TEI By Example

TEI by Example. Module 8

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 OF CONTENTS

4. Extending TEI .................................................................................................................... 1
   4.1 Adding elements ........................................................................................................... 1
   4.2 Adding attributes ...................................................................................................... 9
   4.3 Other types of extension ......................................................................................... 15
4. Extending TEI

So far, all modifications described were reductions of the general TEI model: either by selecting existing modules, elements, or attributes; or reducing the possible values of attributes. These kinds of modifications can be seen as 'clean' modifications: they define true subsets of the TEI model (provided they adhere to the minimal rules sketched out above (p.______). Put differently: a document that is valid against such a schema will always be valid against the maximal TEI schema.

Not so for customisations that add things to the maximal TEI schema: these could lead to TEI schemas that add new elements and/or attributes, or extend existing TEI definitions in such ways that they are not fully 'backward compatible' with 'native TEI'. In order to facilitate the understanding of TEI customisations, following terms are used:

- **TEI conformant customisation**: subtractive customisation, only restricting and constraining existing components of the TEI model. TEI conformant customisations define schemas that are subsets of the maximal TEI schema.
- **TEI extension**: additive customisation, extending the TEI model with new components. TEI extensions produce schemas that aren't subsets of the maximal TEI schema.

In order to guarantee maximal interoperability for TEI documents, the TEI Guidelines strongly advise to formally separate added elements and attributes from the standard TEI schema. This can be done by defining them in another namespace than the TEI namespace ("http://www.tei-c.org/ns/1.0"). You can freely decide on this namespace; for the purpose of this tutorial, we'll use a dedicated TBE namespace: "http://www.teibyexample.org/".

4.1 Adding elements

As illustrated above (p.______), the TEI core module already provides the <name> element, whose @type attribute can be used to provide more details about the type of name. However, suppose we want to categorise names along more dimensions than just the type of creature they refer to, or we are not entirely satisfied with such a mechanism of subtyping general elements for rather diverse uses. For such cases, the TEI provides a set of more specialised naming elements that add more semantic detail and leave more room for further (sub)typing. They are grouped in the namesdates module. Let's have a look at what namesdates has to offer:

1. load the TBEcustom customisation again if you haven't done so already,
2. move to the ‘Modules’ tab, click the ‘namesdates’ hyperlink

In this long list of specific elements for names and dates, two look particularly interesting: <persName> and <placeName>. In order to avoid overloading our customisation with unneeded elements, let's globally delete all of them first, by clicking the ‘Exclude’ hyperlink in the top row. Next, scroll down to the <persName> and <placeName> definitions, and change the select option to ‘Include’. Finally, scroll down entirely and press the ‘Save’ button at the bottom of the page. This will return us to the ‘Modules’ tab, but this time the namesdates module features in the right hand column of selected modules.
If we generated a schema of this TBEcustom customisation at this point, we would be able to rephrase the different names in our Alice fragment as follows:

```xml
<persName>Alice</persName>
<name type="animal">Lobster</name>
<name type="animal">Gryphon</name>
<name type="animal">Mock Turtle</name>
```

Of course, this dual approach to name encoding, with the general `<name type=""></name>` construct for all but person and place names, and the more specialised `<persName>` and `<placeName>` elements for the latter groups, is undesirable. Therefore, we'll add another dedicated element to our customisation, for specialised encoding of animal names.

In order to add an element in Roma, navigate to the ‘Add Elements’ tab. This contains the following fields:

- **Name**: the name of the element
- **Namespace**: the namespace of the (non-TEI) element
- **Description**: a prose description of the element's meaning
- **Model classes**: a formal declaration of the 'behaviour' of the element: assigning it to a model class will determine the contexts in which it may occur
- **Attribute classes**: a formal declaration of the attributes that will be assigned to the element
- **Contents**: a formal declaration of the content type for the element, either by
  - selecting one of the TEI defined classes in the dropdown list
  - providing a custom Relax NG definition in the text box below

An explanation of all options on this page admittedly is too advanced for the purposes of this tutorial. As always, Roma offers a quite intuitive way to gain information by clicking the names of the different classes in the lists, which will provide you with their formal definition.

**Crosslink**

Exhaustive sources of reference information for the TEI class system can be found in the TEI Guidelines, [Appendix A: Model Classes](#) and [Appendix B: Attribute Classes](#). Datatypes definitions can be accessed from [Appendix E: Datatypes and Other Macros](#). For an in-depth prose description of the entire TEI infrastructure, see Chapter 1 of the TEI Guidelines.

It will be clear by now that adding elements requires conscious thought. Of course, the easiest design choice could be to define a new element as freely as possible, for example by declaring it as member of the model.global model class of global elements that can occur anywhere, and declaring the broadest possible content definition. However, this would leave judgement on the most sensible use of this element completely to the encoder, which would lead to highly unpredictable encoding results and thus reduce the value of this encoding. Therefore,
it is strongly advised to determine the contexts and contents of new elements as precise as possible, in order to ensure that they fit neatly in the TEI semantic model of a text. Consequently, defining new elements requires some insight in the TEI's internals (organisation of modules, model classes, attribute classes, content macros). However, for simple cases like ours we can follow a common sense approach. Since we are modelling a new element for naming animals to the existing `<persName>` TEI element, we can use the declaration of this element as a source of inspiration, or just plainly copy it. Let's have a look at the definition page for `<persName>`:

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

This shows a similar page, only now the relevant options are preselected. Scroll down to the ‘Model Classes’ part, and note how three model classes are selected:

- `model.nameLike` groups elements which name or refer to a person, place, or organisation
- `model.nameLike.agent` groups elements which contain names of individuals or corporate bodies
- `model.persStateLike` groups elements describing changeable characteristics of a person which have a definite duration, for example occupation, residence, or name

These model classes determine the contexts in which the `<persName>` element may occur. If we scroll down further to the ‘Attribute Classes’ section, we see these listed:

- `att.datable` groups attributes for normalisation of names or dates
- `att.editLike` groups attributes for describing the nature of an encoded interpretation
- `att.personal` groups common attributes for names
- `att.typed` groups attributes that allow (sub)classification of an element

These attribute classes define all attributes that can occur on the `<persName>` element. Finally, see how the contents of the `<persName>` element are defined by reference to the TEI `macro.phraseSeq` macro. Macros are nothing more than shortcut names for frequently occurring groups of elements or attribute datatypes. The `macro.phraseSeq` macro defines a sequence of character data and phrase-level elements. Used in the contents definition of the `<persName>` element, this means that this element can contain text intermixed with a whole range of sub-paragraph level elements (`<abbr>`, `<expan>`, `<name>`, `<persName>`,...).

Let's apply these same settings to our new element. Return to the ‘Add Elements’ tab and start defining the new element. A first item is the element's name. There are some of restrictions, but you're safe if the name starts with a letter or underscore and doesn't contain interpunction apart from hyphens, underscores, colons, or full stops. Since our new element for animal names will be analogous to `<persName>` for naming persons, `<animalName>` sounds like a good name. As explained before, adding a non-TEI element is preferably done in its own namespace (in order to avoid e.g. potential name conflicts with existing TEI elements). In the ‘Namespace’ field, we can thus enter "http://www.teibyexample.org/" as namespace declaration. This will allow us to clearly separate the `<animalName>` element from other TEI elements (in the "http://www.tei-
c.org/ns/1.0" namespace) in our transcription of *Alice's Adventures in Wonderland*. Note that the namespace URI (Uniform Resource Identifier) doesn't need to be officially registered and can indeed be any URI (apart from "http://www.tei-c.org/ns/1.0", of course). However, make sure you define a unique namespace for your non-TEI documents (for example, by relating the namespace URI to your project's URI in some way). In the description box, we can enter a prose description for the `<animalName>` element, for example:

```
contains a proper noun referring to an animal
```

Next, we must define how the `<animalName>` element will behave. Copy the ‘Model Classes’ from `<persName>`: tick the boxes next to ‘model.nameLike’, ‘model.nameLike.agent’, and ‘model.persStateLike’. For the ‘Attribute Classes’, select the ‘att.datable’, ‘att.editLike’, ‘att.personal’, and ‘att.typed’ options. The contents of `<animalName>` will consist of the elements and text defined in the `macro.specialPara` macro. This macro is included in the dropdown list, so we can suffice with selecting ‘macro.specialPara’ from this list.

---

1 Note that the ‘Contents’ dropdown list on this page not only includes content macros (starting with `macro.`), but also attribute datatypes (starting with `data.`). These are strictly speaking irrelevant in this context, as attribute datatypes only apply to attribute definitions, not to the definition of an element’s contents. You can safely ignore them here, and scroll down to the content macros (starting with `macro.`).
Save your changes by pressing the ‘Save’ button. This returns us to the ‘Add Elements’ tab, which now consists of a list of added elements:

![Image of added elements]

**Fig. 21.**

Now let’s have a look at the underlying ODD file (click the ‘Save Customisation’ tab):
4. Extending TEI
<elementSpec module="namesdates" ident="forename" mode="delete"/>
<elementSpec module="namesdates" ident="genName" mode="delete"/>
<elementSpec module="namesdates" ident="geo" mode="delete"/>
<elementSpec module="namesdates" ident="geogFeat" mode="delete"/>
<elementSpec module="namesdates" ident="geogName" mode="delete"/>
<elementSpec module="namesdates" ident="langKnowledge" mode="delete"/>
<elementSpec module="namesdates" ident="langKnown" mode="delete"/>
<elementSpec module="namesdates" ident="listNym" mode="delete"/>
<elementSpec module="namesdates" ident="listOrg" mode="delete"/>
<elementSpec module="namesdates" ident="listPerson" mode="delete"/>
<elementSpec module="namesdates" ident="listPlace" mode="delete"/>
<elementSpec module="namesdates" ident="location" mode="delete"/>
<elementSpec module="namesdates" ident="nameLink" mode="delete"/>
<elementSpec module="namesdates" ident="nationality" mode="delete"/>
<elementSpec module="namesdates" ident="nym" mode="delete"/>
<elementSpec module="namesdates" ident="occupation" mode="delete"/>
<elementSpec module="namesdates" ident="offset" mode="delete"/>
<elementSpec module="namesdates" ident="org" mode="delete"/>
<elementSpec module="namesdates" ident="orgName" mode="delete"/>
<elementSpec module="namesdates" ident="person" mode="delete"/>
<elementSpec module="namesdates" ident="personGrp" mode="delete"/>
<elementSpec module="namesdates" ident="place" mode="delete"/>
<elementSpec module="namesdates" ident="population" mode="delete"/>
<elementSpec module="namesdates" ident="region" mode="delete"/>
<elementSpec module="namesdates" ident="relation" mode="delete"/>
<elementSpec module="namesdates" ident="relationGrp" mode="delete"/>
<elementSpec module="namesdates" ident="residence" mode="delete"/>
<elementSpec module="namesdates" ident="roleName" mode="delete"/>
<elementSpec module="namesdates" ident="settlement" mode="delete"/>
<elementSpec module="namesdates" ident="sex" mode="delete"/>
<elementSpec module="namesdates" ident="socsecStatus" mode="delete"/>
<elementSpec module="namesdates" ident="state" mode="delete"/>
<elementSpec module="namesdates" ident="surname" mode="delete"/>
<elementSpec module="namesdates" ident="terrain" mode="delete"/>
<elementSpec module="namesdates" ident="trait" mode="delete"/>
<elementSpec ident="animalName" ns="http://www.teibyexample.org/" mode="add">
<desc>contains a proper noun referring to an animal</desc>
<classes>
  <memberOf key="model.nameLike"/>
  <memberOf key="model.nameLike.agent"/>
  <memberOf key="model.persStateLike"/>
  <memberOf key="att.datable"/>
</classes>

4. Extending TEI
As could be expected, this time the namesdates module is included by a \textless \texttt{moduleRef} \textgreater element. Since we only retained the \texttt{persName} and \texttt{placeName} elements from this module in our \texttt{TBEcustom} customisation, all other 48 elements of this module are explicitly deleted by a dedicated \texttt{elementSpec} element. Each of these has the value \texttt{delete} for its \texttt{@mode} attribute, and identifies the element in the \texttt{@ident} attribute.

Finally, an extra \texttt{elementSpec} element contains the definition for our added \texttt{animalName} element, whose name is given in the \texttt{@ident} attribute. The \texttt{@ns} attribute contains the namespace URI we specified for this element. Finally, the \texttt{add} value for the \texttt{@mode} attribute of this element specification indicates that this declaration is added to the TEI set of definitions. The element specification further contains the prose description of the \texttt{animalName} element in the \texttt{desc} element. The model and attribute classes to which this element is added, are listed in the \texttt{classes} element. Each class declaration consists of a \texttt{memberOf} element, with a \texttt{@key} attribute holding the reference to a TEI model class (starting with 'model.') or attribute class (starting with 'attribute.'). The content of the element is declared within a \texttt{content} element, in the form of a Relax NG expression that either refers to a predefined TEI macro, or defines a new content model. In this case, a Relax NG reference is made to the TEI \texttt{macro.specialPara} macro.

\begin{quote}
\textbf{Summary}

Elements can be added to the existing TEI model by declaring them with an \texttt{elementSpec} element, with the value \texttt{add} for its \texttt{@mode} attribute. As with other element specifications, the \texttt{@ident} attribute must give the name of the element. Specific to added elements is the use of the \texttt{@ns} attribute, whose value should provide a unique namespace URI for this element, different from the default TEI namespace ("http://www.tei-c.org/ns/1.0"). A prose description of the element can be given in a \texttt{desc} element. The structural behaviour and attributes of an element are defined in the \texttt{classes} element,
\end{quote}

\begin{quote}
\texttt{......}
\end{quote}

2 Syntactically, the TEI model does not require you to use different namespaces for non-TEI elements, but strongly advises you to: this is the safest way to avoid name collisions. You can for example define a \texttt{name xmlns="http://www.teibyexample.org"} variant that differs from the standard TEI \texttt{name} element. For the sake of clarity, however, this is not really advisable.
4.2 Adding attributes

So far, we have customised our schema for the transcription of the Alice text in such a way that we can distinguish between person, place, and animal names, either as types of the general <name> element, or by means of the TEI elements <persName> and <placeName>, and the non-TEI element <placeName xmlns="http://www.teibyexample.org/">. We fine-tuned all elements belonging to the att.naming class by deleting the unneeded @nymRef attribute from this class.

For our specific analysis of Alice’s Adventures in Wonderland we would like to experiment with a basic way of adding further interpretation of the ontological status of the referents of the names in this fictitious story: it could be interesting to analyse the characters in terms of the kind of reality they exist in. A possible place for such information could be the @type and @subtype attributes of the att.typed class. However, we would like a more specific label for this kind of information, and reserve these TEI attributes for possible different categorisations in the future. Therefore, we want to add a new attribute to our customisation. Similar to deleting attributes, adding new ones can happen on two levels:

- element level: attributes may be added to an individual element, which will apply to this element only
  # This is accessible in Roma from the individual element's definition (via the ‘Modules’ tab), where you can click the ‘Change Attributes’ hyperlink. In ODD, it will affect the attribute definition of an <elementSpec> element.
- class level: attributes may be added to an attribute class, which will apply to all elements that are member of this class
  # This is accessible in Roma from the attribute class's definition (via the ‘Change Classes’ tab), where you can click the ‘Change Attributes’ hyperlink. In ODD, it will affect the attribute definition of a <classSpec> element.

In this case, information on the ontological status of names' referents not only applies to personal and place names, but also to our recently added animal names, names in general, and by extension all kinds of referring strings. This suggests the att.naming attribute class as a good place to add this attribute.

In order to extend an attribute class with new attributes in Roma, click the ‘Change Classes’ tab, locate the desired attribute class (in our case, the att.naming class) and click the ‘Change Attributes’ hyperlink on its right hand side. This calls an overview of the attributes in this class (note how the @nymRef attribute still is excluded from our modification). This list is preceded by an hyperlink labelled ‘Add new attributes’. This hyperlink takes us to an empty attribute definition page, where the same types of information can be declared as we saw before (p.
the attribute’s name, occurrence indicator, contents, default value, openness, a possible list of values, and a prose description. Before we start defining the attribute, a little thought is needed on its design. Following examples could illustrate different possibilities:

\[
\begin{align*}
\langle \text{persName} \text{ fantastic}^\text{"no"} \rangle & \text{Alice}\langle/\text{persName} \\
\langle \text{animalName} \text{ realistic}^\text{"0.5"} \rangle & \text{Mock Turtle}\langle/\text{animalName} \\
\langle \text{animalName} \text{ ontStatus}^\text{"mythological"} \rangle & \text{Gryphon}\langle/\text{animalName}
\end{align*}
\]

Attributes could be designed as binary choices taking some form of truth value, as categories taking some kind of degrees on a scale, as neutral labels taking a list of keywords, or many more. As we are in the early stages of the encoding project, and feel this ontological classification is still experimental, we can anticipate that categories are likely to pop up, merge, or be adapted along the way. Therefore, it makes most sense to design it as a general semantic field, allowing for an open-ended list of keywords. Considering these requirements, a sensible name for this attribute could be ‘ontStatus’. In Roma, this can be declared next to the field labelled ‘Add a new attribute’. In the ‘Description’ field we'll describe it as:

\[
\text{describes the ontological status of a name's referent}
\]

We’ll define it as an optional attribute by selecting ‘yes’ for the ‘Is it optional?’ field. The other fields define the actual content of the attribute. For this example, suppose that an initial (experimental) categorisation for the ontological status of the people, places and animals in the Alice story could look like this:

- **realistic**: the referent can / could occur in the extra-textual reality
- **mythological**: the referent does not exist in real life, but belongs to a major mythology
- **fantastic**: the referent belongs to an idiosyncratic fantasy universe

However, it is prone to be extended with other categories, and would probably allow more categories to be applied simultaneously, for names referring to ambiguous creatures or places. This analysis obviously translates into an open list (option ‘no’ for ‘Closed list?’) of these values (‘List of values’):

\[
\text{realistic,mythological,fantastic}
\]

Finally, the datatype and occurrence for the attribute's value can be declared in the ‘Contents’ field. The declaration of the list of values suggests the TEI datatype `data.enumerated`, which is explicitly designed to define a single word from a list of possibilities. If we decide to dismiss the list and allow for any word, other viable datatype options would be `data.word`, or `data.name`, depending on the range of characters we want to allow. Although we defined the attribute as optional, we wouldn't like it to be empty when used on an element. Therefore, we can specify
the value '1' after the >= sign, specifying that at least one value is expected for this attribute. To allow for an unlimited combination of values from the list in the attribute, the value 'unbounded' can be selected after the <= sign.

Fig. 22.

To save these changes to our customisation, press the ‘Save’ button, which will take us to the list of attributes for the att.naming class again. Note how our freshly defined @ontStatus attribute is listed, and can be further manipulated (further changes, include / exclude, delete).
Fig. 23.

After saving the ODD file (by clicking the ‘Save Customisation’ tab), we'll notice that the `<classSpec>` element is updated to:
As we added the attribute to the `att.naming` attribute class, the corresponding `<attDef>` declaration is added to the list of attribute declarations of the corresponding `<classSpec>` element. As before, the class specification's `@mode` is set to `change`, indicating that only the specifications present in this ODD file will update the existing TEI definitions. Inside the `<attList>` section, the `@nymRef` attribute still is deleted, in accordance with our previous changes. However, there's a new `<attDef>` element for our `@ontStatus` attribute (identified in the `@ident` attribute), this time with the value `add` for its `@mode` attribute. Although not explicitly specified, the `@ontStatus` will be optional in our customisation. This could have been stated explicitly with the optional `@usage` attribute, which defaults to the value `opt`, but can indicate other usage patterns as well (for example, `req` for required attributes). Inside the attribute definition, the `<desc>` element contains the prose description of the attribute. The `<datatype>` section declares that the `@ontStatus` attribute should have minimally one value (`@minOccurs = 1`), while there's no limit on the frequency of its values (`@maxOccurs = unbounded`). The actual datatype of the attribute is defined by the contents of `<datatype>`. As the underlying TEI schema is expressed in Relax NG, this will consist of elements of the Relax NG namespace. In this case, reference is made to a TEI datatype definition with the name ‘data.enumerated’, which basically restricts the possible values to strings consisting of words or a limited range of punctuation marks. Combined with the declarations in `@minOccurs` and `@maxOccurs`, this means that the `@type` attribute for `<name>` can only contain a single term consisting of word characters and some punctuation marks.
Finally, the list of possible values is given inside `<valList>`, which is declared as an open list (`@type = open`).

In the introduction to this section we stated that extending the TEI always leads to TEI document models that are broader than and hence may be incompatible with the TEI model. For maximal separation of the standard TEI model from extensions, the TEI guidelines therefore advice to define extensions in their own namespace. We already did so when adding new elements in the previous section. However, it seems that Roma does not provide an option in its interface to define namespaces for added attributes. Yet, it is possible, indeed advisable, to do so. Therefore, we'll manually add a namespace declaration to the `TBEcustom` ODD file. Analogous to the namespace declaration in element specifications, we can add a `@ns` attribute to the `<attDef>` declaration for our `@ontStatus` attribute:

```xml
<attDef ident="ontStatus" mode="add" ns="http://www.teibyexample.org/"
<desc>describes the ontological status of a name's referent</desc>
<datatype minOccurs="1" maxOccurs="unbounded">
    <rng:ref name="data.enumerated"/>
</datatype>
<valList type="open">
    <valItem ident="realistic"/>
    <valItem ident="mythological"/>
    <valItem ident="fantastic"/>
</valList>
</attDef>
```

**Summary**

Adding attributes is done within an `<attDef>` declaration inside the `<attList>` declaration of all attributes for an element `<elementSpec>` or attribute class `<classSpec>`. The addition is specified in the `add` value for the `@mode` attribute of the attribute definition; the name of the attribute is given in the `@ident` attribute. Additionally, `<attDef>` specifies the usage of the attribute within `@usage` (opt for optional attributes, req for mandatory ones). In order to distinguish added attributes from standard TEI ones, it is highly recommended to manually declare a dedicated namespace in the `@ns` attribute (although Roma currently doesn't include this option in its graphical interface). An attribute definition typically contains a prose description in `<desc>`, an indication of the attribute's datatype in `<datatype>` (referring to one or more of the predefined TEI datatypes), and a list of possible values in `<valList>`. Such lists may be
specified as open or closed in the @type attribute. Each predefined attribute value is declared in the @ident attribute of a separate <valItem> element.

4.3 Other types of extension

Besides these common cases of TEI extension by adding elements and attributes, TEI can be extended in both in more subtle and complex ways:

- existing TEI elements can be renamed
- content models of existing TEI elements can be broadened
- datatypes and occurrence indicators of attributes can be broadened
- existing TEI elements can be redefined to different model classes

Most of these make use of the mechanisms covered in this tutorial. However, these kinds of modifications are considered advanced topics and are not treated in this introductory tutorial. For more information, you are referred to chapters 22 and 23 of the TEI Guidelines, or one of these tutorials:

- TEI: Getting Started with ODDs
- TEI: Using Roma