TEI BY EXAMPLE



MODULE 8: CUSTOMISING TEI, ODD, ROMA

Ron Van den Branden

Edward Vanhoutte

Melissa Terras

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

Last updated September 2020

Licensed under a Creative Commons Attribution ShareAlike 3.0 License

TABLE OF CONTENTS

6. Extending TEI..... 1

6.1 Adding Elements..... 2

6.2 Adding Attributes..... 14

6.3 Other Types of Extension..... 21

6. 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 define true subsets of the TEI model, as long as they don't remove any mandatory elements or attributes. In other words: TEI documents that are valid according to a schema generated from such reductive TEI customisations (like ours so far), will always be valid against the **`tei_all`** customisation, which includes all TEI elements and attributes: see https://tei-c.org/release/xml/tei/custom/odd/tei_all.odd.

Not so for customisations that add things to the maximal TEI schema: these will produce 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

> a reductive customisation, that only restricts and constrains existing components of the TEI model (without removing mandatory components). TEI conformant customisations define schemas that are a subset of the maximal TEI schema. Everything documented in a TEI conformant customisation is documented in the TEI Guidelines.

TEI extension

> additive customisation, extending the TEI model with new components. TEI extensions produce schemas that aren't subsets of the maximal TEI schema. TEI extensions define concepts that are not documented in the TEI Guidelines.

It is very common to create TEI extensions: often, projects can have specific encoding needs for which no ideal TEI solution exists. In order to avoid "tag abuse," where existing TEI elements or attributes are used in a way that more or less stretches their actual definition in the TEI Guidelines, projects can define their own means for encoding such phenomena in a TEI extension. These can be used internally in a project, or—if a proposed encoding solution has broader use—, it can eventually be incorporated in the TEI Guidelines, through the process of Feature Requests in the TEI source code repository.

While TEI extensions are a very common way of customising TEI, in order to guarantee maximal interoperability for TEI documents, the TEI Guidelines strongly advise to formally separate elements and attributes added in a TEI extension from the "native" ones in 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/these extension namespace/s; for the purpose of this tutorial, we'll use a dedicated TBE namespace: **`http://teibyexample.org/ns/TBE`**. Notice that this namespace URI (Uniform Resource Identifier) doesn't need to be officially registered, nor lead to an actual web resource. It can indeed be any URI, as long as it provides a unique identification for the non-TEI parts of your TEI customisation (so it must definitely differ from **`http://www.tei-c.org/ns/1.0`**). In the context of an encoding project, it is often a good idea to relate the namespace URI to the project's URI in some way.

6.1 Adding Elements

So far, our **`TBEcustom`** customisation includes a generic <name> element (from the **`core`** module) for identifying names in our *Alice* encoding project. Apart from this generic element, TEI defines a set of more specialised elements for encoding names, in the **`namesdates`** module. These elements add more semantic detail and leave more room for further (sub)typing. Let's use Roma to have a look at what **`namesdates`** has to offer:

1. load the **`TBEcustom`** customisation again in Roma if you haven't done so already, and click the "Customize ODD" button,
2. in the "Elements" tab, click the "by module" button at the top right of the Roma screen and scroll down to the elements of the **`namesdates`** class.

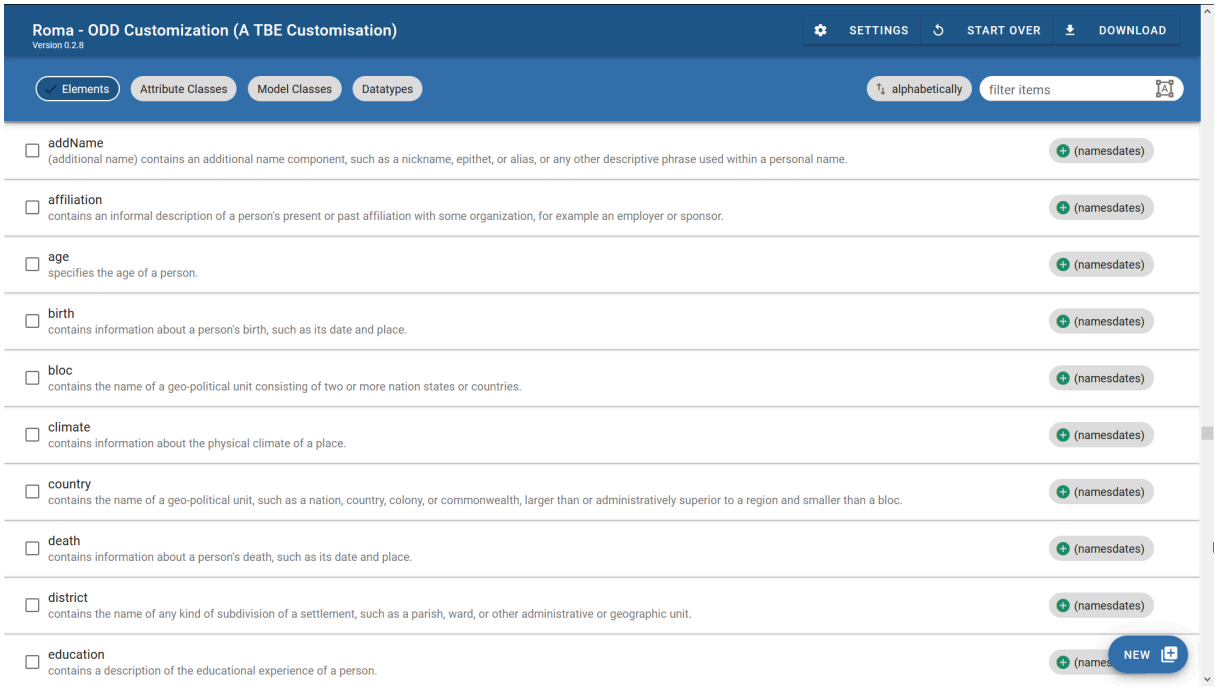

Figure 22. Grouping all elements of a TEI module in Roma.

In this long list of specific elements for names and dates, two look particularly interesting for our purposes: <persName> (as an alternative for <name type="person">) and <placeName> (as an alternative for <name type="place">). Let's include them in our customisation: tick the boxes in Roma, and store the ODD file via the "Download" > "Customization as ODD" button at the top right of the Roma screen. As you probably have expected, this will add following instruction to the ODD file:

```
<moduleRef xmlns="http://www.tei-c.org/ns/1.0" key="namesdates" include="persName placeName"/>
```

If we generated a schema of the **`TBEcustom`** customisation at this point, we would be able to rephrase the encoding of the different names in our *Alice* fragment as follows:

```
<persName xmlns="http://www.tei-c.org/ns/1.0">Alice</persName>
<name xmlns="http://www.tei-c.org/ns/1.0" type="animal">Lobster</name>
<name xmlns="http://www.tei-c.org/ns/1.0" type="animal">Gryphon</name>
<name xmlns="http://www.tei-c.org/ns/1.0" type="animal">Mock Turtle</name>
```

Of course, this dual approach to name encoding, with the specialised <persName> and <placeName> elements for person and place names respectively, and the general <name> element with @type="animal" for animals, is undesirable. Therefore, we'll extend our TEI customisation with a dedicated element for encoding animal names. As before, the remainder of this section first outlines the necessary steps in Roma, and next discusses the resulting ODD file.

In Roma, new components (elements, classes, datatypes) can be added to a TEI customisation by clicking the "NEW" button at the bottom right of the screen:



Figure 23. The "NEW" menu for adding new TEI components in Roma.

We'll be creating a new element for animal names, so click the "Element" button of the pop-up menu. This opens a form with three fields. Let's fill them:

- "Name": the name of the element. By analogy to <persName> and <placeName>, let's make this <animalName>
- "Module": the TEI module this new element will belong to. Since this is a specialised naming element, let's select **namesdates**.
- "Namespace": the namespace of the (non-TEI) element. As we've mentioned before, let's make this **http://teibyexample.org/ns/TBE**

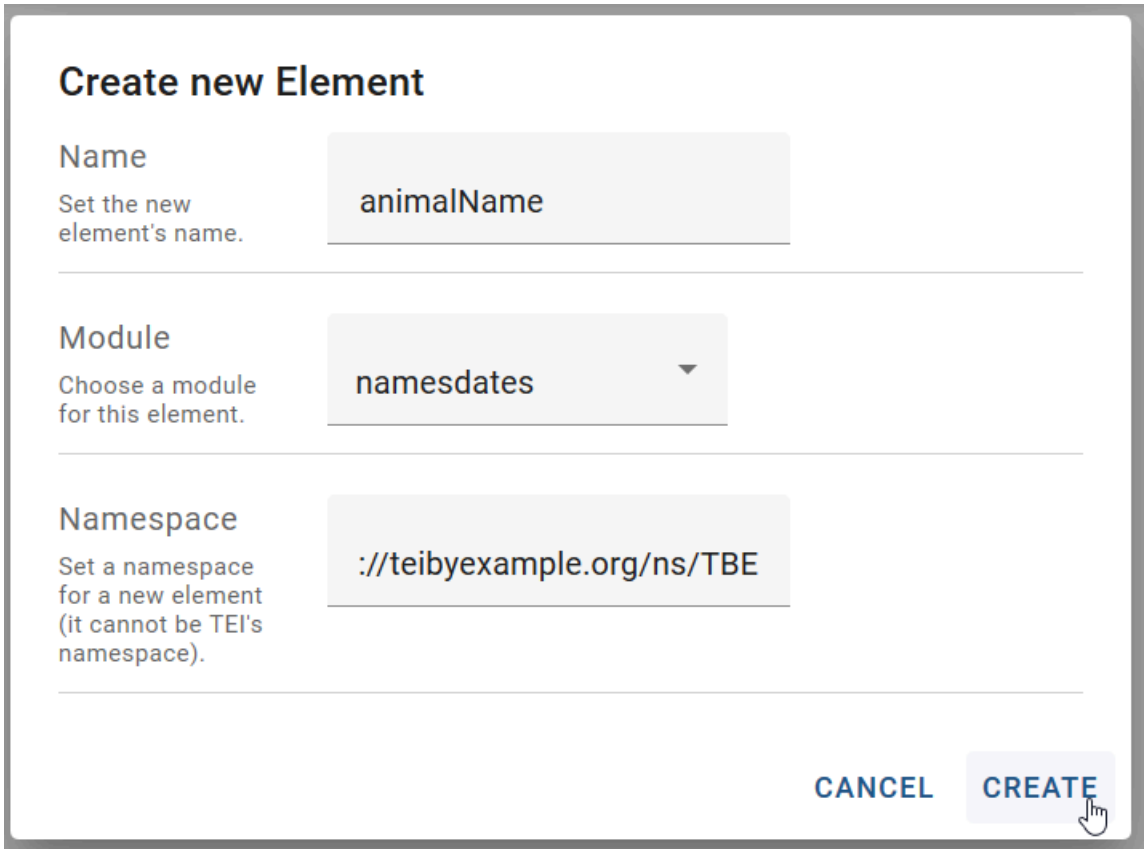

Figure 24. "Create new element" form in Roma.

After clicking the "Create" button, our new element is born! If an ODD file is generated at this point, it would be expanded with following instructions:

```
<elementSpec xmlns="http://www.tei-c.org/ns/1.0" ident="animalName" ns="http://
teibyexample.org/ns/TBE" mode="add" module="namesdates">
  <classes/>
  <attList/>
</elementSpec>
```

This tells that our customisation defines a new element <animalName> in the **http://teibyexample.org/ns/TBE** namespace, which will be made available in the **namesdates** module.

Yet, with empty <classes> and <attList> children, this element definition is still utterly useless. We have to define a number of essential properties:

- the **attribute classes** this new element belongs to
- the **model classes** this new element belongs to
- the **content** of this new element

> REFERENCE
>
> Exhaustive 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 The TEI Infrastructure of the TEI Guidelines.

These are important decisions to make, which require conscious thought, as well as an understanding of the TEI class system. Fortunately, we can learn by example: since we are modelling a new element for animal names to the existing <persName> TEI element, we can use the declaration of the latter 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 in Roma if you haven't done so already, and click the "Customize ODD" button,
2. move to the "Elements" tab and scroll down to the definition of <persName>.

The "Documentation" button shows the description for the <persName> element.



Figure 25. Display of the documentation for the <persName> element in Roma.

The “Attributes” button shows us the attribute classes. If only the top-level attribute classes are considered, that aren’t defined inside other attribute classes, (the others are being included automatically when their superclass is included), this list can be reduced to:

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

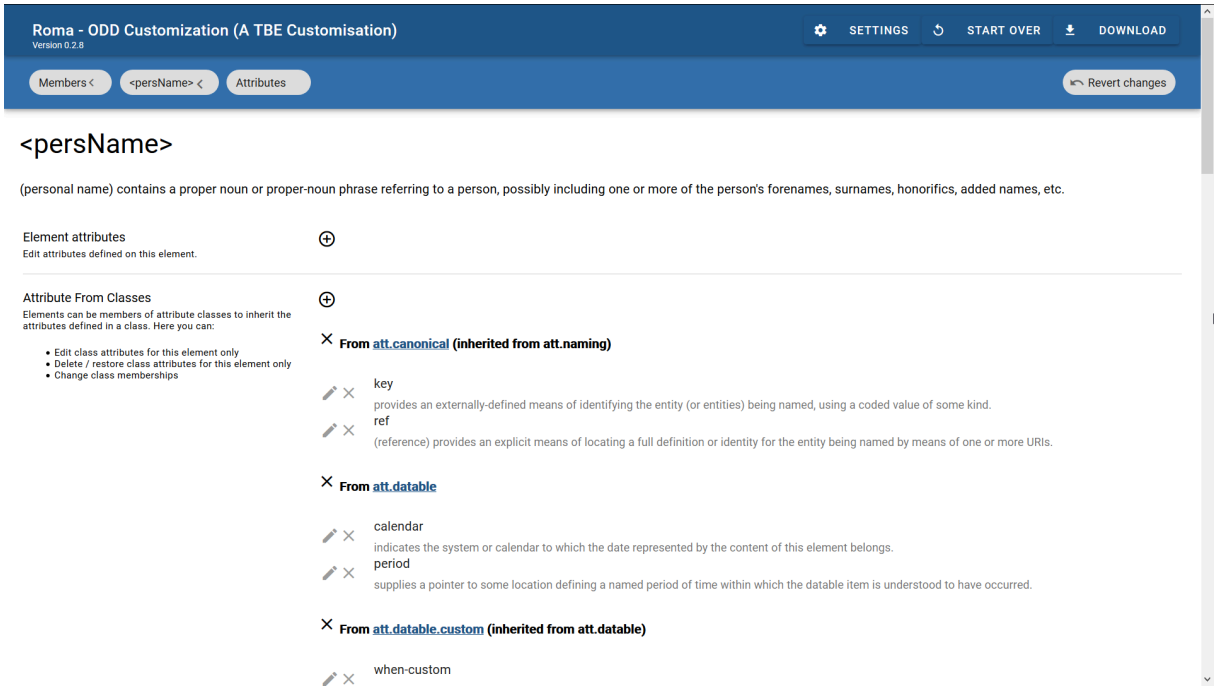

Figure 26. Display of the attributes for the `<persName>` element in Roma.

The “Class Membership & Content Model” button shows us the model classes of which <persName> is a member:

- 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

This means that <persName>, and all other members of these model classes, can occur wherever TEI elements specify their contents in terms of these classes.

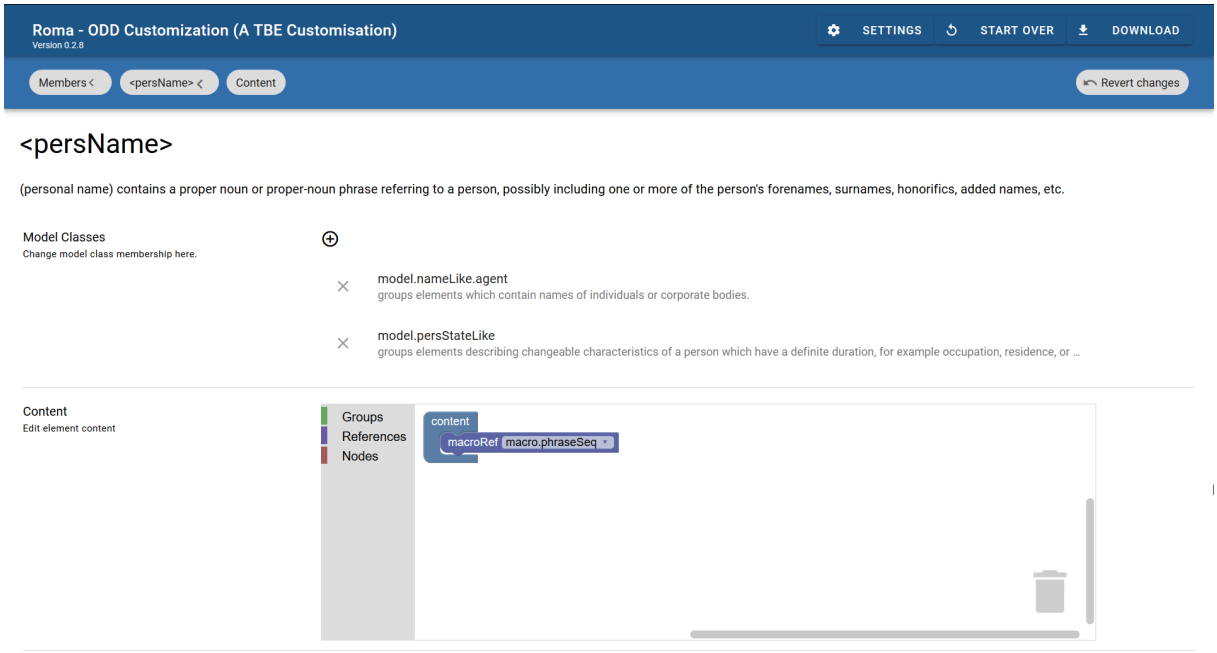

Figure 27. Display of the class membership for the `<persName>` element in Roma.

The content model of the <persName> element is represented graphically in Roma. It refers to the macro.phraseSeq macro, which represents a predefined sequence of character data and phrase-level elements. This means that <persName> can contain text intermixed with a whole range of sub-paragraph level elements (<abbr>, <expan>, <name>, <persName>, ...).

Let's copy these same declarations to our new element. Return to the "Elements" tab and click the `<animalName>` element. In the description box, we can enter a prose description for the `<animalName>` element, for example:



Figure 28. Editing the documentation for the `<animalName>` element in Roma.

Clicking the button labeled "`<animalName>` <" on top left of the Roma screen, gets us back to the overview of the `<animalName>` definition. A click on the "Attributes" button shows us that its attribute list is empty, as we saw already in the generated ODD file. In order to declare its membership of the 5 attribute classes we had identified in the definition of <persName>, start by clicking the plus sign in the "Attribute from Classes" row. Next, filter or scroll through the list of attribute classes, and click the plus sign next to the ones we want to add.

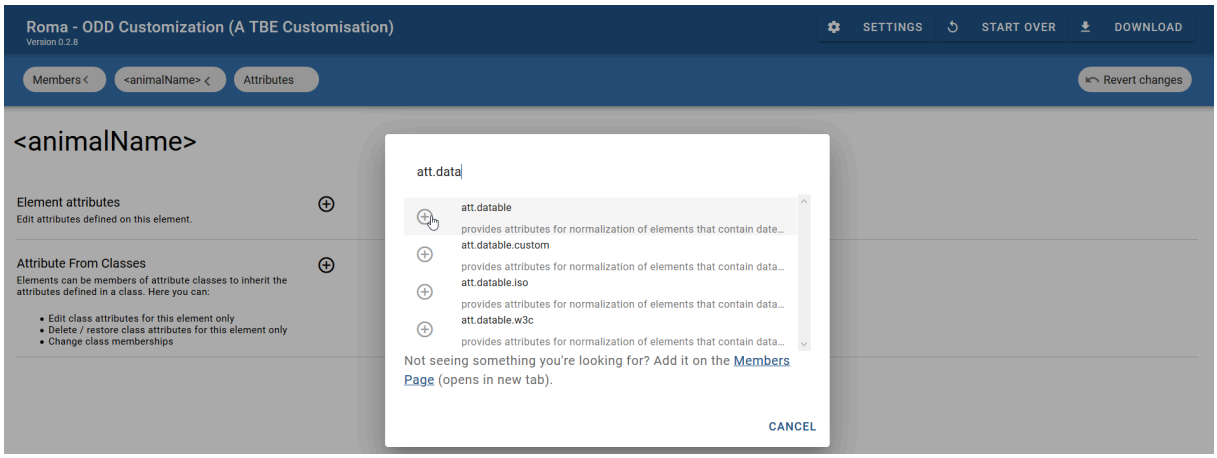

Figure 29. Adding the `<animalName>` element to attribute classes in Roma.

Finally, return to the overview of the `<animalName>` definition, and click the "Class Membership & Content Model" button. In order to define how our `<animalName>` element will behave, click the plus sign in the "Model Classes" row and select the 2 model classes we had identified by clicking the plus sign next to their name.

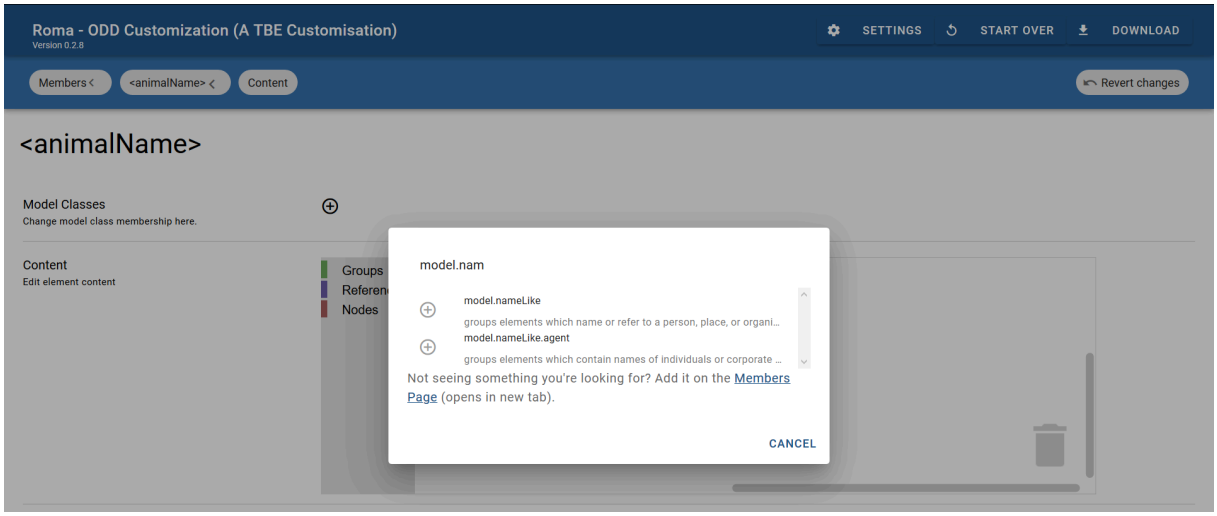

Figure 30. Editing the class membership of the `<animalName>` element in Roma.

Next, its content can be defined by first clicking one of the "Groups," "References," or "Nodes" blocks. This will produce an overlay menu, with a graphical indication of the different types of ODD constructs that can be used for selecting individual nodes or combining them in groups, or referencing one of the predefined TEI macros or classes. We'll be inserting a reference to the macro.phraseSeq macro, so we'll click the "References" block.

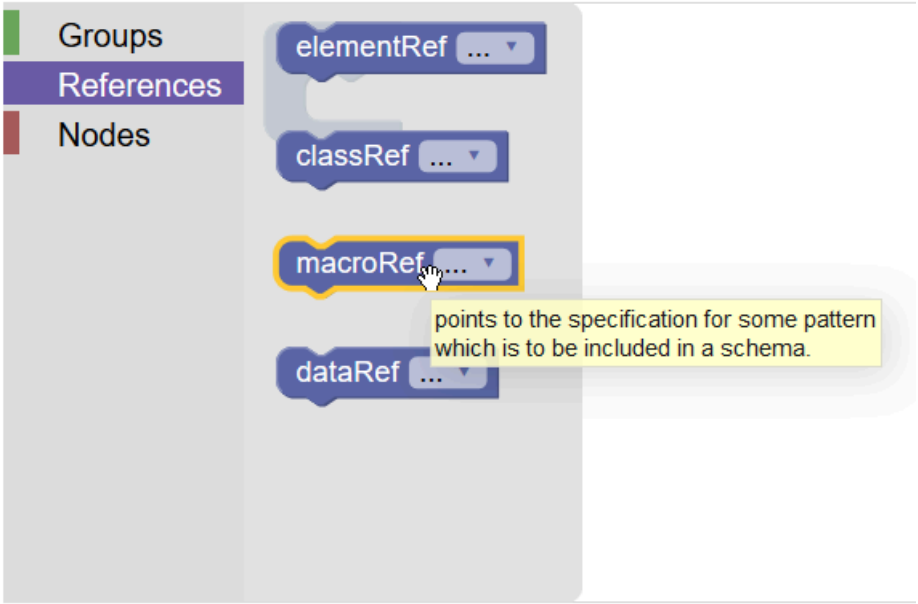

Figure 31. Selecting a macro for the content model of the `<animalName>` element in Roma.

We want to include a reference to a macro, so “macroRef” is the block we’ll want to add. This is done by *dragging* that block into the graphical editor screen of Roma. Next, clicking the arrow in that “macroRef” block calls a dialog window where the macro.phraseSeq macro can be selected via the plus sign next to it. Finally, make sure to drag this populated “macroRef” block *into* the empty space of the blue “content” block in the graphical editor (you’ll hear a little “click” sound if it succeeds):



Figure 32. Adding a macro to the content model of the `<animalName>` element in Roma.

Now it’s time to inspect our ODD customization: store the ODD file via the “Download” > “Customization as ODD” button at the top right of the Roma screen. The resulting ODD file looks as follows:

```
<schemaSpec xmlns="http://www.tei-c.org/ns/1.0" ident="TBEcustom" start="T
EI" prefix="tei_" targetLang="en" docLang="en">
  <moduleRef key="figures" include="figDesc figure"/>
  <moduleRef key="header" include="teiHeader fileDesc titleStmt publicationStmt source
  Desc"/>
  <moduleRef key="core" include="p title emph lg l pb pubPlace publisher q quote name
   graphic"/>
  <moduleRef key="textstructure" include="TEI text body titlePage docTitle docImprint
   docDate docAuthor byline div"/>
  <moduleRef key="tei"/>
  <moduleRef key="namesdates" include="persName placeName"/>
  <elementSpec ident="name" mode="change">
    <classes mode="change">
      <memberOf key="att.datable" mode="delete"/>
      <memberOf key="att.editLike" mode="delete"/>
      <memberOf key="att.personal" mode="delete"/>
    </classes>
    <attList>
      <attDef ident="subtype" mode="delete"/>
      <attDef ident="nymRef" mode="delete"/>
```

```
    <attDef ident="type" mode="change">
      <valList type="closed" mode="change">
        <valItem mode="add" ident="place">
          <desc versionDate="2020-04-23" xml:lang="en">used for place names</desc>
        </valItem>
        <valItem mode="add" ident="person">
          <desc versionDate="2020-04-23" xml:lang="en">used for person names</desc>
        </valItem>
        <valItem mode="add" ident="animal">
          <desc versionDate="2020-04-23" xml:lang="en">used for animal names</desc>
        </valItem>
      </valList>
    </attDef>
    <attDef ident="cert" mode="delete"/>
    <attDef ident="resp" mode="delete"/>
    <attDef ident="source" mode="delete"/>
  </attList>
</elementSpec>
<classSpec ident="att.naming" type="atts" mode="change">
  <attList>
    <attDef ident="nymRef" mode="delete"/>
  </attList>
</classSpec>
<elementSpec ident="animalName" ns="http://teibyexample.org/ns/
TBE" mode="add" module="namesdates">
  <desc>contains a proper noun referring to an animal</desc>
  <classes>
    <memberOf key="att.datable"/>
    <memberOf key="att.editLike"/>
    <memberOf key="att.global"/>
    <memberOf key="att.personal"/>
    <memberOf key="att.typed"/>
    <memberOf key="model.nameLike.agent"/>
    <memberOf key="model.persStateLike"/>
  </classes>
  <content>
    <macroRef key="macro.phraseSeq"/>
  </content>
```

```
  </elementSpec>
</schemaSpec>
```

Example 5. ODD definition of a new (non-TEI) element (download).

We see how an extra <elementSpec> element is added to the ODD file, with the definition for our newly added <animalName> element. Its name is given in the @ident attribute, and @mode="add" indicates that this is a new element. The @ns attribute contains the namespace URI we specified for this element.

The description of our brand new <animalName> element is provided in a <desc> element. Its class membership is declared in a <classes> element; each of the attribute and model classes we had selected, is listed in a separate <memberOf> element, whose @key attribute identifies the relevant class. The content of the element is declared within a <content> element. Since we included the shortcut to the macro.phraseSeq macro, this is recorded in a <macroRef> element, whose @key attribute identifies the macro.

In order to give an impression of the convenience of macros, let's see what this macro.phraseSeq macro resolves to when the ODD file is being processed:

```
<alternate xmlns="http://www.tei-c.org/ns/1.0" minOccurs="0" maxOccurs="unbounded">
  <textNode/>
  <classRef key="model.gLike"/>
  <classRef key="model.qLike"/>
  <classRef key="model.phrase"/>
  <classRef key="model.global"/>
</alternate>
```

We'll not go into full details here, but you'll be able to understand that this macro defines a sequence with zero (@minOccurs="0") or more (@maxOccurs="unbounded") combinations of plain text nodes (<textNode/>), or the elements from the 4 model classes identified in <classRef>. For full coverage of the definition of content models: see section 22.5.1.1 Defining Content Models: TEI of the TEI Guidelines.

SUMMARY

Elements can be added to the existing TEI model by declaring them with an <elementSpec> element, with the value "add" for its @mode attribute. The @ident attribute must give the name of the element. Specific to added elements is the use of the @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 <desc> element. The structural behaviour and attributes of an element are defined in the <classes> element, containing <memberOf> declarations for each model or attribute class to which the element is added, identified with the @key attribute. The content of the element is declared in the <content> element, containing either references to TEI macros, or hand-crafted sequences of nodes, elements, or element classes.

6.2 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 with specific @type values of the generic <name> element (namely "person", "place", or "animal"), or by means of the TEI elements <persName> and <placeName>, and the non-TEI element <animalName>. 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 information on 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 prefer 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 via the “Attributes” button in an element’s definition (via the “Elements” tab), where you can click the plus sign in the “Element attributes” row. In ODD, it will affect the attribute definition inside 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 “Attribute Classes” tab), where you can click the desired class name, click the “Attributes” button, and add additional attributes to the class. In ODD, it will affect the attribute definition inside 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.

Let’s head over to Roma! Click the “Attribute Classes” tab for our TEI customisation, click the att.naming class, and next hit the “Attributes” button.

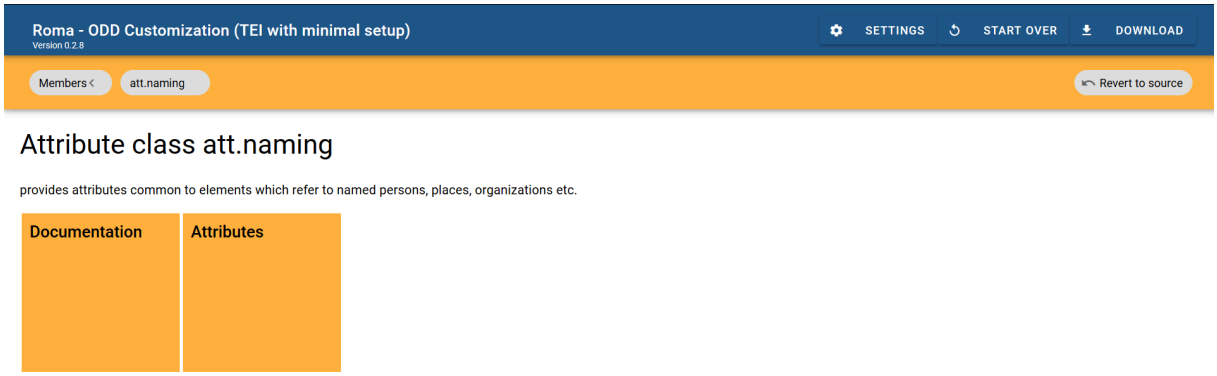

Figure 33. Editing an attribute class for a TEI customisation in Roma.

This calls an overview of the attributes in this class (notice how the @nymRef attribute still is excluded from our modification). Adding a new attribute to an attribute class can be done by clicking the plus sign in the “Class attributes” row. This produces a dialog window:

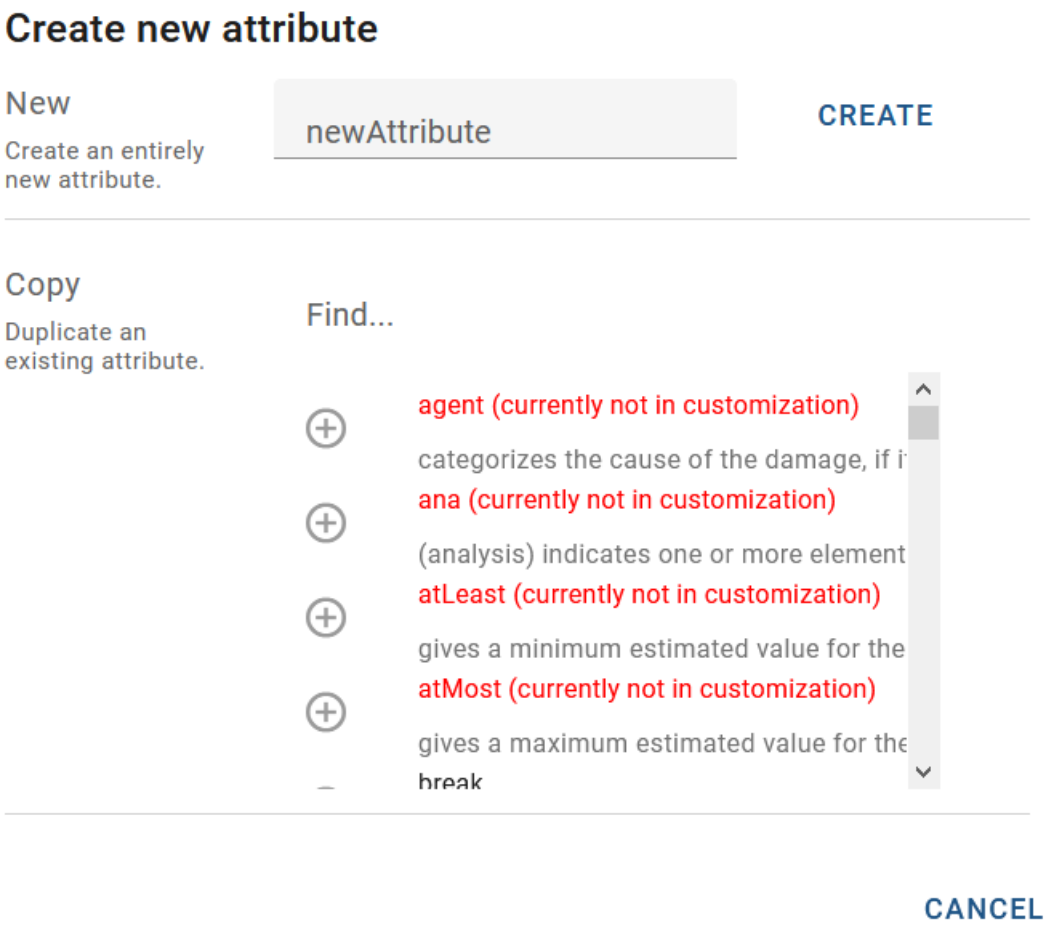

Figure 34. Creating a new attribute in Roma.

This form asks us first for the name of this new attribute. Before we start defining the attribute, some thought is needed on its design. Following examples could illustrate different possibilities:

```
<persName xmlns="http://www.tei-c.org/ns/1.0" fantastic="no">Alice</persName>
<animalName xmlns="http://www.tei-c.org/ns/1.0" realistic="0.5">Mock
 Turtle</animalName>
<animalName xmlns="http://www.tei-c.org/
ns/1.0" ontStatus="mythological">Gryphon</animalName>
```

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." Let's fill that name, and click the "Create" button. Now, this (empty) attribute definition is added to the list of "Class attributes." In order to define this attribute, click the pencil icon next to its name. This will produce a page similar to the one we've seen before (when restricting the values of the `@type` attribute for <name>). We'll define this `@ontStatus` attribute as an `"optional"` attribute, by selecting this in the dropdown box next to "Usage." In the "Description" field, we can provide a description, such as:

```
describes the ontological status of a name's referent
```



Figure 35. Editing documentation for a new attribute in Roma.

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 a "semi-open" list: choose that option for the "Values" field. Attribute lists in ODD can have three types:

"closed"

only the values defined in <valList> are permitted

"semi"

the values defined in <valList> are treated as suggested values, but others are allowed

"open"

any value is allowed (as long as it complies with the datatype); values in <valList> are treated as mere examples

Suggested values can be entered in the text field of the "Values" field, and next pressing the plus sign next to the value. Let's add the values mentioned above, as well as their description.

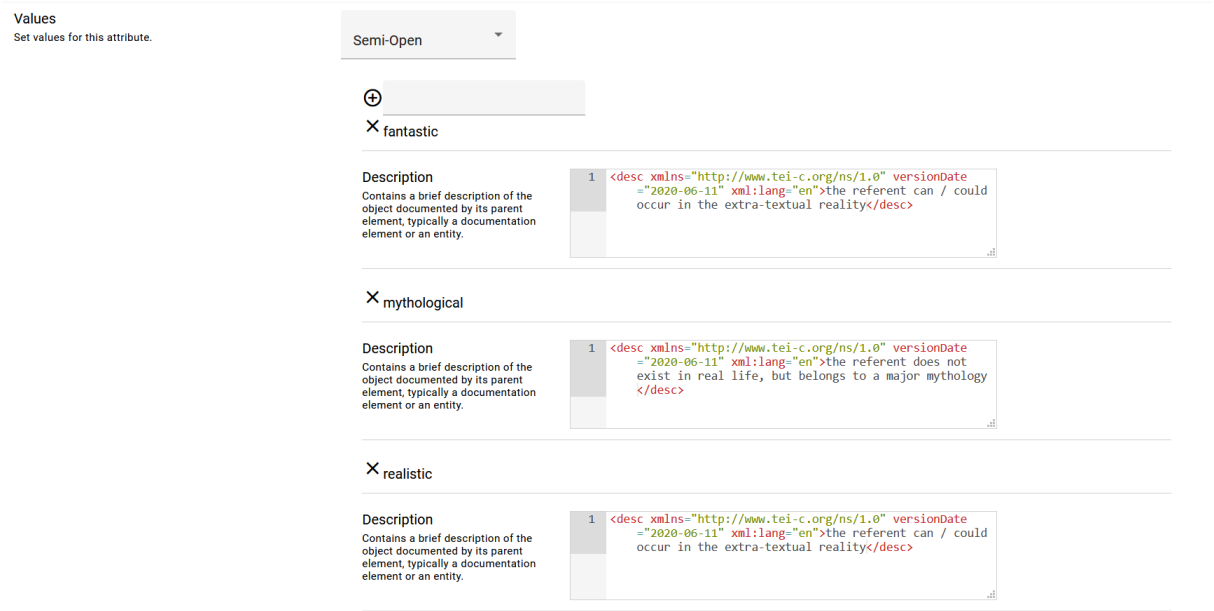

Figure 36. Defining attribute values for a new attribute in Roma.

Finally, the datatype for the attribute's value can be declared in the "Datatype" field. The TEI datatype teidata.enumerated, which is explicitly designed to define a single word from a list of possibilities, seems the best fit.

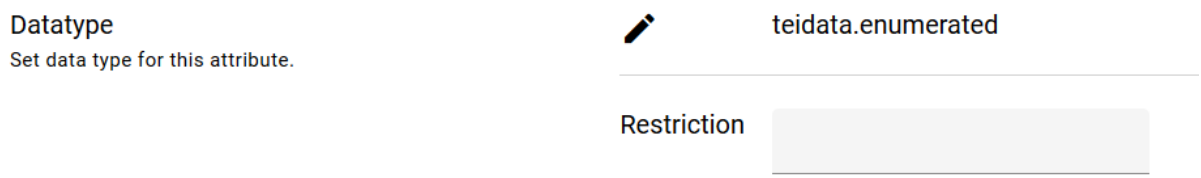

Figure 37. Defining the datatype for a new attribute in Roma.

In the introduction to this section we stated that extending the TEI always leads to TEI document models that are broader than and hence possibly incompatible with the standard TEI model. For maximal separation of extension features from the standard TEI model, the TEI Guidelines therefore advice to define extensions in their own namespace. We already did so when adding new elements in the previous section. Let's declare the same namespace **`http://teibyexample.org/ns/TBE`** in the "Namespace" field for our `@ontStatus` attribute:

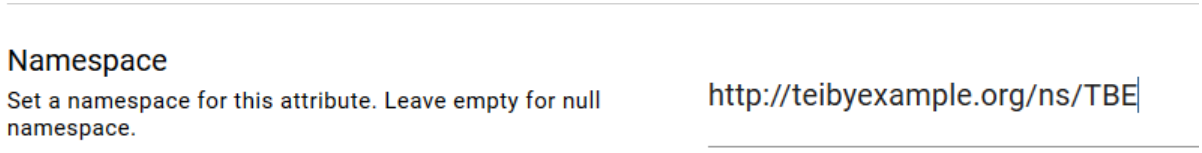

Figure 38. Defining a namespace for a new attribute in Roma.

To save these changes as an ODD file, click the "Download" > "Customization as ODD" button at the top right of the Roma screen. We'll notice how a defintition of the new `@ontStatus` attribute now is added to the <classSpec> declaration for the att.naming attribute class:

```
<classSpec xmlns="http://www.tei-c.org/ns/1.0" ident="att
.naming" type="atts" mode="change">
  <attList>
    <attDef ident="nymRef" mode="delete"/>
    <attDef ident="ontStatus" mode="change" usage="opt" ns="http://teibyexample.org/ns/
    TBE">
      <desc versionDate="2020-04-28" xml:lang="en">describes the ontological status of a
       name's referent</desc>
      <datatype maxOccurs="unbounded">
        <dataRef key="teidata.enumerated"/>
      </datatype>
      <valList mode="change" type="semi">
```

```
        <valItem mode="add" ident="realistic">
          <desc versionDate="2020-04-28" xml:lang="en">the referent can / could occur in
           the extra-textual reality</desc>
        </valItem>
        <valItem mode="add" ident="mythological">
          <desc versionDate="2020-04-28" xml:lang="en">the referent does not exist in
           real life, but belongs to a major mythology</desc>
        </valItem>
        <valItem mode="add" ident="fantastic">
          <desc versionDate="2020-04-28" xml:lang="en">the referent can / could occur in
           the extra-textual reality</desc>
        </valItem>
      </valList>
    </attDef>
  </attList>
</classSpec>
```

Example 6. ODD definition of a new (non-TEI) attribute in an attribute class (download).

Since we added the `@ontStatus` 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 attribute is set to `"change"`, indicating that only the declarations 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 `"change"` for its @mode attribute (since there was no existing definition for this element, this has the same effect as @mode="add"). The `"opt"` value for the @usage attribute indicates that the `@ontStatus` attribute will be optional in our customisation (required attributes would have `"req"`).

Inside the attribute definition, the <desc> element contains the prose description of the attribute. The datatype of the attribute is defined in <datatype>. By means of a <dataRef> element, it is referring to the existing TEI datatype definition `"teidata.enumerated"` (which is given as the value for its @key attribute). This datatype

basically restricts the possible values to strings consisting of words or a limited range of punctuation marks. Combined with the declarations in `@maxOccurs`, this means that the `@ontStatus` attribute for `<animalName>` can contain a white space separated list of word characters and some punctuation marks.

> REFERENCE
>
> This introductory tutorial doesn't cover the advanced inner mechanisms of TEI in full; for more information you can read section 1.4.2 Datatype Specifications of the TEI Guidelines, or the reference section in Appendix E: Datatypes and Other Macros.

Finally, the list of possible values is given inside <valList>, which is declared as a semi-open list. Each of these values is given in its own <valItem> element, with a description of that value in <desc>.

> SUMMARY
>
> Adding attributes is done within an <attDef> declaration inside the <attList> declaration, either in the definition of a single element (<elementSpec>) or an 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 declare a dedicated namespace in the @ns attribute. 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 `"closed"`, `"semi"`, or `"closed"` in the @type attribute. Each predefined attribute value is declared in the @ident attribute of a separate <valItem> element.

6.3 Other Types of Extension

Besides these common cases of TEI extension, involving the addition and deletion of elements and attributes, TEI can be extended in both 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 types of customisations 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. Documentation Elements and 23. Using the TEI of the TEI Guidelines.