TEI By Example

Edward Vanhoutte
Ron Van den Branden
Melissa Terras

Centre for Scholarly Editing and Document Studies (CTB), Royal Academy of Dutch Language and Literature, Belgium, Gent, 9 July 2010

Last modified on: 2018-04-28
Licensed under a Creative Commons Attribution ShareAlike 3.0 License
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Selecting and restricting the TEI model</td>
<td>1</td>
</tr>
<tr>
<td>3.1 Starting from a minimal schema</td>
<td>1</td>
</tr>
<tr>
<td>3.1.1 Generating a schema</td>
<td>4</td>
</tr>
<tr>
<td>3.1.2 Generating documentation for a schema</td>
<td>6</td>
</tr>
<tr>
<td>3.1.3 Generating an ODD file</td>
<td>7</td>
</tr>
<tr>
<td>3.2 What does a minimal TEI customisation tell us?</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Selecting modules and elements</td>
<td>15</td>
</tr>
<tr>
<td>3.4 Changing attributes</td>
<td>21</td>
</tr>
<tr>
<td>3.4.1 Changing individual attributes</td>
<td>21</td>
</tr>
<tr>
<td>3.4.2 Changing attribute classes</td>
<td>25</td>
</tr>
</tbody>
</table>
3. Selecting and restricting the TEI model

3.1 Starting from a minimal schema

If you point your browser to https://roma.tei-c.org/, following screen should appear:

![Annotation interface](image)

**Fig. 4.**
This is the start screen for new customisations, offering you a choice between four options:

- **Build schema**: start creating a customisation from the absolute minimum TEI requirements
- **Reduce schema**: start creating a customisation by reducing the maximal possible TEI model
- **From template**: start creating a customisation from one of the TEI sample customisations
- **From existing customisation**: start creating a customisation from an existing customisation that can be uploaded

For the purpose of this tutorial, we’ll set out from a minimal customisation. Select the first option and press the ‘Start’ button.

This will produce the main Roma dashboard:
Fig. 5.

You'll see no less than 10 different tabs at the top of the screen. They are:

- **New** takes you back to the start screen, where you can start creating a new TEI customisation
- **Customize** the current tab, where you can provide metadata for your customisation
- **Language** allows you to choose between translations in different languages for the schema and its documentation
- **Modules** lets you pick the parts of TEI you need
Add elements allows you to add your own elements
Change classes allows you to change and add attributes
Schema lets you choose what kind of schema you want to generate
Documentation allows you to choose what kind of output format you want for your schema documentation
Save allows you to save your customisation as an ODD file
Sanity Checker allows you to formally check the decisions you made for your customisation

For now, let’s just personalise the metadata: fill in ‘A TBE customisation’ in the ‘title’ field; ‘TBEcustom’ in the ‘Filename’ field; and ‘The TBE Crew’ in the ‘Author name’ field. Afterwards, press ‘Save’. This will produce the same screen, only now your values are saved (you can check for yourself how the message in the top right corner now states that ‘You are currently working on A TBE customisation’).

That’s it! We have created a first TEI customisation already. Before we proceed, let’s see how we can use Roma to derive documentation, schemas and an ODD file for this (minimal) TEI customisation. Since these are frequent operations in customising TEI, they are treated in separate subdivisions below.

3.1.1 Generating a schema

Select the ‘Schema’ tab, choose the schema language of your choice (Relax NG compact, Relax NG XML, W3C Schema, or DTD), and press the ‘Generate’ button.

......

1 It’s important to remember saving your changes in Roma at all times! This is usually done by pressing the ‘Save’ (or like-named) button at the bottom of the different tab screens.
Make sure you save the file, and see how this produces a file named ‘TBEcustom’, as we specified in the ‘Customize’ tab. The file’s extension depends on the schema format chosen: .rnc (Relax NG compact), .rng (Relax NG XML), .xsd (W3C schema), or .dtd (DTD). You can use this file to validate your TEI documents against.
3.1.2 Generating documentation for a schema

Select the ‘Documentation’ tab, choose the output format of your choice (html, PDF, TEI Lite, or TEI) and press the ‘Generate’ button.

![Image of TEI site](image-url)

**Fig. 7.**
Make sure you save the file, and see how this produces a file named ‘TBEcustom_doc’, either in HTML, PDF, or TEI XML format. This documentation will serve as your personal TEI Guidelines, containing formal references for all elements in the schema, as well as any prose documentation present in the ODD file.

### 3.1.3 Generating an ODD file

Without doubt, saving your customisation as an ODD file is the most important step of customising TEI. It will allow you (or others) to upload this customisation again for reuse, further fine tuning, and / or generating both schemas and documentation from this single source file again. In order to save your customisation as an ODD file, all you have to do is selecting the ‘Save Customization’ tab in Roma. This will immediately download the ODD file.
This will create a file called ‘TBEcustom.xml’. Note how, again, the file name corresponds to the one specified in the ‘Customize’ tab.

2 Always make sure to first save all changes you’ve made in Roma before saving a customisation as an ODD file!

Fig. 8.
Roma provides a visual interface to create TEI customisations, either from scratch, from a maximal TEI schema, from a TEI template, or from a previously saved ODD file. Customisations can be edited in Roma and exported as an ODD (One Document Does it all) file, from which both the actual TEI schema and accompanying documentation can be derived, in a number of output formats. The ODD file is the heart of your TEI customisation.

3.2 What does a minimal TEI customisation tell us?

Before proceeding, there are some interesting insights to be gained from an analysis of our first, minimal, TEI customisation. Currently, the ‘TBEcustom.xml’ ODD file looks like this:
One immediate observation is that an ODD file is just a regular TEI document, with a `<TEI>` document element, containing a `<teiHeader>` element and a `<text>` element. Remember the metadata you entered in the ‘Customize’ tab, and see how it is reflected at the proper places inside the `<teiHeader>`. However, the most interesting bits are in the `<body>` part. Apart from regular body content, as illustrated by the `<p>` contents of our minimal TEI customisation, an ODD file contains a specific `<schemaSpec>` element. This element indicates a formal definition of a TEI schema. It has a mandatory `@ident` attribute, supplying an identifier for the schema. The language of the documentation can be specified with an optional `@docLang` attribute; when necessary a `@targetLang` attribute can specify what language to use for element and attribute names. The `@prefix` attribute specifies the prefix that will be reserved for definitions of TEI patterns in the customisation. The `@start` attribute identifies the root element(s) of the
customisation: in this case, it will produce a schema that only allows the `<TEI>` element as root element for adhering TEI documents.

The `<schemaSpec>` element is the heart of any ODD file containing the formal definition of a TEI schema. A schema can be constructed by referring to definitions of existing TEI objects, or-as will be covered later in this tutorial- declaring new objects as well. In this case the schema specification only contains references to predefined TEI modules, with the `<moduleRef/>` element. For each module to be incorporated in the schema, the identifier is provided in the `@key` attribute. This leads to two more observations:

- A minimal TEI customisation isn't empty, but will always refer to the core, tei, header, and textstructure modules. This means that an ODD file without these modules can never define a TEI conformant schema.
- All 505 TEI elements and their attributes are organised thematically in 21 higher-level modules. Compare them to the shelves holding the elements and attributes, in the library analogy developed in the introduction. If a module is selected, by default all elements and attributes of that module are incorporated in the schema.

Indeed, a TEI document must conform to a minimal structure in all cases: it must be contained in a `<TEI>` element in the "http://www.tei-c.org/ns/1.0" namespace, and consist of a `<teiHeader>` element followed by a `<text>` element. Within these elements, all mandatory child structures must be present as well, and so on. This means that a minimal TEI document looks like this:

```
......
```

Since an ODD file is just a regular TEI file with a specific schema specification section inside a `<schemaSpec>` element, it may as well contain a prose documentation of the TEI customisation (rather, an ODD file is explicitly intended to contain both a formal schema specification and documentation). This can be encoded inside the `<body>` part, as with any TEI document, For an excellent example, see the documentation in the TEI Lite ODD file.
Besides this minimal structure, the current selection of modules allows for far more TEI elements, from \texttt{titlePage}, over \texttt{hi} to \texttt{note}, and many more. One way of learning which of the 505 TEI elements are defined by what TEI modules, is studying the prose in the full TEI Guidelines, whose chapters 1 to 22 (apart from chapter 20) each correspond to one of the 21 modules. But the exact contents of a customisation can also be explored in Roma, by selecting the ‘Modules’ tab. This will produce following screen:
Fig. 9.

On the left hand side, all TEI modules are listed. The right hand column lists all modules that are selected in the current customisation. The names of the modules are presented as hyperlinks pointing to a list of elements defined in that module. To see what elements the core module holds, just click on the ‘core’ hyperlink and see all its elements listed on the next screen:

3. Selecting and restricting the TEI model
The same can be done for all other modules on the ‘Modules’ tab. If you want more information on the modules or elements, click on the question mark to navigate to the relevant documentation in the TEI Guidelines, or on its name for technical information.

4 In fact, all elements described in this TBE tutorial module belong to the TEI tagdocs module, documented in chapter 22 of the TEI Guidelines.
Of course, those ‘add’, ‘delete’, ‘include’ and ‘exclude’ options suggest a range of customisation possibilities. These will be covered in the next sections of this tutorial.

Summary
A TEI document must adhere to a minimal structure, with a <TEI> element containing a <teiHeader> and <text> element, and their mandatory substructures. TEI groups its 505 different elements and their attributes in 21 modules. These can be referred to in an ODD file, defining a TEI customisation. An ODD file is just a regular TEI document with a specific element for defining a TEI schema: <schemaSpec>. An identification for the schema must be provided in an @ident attribute. Inside the schema specification, modules can be referenced with a <moduleRef/> element, naming the module in an @key attribute.

3.3 Selecting modules and elements

Back to Alice! Currently, our minimal TBEcustom TEI schema already covers a great deal of the document analysis made at the start of this tutorial:

- The header module contains all header elements for meta documentation.
- The textstructure module contains all elements for marking up front and back matter, the text's body, text divisions, the title page and more.
- The core module has all elements for headings, paragraphs, quotations, citations, page breaks, simple graphical elements, and line groups.

Our quick look over the contents of the core module reveals one lack, however. Although it positively identifies the <graphic/> element for indicating graphical elements, this element does not allow us to describe it, or to connect it with related prose. As introduced in TBE Module 3: Prose, this is what the <figure> element is for. Together with other specialised graphical elements, this element is defined in the figures module. Therefore, we'll add the figures module to our customisation. If you still have the TBEcustom ODD loaded in Roma, you can skip the next step. Otherwise, the way to proceed is as follows. First, point your browser at the Roma web tool. Choose ‘Open existing customization’, locate the ‘TBEcustom.xml’ ODD file with the ‘Browse’ button, and press ‘Start’. Again, we are presented with the ‘Customize’ tab for our customisation, where all metadata (title, schema identifier, author) are neatly picked up from the ODD file. As we want to add a module, move to the ‘Modules’ tab, which will show an identical page as shown in the previous section. Only, this time we'll add the figures module, by pressing the ‘add’ link on its left hand side. This will add the figures module to the list of selected modules in the right column:
By default, all elements of a module are selected for inclusion in the schema. However, inspection of the elements in the figures module (by clicking the ‘figures’ hyperlink in the right hand column of modules in the current customisation) tells us that it basically defines three types of graphical elements: tables, figures and formulae. Since our document analysis did not anticipate any tables in this text, we can exclude all but the figure related ones. This can be done manually, by changing the ‘Include’ form option in the left column to ‘Exclude’ for each element. A quicker way of changing this status globally, is by clicking the ‘Exclude’
hyperlink in the first row of the table (or ‘Include’ to include all elements). Remember, however, to manually include the <figure> and <figDesc> elements again. After picking the elements we want, remember to save your changes by pressing the ‘Save’ button at the bottom of the page. This will reload the page with a success notification at the top:

Fig. 12.

Now, save your customisation as an ODD file (click the ‘Save Customization’ tab). Its <schemaSpec> will be updated to:

3. Selecting and restricting the TEI model
A first thing of notice is the addition of our extra module `figures` with a `<moduleRef/>` element, followed by an exclusion of the table and formula related elements from the `figures` module. This is done in an `<elementSpec>` element specification for each element, documenting the structure, content, and purpose of a single element. Each `<elementSpec>` must identify the element it specifies in an `@ident` attribute. Since all TEI elements are part of TEI modules, this module should be identified in the `<module>` attribute. A third attribute, `@mode`, describes the operation to be performed. This attribute can occur on other elements in a schema specification, with one of four values:

- **add** the current specification is added to the schema
- **delete** the current specification is deleted from the schema
- **change** the current specification changes the declaration of an item with the same name in a schema
- **replace** the current specification replaces the declaration of an item with the same name in a schema

When we generate a TEI schema from this customisation (via the ‘Generate Schema’ tab), this allows us to encode the typical page of the document (the third image above) as follows:

```xml
<schemaSpec ident="TBEcustom" docLang="en" prefix="tei_" start="TEI" xml:lang="en">  
  <moduleRef key="core"/>
  <moduleRef key="tei"/>
  <moduleRef key="header"/>
  <moduleRef key="textstructure"/>
  <moduleRef key="figures"/>
  <elementSpec module="figures" ident="cell" mode="delete"/>
  <elementSpec module="figures" ident="formula" mode="delete"/>
  <elementSpec module="figures" ident="row" mode="delete"/>
  <elementSpec module="figures" ident="table" mode="delete"/>
</schemaSpec>
```

Because all further changes to the ODD file in this TBE tutorial module will affect only its `<schemaSpec>` part, the example fragments will focus on this element.
However, she got up, and began to repeat it, but her head was so full of the <title>Lo-bster-Quadrille</title>, that she hardly knew what she was saying, and the words came very queer indeed:—""

"How the creatures order one about, and make one repeat lessons!" thought Alice; "I might just as well be at school at once." However, she got up, and began to repeat it, but her head was so full of the <title>Lo-bster-Quadrille</title>, that she hardly knew what she was saying, and the words came very queer indeed:—"

"That's different from what I used to say when I was a child," said the Gryphon; "but it sounds uncommon nonsense." said the Mock Turtle; "I should like to have it explained," said the Mock Turtle; "She can't explain it," said the Gryphon; "Go on with the next verse."
"But about his toes?" he persisted.

"How could he turn them out with his nose, you know?"

"It's the first position in dancing."

said; but she was dreadfully puzzled by the whole thing, and longed to change the subject.

"Go on with the next verse,"

repeated impatiently:

"it begins 'I passed by his garden.'"

Alice did not dare to disobey, though she felt sure it would all come wrong, and she went on in a trembling voice:—

Alice did not dare to disobey, though she felt sure it would all come wrong, and she went on in a trembling voice:—

So far for selecting modules and elements. The obvious counterpart, adding new elements, will be dealt with later in this tutorial. First we will focus on attributes.

Summary

Modules can be selected simply by referencing them with a `<moduleRef/>` element, whose `@key` attribute must be used to identify the desired TEI module. By default, all elements of a module are selected for inclusion in the schema. Deleting unneeded elements can be done simply with an `<elementSpec>` element, with an `@ident` attribute indicating the existing name of the TEI element whose declaration is to be altered. The module to which the element belongs must be named in the `@module` attribute. In order to specify that these elements should be deleted, the `@mode` attribute should state `delete`.
3.4 Changing attributes

3.4.1 Changing individual attributes

As the previous example shows, the core module's general `<name>` element could cover our needs for encoding the story's character names and places by making use of its `@type` attribute. This section will address ways of modifying existing TEI attributes.

By default, the `@type` attribute can contain any single keyword from an unspecified list: anything goes as long as it conforms to some syntactic rules (basically, only a few punctuation marks are allowed and it should start with a letter). Apart from that, there is no limit on possible values for the `@type` attribute. However, to facilitate the encoding, we would like to trim down these possibilities for the `@type` attribute of the `<name>` element to following categories: 'person', 'place', and 'animal'. This can be done in Roma, by navigating to the definition of the `<name>` element. In order to do so,

1. load the TBEcustom customisation again if you haven't done so already,
2. move to the ‘Modules’ tab, click the ‘core’ hyperlink and
3. scroll down to the definition of `<name>`.

In order to edit its attributes, click the relevant ‘Change attributes’ hyperlink on the right hand side. This produces a similar page, only now the attributes are listed:
Fig. 13.

By clicking the ‘type’ hyperlink, a page is shown with the definition of the @type attribute. There you can determine whether the attribute should be mandatory or optional, what the datatype and occurrence of its value(s) should be, its default value, a list of possible values, and whether this list is exhaustive or not. Finally, the prose description of the attribute can be given. For our purpose, we can leave most settings unchanged but only add a comma-separated list of the values we expect, in the ‘List of values’ field:

3. Selecting and restricting the TEI model
We might consider defining this value list as exhaustive (closed), but the story’s hazy realm of fantasy and mythology figures might as well impose other categories of their names. Therefore, we'll leave this setting to ‘open list’. Yet, as we anticipate that most names will apply to persons, we define ‘person’ as the default value for the @type attribute:

![Fig. 14.](image_url)
Pressing the ‘Save’ button returns us to the attribute list page. Now, another change we want to make is getting rid of the @nymRef attribute. This is meant to point to a canonical or normalized form of a name, for onomastic purposes. As this is too specific for our purposes with the Alice story, we’ll delete it. This way, it won’t bother us when actually marking up the names in the text. Selection of attributes is similar to selection of elements (see the previous section (p. 15)). Just check the desired option: ‘Include’ (default) to include the attribute to this element in the schema; ‘Exclude’ to delete it. Selecting the ‘Exclude’ option next to the @nymRef attribute will do so, after pressing the ‘Save’ button.

If we save the customisation at this stage (by clicking the ‘Save Customization’ tab), the ODD file gets updated to:

```xml
<schemaSpec ident="TBEcustom" docLang="en" prefix="tei_" start="TEI" xml:lang="en">
  <moduleRef key="core"/>
  <moduleRef key="tei"/>
  <moduleRef key="header"/>
  <moduleRef key="textstructure"/>
  <moduleRef key="figures"/>
  <elementSpec module="figures" ident="cell" mode="delete"/>
  <elementSpec module="figures" ident="formula" mode="delete"/>
  <elementSpec module="figures" ident="row" mode="delete"/>
  <elementSpec module="figures" ident="table" mode="delete"/>
  <elementSpec ident="name" module="core" mode="change">
    <attList>
      <attDef ident="type" mode="change">
        <defaultVal>person</defaultVal>
        <valList type="open" mode="replace">
          <valItem ident="person"/>
          <valItem ident="place"/>
          <valItem ident="animal"/>
        </valList>
      </attDef>
      <attDef ident="nymRef" mode="delete"/>
    </attList>
  </elementSpec>
</schemaSpec>
```

Note how a new `<elementSpec>` element is introduced. Its @ident and @module attributes tell us that it concerns the `<name>` element from the `core` module. This time, however, the @mode attribute is set to `change`, indicating that the existing TEI definition for `<name>` is to be changed. Inside `<elementSpec>` all attribute-related declarations are grouped in an `<attList>` element. For each affected attribute, an `<attDef>` element is added, with the same attributes as
<elementSpec>: @ident to identify the relevant attribute, and @mode to specify the kind of modification. The simplest case is the deletion of the @nymRef attribute: this is simply done by an empty <attDef> element with a delete value for the @mode attribute.

The modification of the value list for the @type attribute will include those parts of its TEI definition that have changed. The default value for an attribute is specified in the <defaultVal> element; in this case it is 'person'. Finally, the list of possible values for the @type attribute is defined in the <valList> element. The value open for the @type attribute on the <valList> element specifies that the list of values is non-exhaustive and can be considered a list of suggested values. Entering a new value in the transcription which is not in this list won't produce an error. This would be the case, however, if the value list were defined as a closed one, by specifying the value closed for the @type attribute. The actual values are enumerated in <valItem> elements, with the actual value as content for the @ident attribute. Note how the <valList> element gets the value replace for the @mode attribute. This indicates that this declaration will entirely override the default TEI definition. Contrast this to the 'change' mode for the higher-level <elementSpec> and <attDef> declarations, which specifies that only those parts of the default TEI definition will be overridden which occur in the ODD file; parts which aren't mentioned are copied over from the default TEI definition.

6 Note, however, that a full attribute definition consists of more fields, like a description, declarations of datatype and occurrence indicators. These are discussed later (p. ______) in this module.

Summary

Individual attributes can be changed inside an <attList> element inside an <elementSpec> declaration with a 'change' mode. Each single attribute is given its own definition inside an <attDef> element. This element too carries the @ident and @mode attributes, respectively for identifying the attribute and specifying the status of the declaration. To delete attributes, indicating the @mode as delete suffices.

Changing attributes requires a change mode. Some of the components of an attribute definition are the default value (<defaultVal>), and a list of possible values (<valList>). Value lists have a @type attribute, stating whether the value list is open-ended (open) or closed (closed). The @mode attribute can specify whether a <valList> declaration merely contains some changes to the existing TEI declaration (change), or replaces the original definition (replace). A value list declares each separate value for an attribute in a <valItem> element, with an @ident attribute providing the contents of this value.

3.4.2 Changing attribute classes

Similar to the organisation of elements in modules, attributes are grouped into classes. This facilitates the definition of elements that share the same attributes, by declaring them as members of an attribute class. For example, all TEI elements are declared as members of......
the `att.global` attribute class, which defines the global attributes `@xml:id`, `@n`, `@xml:lang`, `@rend`, `@rendition`, and `@xml:base`.

As it happens, the `@nymRef` attribute we deleted from the definition of the `<name>` element in the previous section, is defined in such an attribute class, namely `att.naming`, of which `<name>` is declared a member. This information may seem disparate, but is actually easy to find in Roma. To find out the attribute classes an element (in this case, the `<name>` element) belongs to:

1. load the TBEcustom customisation again if you haven't done so already,
2. move to the ‘Modules’ tab, click the ‘core’ hyperlink,
3. scroll down to the definition of `<name>`
4. click the ‘name’ hyperlink

This calls a page with the definition of the `<name>` element. If you scroll down to the ‘Attribute classes’ section, you will see the ‘att.naming’ option selected:
As always in Roma, clicking the name of this attribute class will produce a formal definition of this attribute class:
TEI by Example. Module 8

Fig. 16.

This tells us that the **att.naming** attribute class defines the attribute @nymRef directly, and by reference to the **att.canonical** attribute class declares the @key and @ref attributes for a whole range of name-related elements, of which <name> is only one.

Now, instead of removing the @nymRef attribute only from the <name> element as we did in the previous section, we could as well delete it globally from all these elements at once. This can be done by changing the attribute class itself. In Roma, click the ‘Change Classes’ tab. This calls a list of all attribute classes defined in TEI:
In order to change the `att.naming` class, click the ‘Change Attributes’ hyperlink next to it. This produces a list of all attributes defined by the `att.naming` class (which only contains the `@nymRef` attribute). Now, all we have to do to delete the `@nymRef` attribute from all name-related TEI elements, is selecting the ‘Exclude’ option next to it, and clicking the ‘Save’ button.
Fig. 18.

If we save the customisation again (by clicking the ‘Save Customization’ tab), this produces following ODD file:
As we see, an ODD file can change attribute classes by inserting a `<classSpec>` element. As with other parts of a schema declaration, the mandatory `@ident` attribute identifies the definition of the attribute class, the optional `@module` attribute specifies the module in which the attribute class is defined (in this case, `tei`), and the `@mode` attribute specifies what operation should be performed on the declaration. One other required attribute for `<classSpec>` is `@type`, stating that the class under consideration is an attribute class (atts), or a model class (model) grouping elements that can occur in the same context. The contents of the class specification look familiar: an `<attList>` element groups all attribute declarations defined by the class. Inside the list of attribute declarations, an `<attDef>` element specifies that the `@nymRef` attribute (identified in the `@ident` attribute) should be deleted (see the `@mode` attribute).
Actually, the deletion of the @nymRef attribute from the att.naming attribute class obsoletes the explicit deletion of the same attribute from the <name> attribute. However, it does no harm to have this deletion on both <elementSpec> and <classSpec> levels (they don’t contradict each other). The effect of this customisation can be seen by generating a TEI schema (via the ‘Generate Schema’ tab): this will only validate documents whose name-like elements don’t have a @nymRef attribute.

Summary

Attributes that are defined in an attribute class can be changed globally by changing the class specification in a <classSpec> element. This element should identify the name of the class in an @ident attribute, the module which defines this class in a @module attribute, and the type of class in a @type attribute. As with other schema specification elements, the mode of operation should be stated in a @mode attribute. Inside the <classSpec> declaration of an attribute class, all attribute definitions are grouped in an <attList> element, with an <attDef> declaration for each separate attribute.

......

Be careful, though, when changing global attribute definitions. Some elements may use attributes that are defined in an attribute class directly, without referring to the class. For example, the @type attribute is defined in the class att.typed; however, the <title> element has a @type attribute that is defined literally. Changing something to the definition of @type in the att.typed definition will thus not affect the @type attribute of the <title> element. On the other hand, be aware that changing attribute classes can have very wide ranging effects! Always make sure to study the relevant parts of the TEI Guidelines.