relaxNG, elements from the Relax NG namespace may be used, for example

Example 1.36

```
<datatype>
 <text/>
</datatype>
```

permits any string of Unicode characters not containing markup, and is thus the equivalent of `CDATA` in DTD language.

The Relax NG language also provides support for a number of primitive datatypes which may be specified here, using the `<rng:data>` element: thus one may write

Example 1.37

```
<datatype>
 <data type="Boolean"/>
</datatype>
```

to specify that an element or attribute's contents should conform to the W3C definition for Boolean.

Although only one child element may be given, this might be a selector such as rng:choice to indicate multiple possibilities:

Example 1.38

```
<datatype>
 <choice>
  <data type="Date"/>
  <data type="Float"/>
 </choice>
</datatype>
```

which would permit either a date or a real number. In fact, the child element might be a `rng:list` element to indicate that a sequence of values is required, a `rng:param` element to specify a regular expression, or even a list of explicit `rng:values`. Such usages are permitted by the scheme documented here, but are not recommended when it is desired to remain independent of a particular schema language, since the full generality of one schema language cannot readily be converted to that of another. In the TEI abstract model, datatyping should preferably be carried out either by explicit enumeration of permitted values (using the TEI-specific `<valList>` element described below), or by definition of an explicit pattern, using the TEI-specific `<macroSpec>` element.

Value specification The `<valDesc>` element may be used to specify additional constraints on data content in an informal way: for example

Example 1.39

```
<valDesc>must point to another <gi>align</gi>
     element logically preceding this
     one.</valDesc>
<valDesc>Values should be Library of Congress subject
 headings.</valDesc>
<valDesc>A bookseller's surname,
  taken from the list in <bibl>Pollard and Redgrave</bibl>
</valDesc>
```

Alternatively, if a list of suggested or required values can be constructed, it should be supplied using the `<valList>` element, as in the following example:

Example 1.40

```
<valList type="closed">
 <valItem ident="req">
  <gloss>required</gloss>
 </valItem>
 <valItem ident="mwa">
  <gloss>mandatory when applicable</gloss>
 </valItem>
 <valItem ident="rec">
  <gloss>recommended</gloss>
 </valItem>
 <valItem ident="rwa">
  <gloss>recommended when applicable</gloss>
 </valItem>
 <valItem ident="opt">
  <gloss>optional</gloss>
 </valItem>
</valList>
```

Since this value list specifies that it is of type closed, only the values enumerated and glossed above are legal.

Examples The following `<attList>` demonstrates some of the possibilities; for more detailed examples, consult the tagged version of the reference material in these Guidelines.

Example 1.41

```
<attList>
 <attDef>
  <ident>type</ident>
  <datatype>
   <text/>
```

```
  </datatype>
  <defaultVal>simple</defaultVal>
  <valList type="semi">
   <valItem>
    <ident>ordered</ident>
    <gloss>list items are numbered or lettered. </gloss>
   </valItem>
   <valItem>
    <ident>bulleted</ident>
    <gloss>list items are marked with a bullet or other
                      typographic device. </gloss>
   </valItem>
   <valItem>
    <ident>simple</ident>
    <gloss>list items are not numbered or bulleted.</gloss>
   </valItem>
   <valItem>
    <ident>gloss</ident>
    <gloss>each list item glosses some term or
                concept, which is given by a label element preceding
               the list item.</gloss>
   </valItem>
  </valList>
  <desc>describes the form of the list.</desc>
  <remarks>
   <p>The formal syntax of the element declarations allows
            <gi>label</gi> tags to be omitted from lists tagged <tag>list
             type="gloss"</tag>; this is however a semantic error.</p>
  </remarks>
 </attDef>
</attList>
```

In the following example, the *org* attribute is used to indicate that instances of the element concerned may bear either a *bar* attribute or a <baz> attribute, but not both. The *bax* element is always available:

Example 1.42

```
<attList>
 <attDef ident="bax"><!-- ... -->
 </attDef>
 <attList org="choice">
  <attDef ident="bar"><!-- ... -->
  </attDef>
  <attDef ident="baz"><!-- ... -->
  </attDef>
 </attList>
</attList>
```

1.3.4 Element Classes

The element <classSpec> is used to document an element class, as defined in section «STEC». It has the following components, additional to those already mentioned:

- **<classSpec>** contains reference information for a TEI element class; that is a group of elements which appear together in content models, or which share some common attribute, or both. Selected attributes:

  type indicates whether this is a model class, an attribute class, or both.

- **<attList>** contains documentation for all the attributes associated with this element, as a series of attDef elements.

  org specifies whether all the attributes in the list are available (org="group") or only one of them (org="choice") Legal values are:

  group grouped

  choice alternated

The attribute *type* is used to distinguish between 'model' and 'attribute' classes. In the case of attribute classes, the attributes which define membership in the class are documented by an <attList> element contained within the <classSpec>. Members of the class documented by a <classSpec> point to it by supplying its identifier as the value of the *key* attribute on a <memberOf> element, given as a child of the <classes> element in the <elementSpec> that defines the element concerned. For example, the <elementSpec> for the element <hi> contains the following:

Example 1.43

```
<classes>
 <memberOf key="tei.hqphrase"/>
</classes>
```

which indicates that the <hi> element is a member of the class with identifier tei.phrase. The <classSpec> documenting this class is as follows:

Example 1.44

```
<classSpec type="model">
 <ident>tei.hqphrase</ident>
 <desc>groups phrase-level elements related to highlighting. </desc>
 <classes>
  <memberOf key="tei.phrase"/>
 </classes>
</classSpec>
```

which indicates that the class tei.hqphrase is actually a member (or subclass) of the class tei.phrase.

When a <classSpec> contains an <attList> element, all the members of that class inherit the attributes specified by it. For example, the class tei.interpret defines a small set of attributes common to all elements which are members of that class: those attributes are listed by the <attList> element contained by the <classSpec> for tei.interpret. When processing the documentation elements for elements which are members of that class, an ODD processor is required to extend the <attList> (or equivalent) for such elements to include any attributes defined by the <classSpec> elements concerned. There is a single global attribute class, tei.global.

1.3.5 Formal declarations

The elements discussed in this section are formally defined as follows: [1.5] **content**:

```
content = element content { content.content, content.attributes }

content.content = schemapattern | valList
content.attributes =
   tei.global.attributes,
   [ a:defaultValue = "content" ] attribute TEIform { text }?,
   empty
```

schemapattern:

```
schemapattern = element * { ( attribute * { text } | text | schemapattern )* }
```

attList:

```
attList = element attList { attList.content, attList.attributes }

attList.content = ( attDef | attList )*

attList.attributes =
   tei.global.attributes,
   attList.attributes.org,
   [ a:defaultValue = "attList" ] attribute TEIform { text }?,
   empty

attList.attributes.org =
   [ a:defaultValue = "group" ]
   attribute org { attList.attributes.org.content }?

attList.attributes.org.content = "group" | "choice"
```

attDef:

```
attDef = element attDef { attDef.content, attDef.attributes }

attDef.content =
   macro.glossSeq,
   datatype?,
   defaultVal?,
   ( valList | valDesc )?,
   exemplum?,
   remarks?

attDef.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   attDef.attributes.usage,
   [ a:defaultValue = "attDef" ] attribute TEIform { text }?,
   empty

attDef.attributes.usage =
   [ a:defaultValue = "opt" ]
   attribute usage { attDef.attributes.usage.content }?

attDef.attributes.usage.content = "req" | "mwa" | "rec" | "rwa" | "opt"

tei.identifiable |= attDef
```

datatype:

```
datatype = element datatype { datatype.content, datatype.attributes }

datatype.content = schemapattern
datatype.attributes =
   tei.global.attributes,
   [ a:defaultValue = "datatype" ] attribute TEIform { text }?,
   empty
```

defaultVal:

```
defaultVal = element defaultVal { defaultVal.content, defaultVal.attributes }

defaultVal.content = text

defaultVal.attributes =
   tei.global.attributes,
   [ a:defaultValue = "defaultVal" ] attribute TEIform { text }?,
   empty
```

valDesc:

```
valDesc = element valDesc { valDesc.content, valDesc.attributes }

valDesc.content = macro.phraseSeq
valDesc.attributes =
   tei.global.attributes,
   [ a:defaultValue = "valDesc" ] attribute TEIform { text }?,
   empty
```

valItem:

```
valItem = element valItem { valItem.content, valItem.attributes }
```

```
valItem.content = macro.glossSeq
valItem.attributes =
   tei.global.attributes,
   tei.identifiable.attributes,
   [ a:defaultValue = "valItem" ] attribute TEIform { text }?,
   empty

tei.identifiable |= valItem
```

valList:

```
valList = element valList { valList.content, valList.attributes }

valList.content = valItem+

valList.attributes =
   tei.global.attributes,
   valList.attributes.type,
   [ a:defaultValue = "valList" ] attribute TEIform { text }?,
   empty

valList.attributes.type =
   [ a:defaultValue = "open" ]
   attribute type { valList.attributes.type.content }?

valList.attributes.type.content = "closed" | "semi" | "open"
```

1.3.6 Pattern Documentation

The <macroSpec> element is used to document any other entity not otherwise documented by the elements described in this chapter. Its chief uses are to provide systematic documentation of the parameter entities used within TEI DTD fragments and to describe common content models, but it may be used for any purpose. It has the following components additional to those already introduced:

- **<macroSpec>** documents the function and implementation of a pattern. Selected attributes:

  type indicates which type of entity should be generated, when an ODD processor is generating a module using SGML syntax.

- **<remarks>** contains any commentary or discussion about the usage of an element, attribute, class, or entity not otherwise documented within the containing element.
  - No attributes other than those globally available (see definition for tei.global.attributes)
- **<stringVal>** contains the intended expansion for the entity documented by an patternSpec element, enclosed by quotation marks.
  - No attributes other than those globally available (see definition for tei.global.attributes)

1.4 Building a schema

The specification elements, and several of their children, are all members of the identifiable class, from which they inherit the following attributes:

- **<tei.identifiable>** elements which can be referenced by means of a key attribute

  ident Supplies the identifier by which this element is referenced.

  depend The name of a module on which this object depends.

  predeclare Says whether the class should be treated as global, and so needs predefining in the core.

  module Supplies the name of the module in which this object is to be defined.

  mode specifies the effect of this declaration on its parent module. Values are:

  add this declaration is added to the current definitions

  delete this declaration and all of its children are removed from the current setup

  change this declaration changes the declaration of the same name in the current definition

replace this declaration replaces the declaration of the same name in the current definition

These attributes are used by an ODD processor to determine how declarations are to be combined to form a schema or DTD, as further discussed in this section.

As noted above, a TEI schema is defined by a <schema> element containing an arbitrary mixture of explicit declarations for objects (i.e. elements, classes, patterns, or macro specifications) and references to other objects containing such declarations (i.e. references to specification groups, or to modules). A major purpose of this mechanism is to simplify the process of defining user customizations, by providing a formal method for the user to combine new declarations with existing ones, or to modify particular parts of existing declarations.

In the simplest case, a user-defined schema might simply combine all the declarations from two nominated modules:

Example 1.53

```
<schema>
 <moduleRef name="teistructure"/>
 <moduleref name="linking"/>
</schema>
```

An ODD processor, given such a document, would combine the declarations which belong to the named modules, and deliver the result as a schema of the requested type. It might also generate documentation for all and only the elements declared by those modules.

A schema might also include declarations for new elements, as in the following example:

Example 1.54

```
<schema>
 <moduleRef name="teiheader"/>
 <moduleref name="verse"/>
 <elementSpec ident="soundClip">
  <classes memberOf="tei.data"/>
 </elementSpec>
</schema>
```

A declaration for the element <soundClip>, which is not defined in the TEI scheme, will be added to the output schema. This element will also be added to the existing TEI class tei.data, and will thus be avilable in TEI conformant documents.

A schema might also include re-declarations of existing elements, as in the following example:

Example 1.55

```
<schema>
 <moduleRef name="teiheader"/>
 <moduleref name="teistructure"/>
 <elementSpec ident="head" mode="change">
  <content>
   <text/>
  </content>
 </elementSpec>
</schema>
```

The effect of this is to redefine the content model for the element <head> as plain text, by over-riding the <content> child of the selected <elementSpec>. The attribute specification mode="change" has the effect of over-riding only those children elements of the <elementSpec> which appear both in the original specification and in the new specification supplied above: <content> in this example. Note that if the value for mode were replace, the effect would be to replace all children elements of the original specification with the the children elements of the new specification, and thus (in this example) to delete all of them except <content>.

A schema may not contain more than two declarations for any given component. The value of the mode attribute is used to determine exactly how the second declaration (and its constituents) should be combined with the first. The following table summarizes how a processor should resolve duplicate declarations:

mode value	existing declaration	effect
add	no	add new declaration to schema; process its children in add mode
add	yes	raise error
replace	no	raise error

replace	yes	retain existing declaration; process new children in replace mode; ignore existing children
change	no	raise error
change	yes	replace existing declaration; process new children in replace mode; retain existing (unreplaced) children
delete	no	remove existing declaration; ignore this declaration and its children
delete	yes	ignore existing declaration and its children

1.5 Formal declaration

This chapter documents the following two modules

Module Entities for Tag Set Documentation:

Module Elements for Tag Set Documentation:

The elements described in this chapter are all members of one of four classes: oddDecl, oddRef, oddPhr, or oddPara. These classes are declared along with the other general TEI classes, in the basic structure module documented in «ST».

In addition, some elements are members of the tei.identifiable class, which is documented in *1.1.4. Building a schema* above, and make use of the schemapattern pattern, which is documented in *1.1.3.2. Element Specifications* above.

The tagdocs module consists of the following specification groups:

[1.6]

« *include 1.1* » « *include 1.2* » « *include 1.3* » « *include 1.4* » « *include 1.5* »

2 Class catalogue

2.1 tei.identifiable [class]

Description: elements which can be referenced by means of a *key* attribute

Attributes: (In addition to global attributes)

ident Supplies the identifier by which this element is referenced.

depend The name of a module on which this object depends.

predeclare Says whether the class should be treated as global, and so needs predefining in the core.

module Supplies the name of the module in which this object is to be defined.

mode specifies the effect of this declaration on its parent module. Values are:

- **add** this declaration is added to the current definitions
- **delete** this declaration and all of its children are removed from the current setup
- **change** this declaration changes the declaration of the same name in the current definition
- **replace** this declaration replaces the declaration of the same name in the current definition

ADD means Create me if I dont exist[Process my new children in ADD mode] Raise an error if I do exist and am identifiable

REPLACE Leave me alone Process my new children in REPLACE mode Dont process my old children

DELETE Don't process me or my children (any new children supplied are erroneous)

CHANGE Leave me alone and process my old and new children in CHANGE mode

Member of classes (none)

Members: *moduleSpec*: *schemaSpec*: *elementSpec*: *classSpec*: *macroSpec*: *attDef*: *valItem*

Module: tagdocs-decl

3 Pattern catalogue

4 Element catalogue

4.1 att [element]

Description: (attribute) contains the name of an attribute appearing within running text.

Classes: tei.oddPhr

Declaration:

```
element att { tei.global.attributes, att.attributes.scheme, text }
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

scheme supplies the name of the scheme in which this name is defined. Sample values include:

TEI this element is part of the TEI scheme.

DBK this element is part of the Docbook scheme.

Example:

```
<p>The TEI defines three <soCalled>global</soCalled> attributes;
     their names are <att>id</att>, <att>rend</att> and <att>n</att>.
     <att tei="no">type</att> is not among them.</p>
```

A namespace prefix may be used in order to specify the scheme.

Module: tagdocs

4.2 attDef [element]

Description: (attribute definition) contains the definition of a single attribute.

Classes: *tei.identifiable*

Declaration:

```
element attDef
{
   tei.global.attributes,
   tei.identifiable.attributes,
   attDef.attributes.usage,
   (
      macro.glossSeq,
      datatype?,
      defaultVal?,
      ( valList | valDesc )?,
      exemplum?,
      remarks?
   )
}
```

Attributes: (In addition to global attributes and those inherited from : *tei.identifiable*)

usage specifies the optionality of an attribute or element. Legal values are:

req required

mwa mandatory when applicable

rec recommended

rwa recommended when applicable

opt optional

Example:

```
<attDef usage="rec" ident="type">
 <desc>specifies a name conventionally used for this
          level of subdivision, e.g. <q>act</q>, <q>volume</q>,
          <q>book</q>, <q>section</q>, <q>canto</q>, etc.</desc>
</attDef>
```

Module: tagdocs

4.3 attList [element]

Description: contains documentation for all the attributes associated with this element, as a series of attDef elements.

Declaration:

```
element attList
{
   tei.global.attributes,
   attList.attributes.org,
   ( attDef | attList )*
}
```

Attributes: (In addition to global attributes)

org specifies whether all the attributes in the list are available (org="group") or only one of them (org="choice") Legal values are:

group grouped

choice alternated

Example:

```
<attList>
 <attDef ident="type" usage="opt">
  <equiv/>
  <datatype>
   <text/>
  </datatype>
  <desc>type of schema</desc>
 </attDef>
</attList>
```

contains a series of `<attDef>` elements only

Module: tagdocs

4.4 classSpec [element]

Description: contains reference information for a TEI element class; that is a group of elements which appear together in content models, or which share some common attribute, or both.

Classes: tei.oddDecl: *tei.identifiable*

Declaration:

```
element classSpec
{
   tei.global.attributes,
   tei.identifiable.attributes,
   classSpec.attributes.type,
   ( macro.glossSeq, classes?, attList?, exemplum*, remarks?, listRef* )
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddDecl: *tei.identifiable*)

type indicates whether this is a model class, an attribute class, or both. Legal values are:

model members of this class appear in the same content models

atts members of this class share common attributes

both members of this class share attributes and also appear in the same content models

Example:

```
<classSpec type="model" id="SEG">
 <class>seg</class>
 <desc>elements for arbitrary segmentation.</desc>
 <ptr target="COSE"/>
</classSpec>
```

Module: tagdocs

4.5 classes [element]

Description: specifies all the classes of which the documented element or class is a member or subclass.

Declaration:

```
element classes { tei.global.attributes, ( memberOf* ) }
```

Attributes: Global attributes only

Example:

```
<classes>
 <memberOf key="tei.hqinter"/>
 <memberOf key="tei.declable"/>
</classes>
```

This `<classes>` element indicates that the element documented (which may be an element or a class) is a member of two distinct classes: tei.hqinter and tei.declable.

An empty `<classes>` element indicates that the element documented is not a member of any class. This should not generally happen.

Module: tagdocs

4.6 code [element]

Description: contains literal code

Classes: tei.oddPhr

Declaration:

```
element code { tei.global.attributes, code.attributes.type, text }
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

type the language of the code

Module: tagdocs

4.7 content [element]

Description: (schema declaration) contains the text of a declaration for the schema documented.

Attributes: Global attributes only

Declaration:

```
element content { tei.global.attributes, ( schemapattern | valList ) }
```

Example:

This content model allows either a sequence of paragraphs or a series of msItem elements optionally preceded by a summary:

```
<content>
 <choice>
  <oneOrMore>
   <ref name="tei.paragraph"/>
  </oneOrMore>
  <group>
   <optional>
    <ref name="summary"/>
   </optional>
   <oneOrMore>
    <ref name="msItem"/>
   </oneOrMore>
  </group>
 </choice>
</content>
```

As the example shows, content models in P5 may be expressed using the RelaxNG syntax directly. More exactly, they are defined using the pattern schemapattern. Alternatively, a content model may be expressed using the TEI <valList> element.

Module: tagdocs

4.8 datatype [element]

Description: specifies the declared value for an attribute, by referring to any datatype defined by the chosen schema language.

Declaration:

```
element datatype { tei.global.attributes, schemapattern }
```

Attributes: Global attributes only

Example:

```
<datatype><rng:text/></datatype>
```

Module: tagdocs

4.9 defaultVal [element]

Description: specifies the default declared value for an attribute.

Attributes: Global attributes only

Declaration:

```
element defaultVal { tei.global.attributes, text }
```

Example:

```
 <defaultVal>#IMPLIED</defaultVal>
```

any legal declared value or TEI-defined keyword

Module: tagdocs

4.10 eg [element]

Description: contains a single example demonstrating the use of an element or attribute.

Attributes: Global attributes only

Classes: tei.common: tei.data

Declaration:

```
element eg { tei.global.attributes, text }
```

Example:

```
<p>The <gi>term</gi> element may be used to mark any
 technical term:</p>
 <egXML xmlns="http://www.tei-c.org/ns/Examples">This <term>recursion</term> is
 giving me a headache.</egXML>
```

If the example contains SGML or XML markup, either it should be enclosed within a CDATA marked section, or character entity references must be used to represent the markup delimiters.

Module: tagdocs

4.11 egXML [element]

Description: contains a single example demonstrating the use of an XML element or attribute.

Classes: tei.common: tei.data

Declaration:

```
element egXML { tei.global.attributes, text }
```

Attributes: Global attributes and those inherited from : tei.common: tei.data

The element's contents should be marked as belonging to the namespace http://www.tei-c.org/ns/Examples if they are to be validated; if the contents are not well-formed XML, they must be enclosed in a CDATA marked section

Module: tagdocs

4.12 elementSpec [element]

Description: documents the structure, content, and purpose of a single element type.

Classes: tei.oddDecl: *tei.identifiable*

Declaration:

```
element elementSpec
{
   tei.global.attributes,
   tei.identifiable.attributes,
   elementSpec.attributes.ns,
   elementSpec.attributes.usage,
   (
      macro.glossSeq,
      classes?,
      content?,
      attList?,
      exemplum*,
      remarks?,
      listRef*
   )
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddDecl: *tei.identifiable*)

ns (namespace) specifies the namespace to which this element belongs

usage specifies the optionality of an attribute or element. Legal values are:

req required

mwa mandatory when applicable

rec recommended

rwa recommended when applicable

opt optional

Example:

```
<elementSpec module="tagdocs" id="CODE" usage="mwa" ident="code">
 <equiv/>
 <gloss/>
 <classes>
  <memberOf key="tei.oddPhr"/>
 </classes>
 <content>
  <text/>
 </content>
 <attList>
  <attDef ident="type" usage="opt">
   <equiv/>
   <datatype>
    <text/>
   </datatype>
   <desc>the language of the code</desc>
  </attDef>
```

```
 </attList>
 <desc>contains literal code</desc>
</elementSpec>
```

Module: tagdocs

4.13 exemplum [element]

Description: contains a single example demonstrating the use of an element, together with optional paragraphs of commentary.

Attributes: Global attributes only

Declaration:

```
element exemplum { tei.global.attributes, ( p*, ( egXML | eg ), p* ) }
```

Example:

```
    <exemplum>
   <p>The <gi>name</gi> element can be used for both
   personal names and place names:</p>
   <egXML xmlns="http://www.tei-c.org/ns/Examples"><![&nil;CDATA[
     <q>My dear <name type="person">Mr. Bennet</name>,</q>
     said his lady to him one day, <q>have you heard that
     <name type="place">Netherfield Park</name> is let
     at last?</q>]&nil;]></egXML>
   <p>As shown above, the <att>type</att> attribute may be used
   to distinguish the one from the other.</p>
  </exemplum>
```

Note that an explicit end-tag must be supplied for the paragraph immediately preceding the <eg> element within an <exemplum>, to prevent the <eg> from being mistaken for part of the paragraph.

Module: tagdocs

4.14 gi [element]

Description: (generic identifier) contains the name (generic identifier) of an element.

Classes: tei.oddPhr

Declaration:

```
element gi { tei.global.attributes, gi.attributes.scheme, text }
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

scheme supplies the name of the scheme in which this name is defined. Sample values include:

TEI this element is part of the TEI scheme.

DBK this element is part of the Docbook scheme.

Example:

```
<p>The
     <gi>xhtml:li</gi> element is roughly analagous to the
     <gi>item</gi> element, as is the <gi scheme="DBK">listItem</gi> element.</p>
```

This example shows the use of both a namespace prefix and the schema attribute as alternative ways of indicating that the gi in question is not a TEI element name: in practice only one method should be adopted.

Module: tagdocs

4.15 ident [element]

Description: identifies or names any kind of object

Classes: tei.oddPhr

Declaration:

```
element ident { tei.global.attributes, ident.attributes.type, text }
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

type supplies a categorization for this identifier, using any convenient typology.

Example:

```
<ident>profileDesc</ident>
```

In running prose, this element may be used for any kind of identifier in any formal language.

Module: tagdocs

4.16 listRef [element]

Description: (list of references) supplies a list of significant references to places where this element is discussed, in the current document or elsewhere.

Classes: tei.oddDecl

Declaration:

```
element listRef { tei.global.attributes, ( ptr | xptr )* }
```

Attributes: Global attributes and those inherited from : tei.oddDecl

Example:

```
<listRef><ptr target="Dc12"/>
```

Module: tagdocs

4.17 macroSpec [element]

Description: documents the function and implementation of a pattern.

Classes: tei.oddDecl: *tei.identifiable*

Declaration:

```
element macroSpec
{
   tei.global.attributes,
   tei.identifiable.attributes,
   macroSpec.attributes.type,
   ( macro.glossSeq, ( stringVal | content )+, exemplum*, remarks?, listRef* )
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddDecl: *tei.identifiable*)

type indicates which type of entity should be generated, when an ODD processor is generating a module using SGML syntax. Legal values are:

pe parameter entity

epe element parameter entity

ge general entity

dt datatype entity

When generating SGML, an ODD processor should use the `<ident>` and `<string>` children of this element to generate an entity declaration of the type indicated by the *type* attribute. When generating schema, the `<ident>` and `<schemapattern>` children of this element should be used to generate a pattern declaration. ?????

Module: tagdocs

4.18 memberOf [element]

Description: specifies class membership of the parent element or class.

Declaration:

```
element memberOf { tei.global.attributes, memberOf.attributes.key, text }
```

Attributes: (In addition to global attributes)

key (a name) specifies the identifier for a class of which the documented element or class is a member or subclass

Elements or classes which are members of multiple (unrelated) classes will have more than one `<memberOf>` element, grouped by a `<classes>` element. If an element is a member of a class C1, which is itself a subclass of a class C2, there is no need to state this, other than in the documentation for class C1.

Any additional comment or explanation of the class membership may be provided as content for this element.

Module: tagdocs

4.19 moduleRef [element]

Description: references a module which is to be incorporated into a schema.

Classes: tei.oddRef

Declaration:

```
element moduleRef
{
   tei.global.attributes,
   ( moduleRef.attributes.key | moduleRef.attributes.url )?,
   empty
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddRef)

Choice: key the name of a TEI module

url refers to a non-TEI module by external location

Example:

```
<moduleRef key="linking"/>
```

This embeds the linking module.

Modules are identified by the name supplied as value for the ident attribute on the `<module>` element in which they are declared. A URI may also be supplied in the case of a non-TEI module.

The effect of this element is to make all the declarations contained by the referenced module available to the schema being compiled.

Module: tagdocs

4.20 moduleSpec [element]

Description: documents the structure, content, and purpose of a single module, i.e. a named and externally visible group of declarations.

Classes: tei.oddDecl: *tei.identifiable*

Declaration:

```
element moduleSpec
{
   tei.global.attributes,
   tei.identifiable.attributes,
   moduleSpec.attributes.type,
   ( macro.glossSeq, exemplum*, remarks?, listRef* )
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddDecl: *tei.identifiable*)

type type of module to be generated

Module: tagdocs

4.21 remarks [element]

Description: contains any commentary or discussion about the usage of an element, attribute, class, or entity not otherwise documented within the containing element.

Attributes: Global attributes only

Declaration:

```
element remarks { tei.global.attributes, macro.componentSeq }
```

Example:

```
<remarks>
 <p>This element is probably redundant.</p>
</remarks>
```

Contains at least one paragraph, unless it is empty.

As defined in ODD, must contain paragraphs; should be special.para

Module: tagdocs

4.22 schemaSpec [element]

Description: generates a TEI-conformant schema and documentation for it.

Classes: tei.oddPhr: *tei.identifiable*

Declaration:

```
element schemaSpec
{
   tei.global.attributes,
   tei.identifiable.attributes,
   schemaSpec.attributes.type,
   schemaSpec.attributes.start,
   schemaSpec.attributes.namespace,
   ( macro.glossSeq, ( moduleRef | specGrpRef | tei.oddDecl )* )
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr: *tei.identifiable*)

type type of schema

start specifies entry points to the schema, i.e. which elements are allowed to be used as the root of documents conforming to it.

namespace specifies the default namespace (if any) applicable to components of the schema.

A schema combines references to modules or specification groups with other atomic declarations. The processing of a schema element must resolve any conflicts amongst the declarations it contains or references. Different ODD processors may generate schemas and documentation using different concrete syntaxes.

Module: tagdocs

4.23 specDesc [element]

Description: (Element or class description) indicates that a description of the specified element should be included at this point within a `<tagList>`.

Classes: tei.oddPhr

Declaration:

```
element specDesc
{
   tei.global.attributes,
   specDesc.attributes.key,
   specDesc.attributes.atts,
   empty
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

key (identifier) supplies the identifier of the documentary element or class for which a description is to be obtained.

atts (attributes) supplies attribute names for which descriptions should be obtained.

Example:

```
<specDesc key='orth' />
```

It is not required that all attributes defined for an element be included in the taglist at this point. Those that are appear in an embedded glossary list; any list of values defined for an element is itself further embedded as a list; no selection among the values is possible. The list of attributes may include some which are inherited by virtue of an element's class membership; descriptions for such attributes may also be retrieved using the another `<specDesc>`, this time pointing at the relevant class.

Module: tagdocs

4.24 specGrp [element]

Description: (specification group) contains any convenient grouping of specifications for use within the current module.

Classes: tei.oddDecl

Declaration:

```
element specGrp
{
   tei.global.attributes,
   ( tei.oddDecl | tei.oddRef | tei.chunk )*
}
```

Attributes: Global attributes and those inherited from : tei.oddDecl

Example:

```
<specGrp id="DAILC">
 <elementSpec ident="s">
 <!-- ... -->
 </elementSpec>
 <elementSpec ident="cl">
 <!-- ... -->
 </elementSpec>
 <elementSpec ident="w">
 <!-- ... -->
 </elementSpec>
 <elementSpec ident="m">
 <!-- ... -->
 </elementSpec>
 <elementSpec ident="c">
 <!-- ... -->
 </elementSpec>
 </specGrp>
```

A specification group is referenced by means of its *id* attribute. The declarations it contains may be included in a `<module>` element only by reference (using a `<specGrpRef>` element): it may not be nested within a `<module>` element.

Different ODD processors may generate representations of the specifications contained by a `<specGrp>` in different concrete syntaxes. For P5 the intention is to generate modules using both XML and Relax NG, and to use only the compressed Relax NG syntax to represent them.

Module: tagdocs

4.25 specGrpRef [element]

Description: (reference to a specification group) indicates that the declarations contained by the <specGrp> referenced should be inserted at this point.

Classes: tei.oddRef

Declaration:

```
element specGrpRef
{
   tei.global.attributes,
   specGrpRef.attributes.target,
   empty
}
```

Attributes: (In addition to global attributes and those inherited from : tei.oddRef)

target points at the specification group which logically belongs here.

Example:

```
<p>This part of
     the module  contains declarations for names of persons, places,
     and organisations:
     <specGrp>
  <specGrpRef target="names.pers"/>
  <specGrpRef target="names.place"/>
  <specGrpRef target="names.org"/>
 </specGrp>
</p><!-- elsewhere --><specGrp id="names.pers"><!--... -->
</specGrp><!-- elsewhere --><specGrp id="names.place"><!--... -->
</specGrp><!-- elsewhere --><specGrp id="names.org"><!--... -->
</specGrp>
```

In ODD documentation processing, a <specGrpRef> usually produces a comment indicating that a set of declarations printed in another section will be inserted at this point in the <specGrp> being discussed. In schema processing, the contents of the specified <specGrp> are made available for inclusion in the generated schema.

The specification group identified by the *target* attribute must be part of the current ODD document.

Module: tagdocs

4.26 specList [element]

Description: marks where a list of descriptions is to be inserted into the prose documentation.

Classes: tei.oddPhr

Declaration:

```
element specList { tei.global.attributes, specDesc+ }
```

Attributes: Global attributes and those inherited from : tei.oddPhr

Example:

```
<specList>
 <specDesc key='lb' atts='ed'/>
 <specDesc key='div'/>
 </speclist>
```

Module: tagdocs

4.27 stringVal [element]

Description: contains the intended expansion for the entity documented by an patternSpec element, enclosed by quotation marks.

Attributes: Global attributes only

Declaration:

```
element stringVal { tei.global.attributes, text }
```

Example:

```
<stringVal>"the choice of quotes isn't always unimportant"</stringVal>
```

Example:

System entities should include the SYSTEM keyword within the content of this element, as shown:

```
<stringVal>SYSTEM 'teiclasses"'</stringVal>
```

The content of this element is the replacement text for the named entity, including any keywords, and surrounded by appropriate quotation marks.

Module: tagdocs

4.28 tag [element]

Description: contains text of a complete start- or end-tag, possibly including attribute specifications, but excluding the opening and closing markup delimiter characters.

Classes: tei.oddPhr

Declaration:

```
element tag { tei.global.attributes, tag.attributes.scheme, text }
```

Attributes: (In addition to global attributes and those inherited from : tei.oddPhr)

scheme supplies the name of the scheme in which this name is defined. Legal values are:

TEI this element is part of the TEI scheme.

DBK this element is part of the Docbook scheme.

Example:

```
Mark the start of each italicised phrase with a <tag>hi rend="it"</tag>
     tag, and its end with a <tag>/hi</tag> tag.
```

Module: tagdocs

4.29 val [element]

Description: (value) contains a single attribute value.

Attributes: Global attributes only

Classes: tei.oddPhr

Declaration:

```
element val { tei.global.attributes, text }
```

Example:

```
        <val>unknown</val>
```

Module: tagdocs

4.30 valDesc [element]

Description: (value description) specifies any semantic or syntactic constraint on the value that an attribute may take, additional to the information carried by the datatype element.

Attributes: Global attributes only

Declaration:

```
element valDesc { tei.global.attributes, macro.phraseSeq }
```

Example:

```
<valDesc>must point to another <gi>align</gi>
     element logically preceding this
     one.</valDesc>
```

Module: tagdocs

4.31 valItem [element]

Description: (value definition) contains a single value and gloss pair for an attribute.

Attributes: Global attributes only

Classes: *tei.identifiable*

Declaration:

```
element valItem
{
   tei.global.attributes,
   tei.identifiable.attributes,
   ( macro.glossSeq )
}
```

Example:

```
<valItem ident="dub">
 <altIdent xml:lang="FRA">dou</altIdent>
 <equiv name="unknown"/>
 <gloss>dubious</gloss>
</valItem>
```

Module: tagdocs

4.32 valList [element]

Description: (value list) contains one or more `<valItem>` elements defining possible values for an attribute.

Declaration:

```
element valList { tei.global.attributes, valList.attributes.type, valItem+ }
```

Attributes: (In addition to global attributes)

type specifies the extensibility of the list of attribute values specified. Legal values are:

- **closed** only the values specified are permitted.
- **semi** all the values specified should be supported, but other values are legal and software should have appropriate fallback processing for them.
- **open** the values specified are sample values only.

Example:

```
<valList type="closed">
 <valItem ident="req">
  <gloss>required</gloss>
 </valItem>
 <valItem ident="mwa">
  <gloss>mandatory when applicable</gloss>
 </valItem>
 <valItem ident="rec">
  <gloss>recommended</gloss>
```

```
  </valItem>
  <valItem ident="rwa">
   <gloss>recommended when applicable</gloss>
  </valItem>
  <valItem ident="opt">
   <gloss>optional</gloss>
  </valItem>
 </valList>
```

Module: tagdocs