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

1. Introduction....1
2. Some Terminology....2
3. Customising TEI: Why and How?....4
4. Starting from a Minimal Schema....9
4.1 The Power of ODD....11
4.1.1 Generating a Schema....11
4.1.2 Generating Documentation for a Customisation....13
4.1.3 Generating an ODD File....15
4.2 What Does a Minimal TEI Customisation Tell Us?....17
4.3 Inspecting a Customisation in Roma....21
5. Selecting and Restricting the TEI Model....23
5.1 Selecting Modules and Elements....23
5.2 Changing Attributes....29
5.2.1 Changing Individual Attributes....29
5.2.2 Changing Attribute Classes....36
6. Extending TEI....43
6.1 Adding Elements....45
6.2 Adding Attributes....57
6.3 Other Types of Extension....64
7. Summary....65
8. What's Next?....72

1. Introduction

Throughout its history, TEI has grown into a complex and encompassing system, allowing you to express your view on a text in a very flexible way, ranging from rather general statements on the textual structure to highly specific analyses of all kinds of textual phenomena. Currently (at version 4.0.0), the TEI defines no less than 580 different elements and it is hard to imagine a document that would need them all. On the other hand, it is much easier to imagine a document that would need just *that* element that isn't present in the current set of elements defined by TEI.

The TEI community anticipated such concerns and explicitly designed TEI P5 as:

- a highly modular system, allowing users to cherry-pick the parts they need
- an extensible system, allowing users to add new elements and attributes or modify existing ones

Put differently, TEI very much resembles a library of text concepts where you can walk in, stroll through shelves filled with TEI element and attribute definitions, and choose exactly those that suit your document analysis. When you check out at the counter, they will all be collected and put in a nice bag, reading "schema for documents of [your name]." What's more, even if you have brought your own elements and attributes, they will be included in the same schema! You take your receipt, which is a blueprint listing all elements and attributes included in your schema so you can revisit your selection afterwards, walk home and happily start encoding your texts with your very own TEI schema. In the TEI world, this library visit is called "TEI customisation." As it is a visit you will have to repeat often in order to fine-tune a tailored schema for your encoding projects, this tutorial will guide you through the most relevant steps for doing so.

A couple of elements in the above analogy will be the focus of this tutorial, so allow us to elaborate them a bit more:

- In this tutorial, the general term *TEI schema* will be used for any formal representation of the elements and attributes you expect in a document. TEI schemas can be expressed as *Relax NG* schemas, *W3C Schema*s, or *DTD*s (don't worry, you ordered for one of these at check-out). In general, you don't have to work on these schemas themselves; they are rather meant as auxiliary files for your XML editing / processing software to validate your document(s) and make sure they conform to the rules.
- Even more important than a schema is the "receipt" for your TEI schema. This will allow you to remember the choices made and facilitate you to share your schemas with others. In the TEI world, such a "blueprint" is just another TEI document with specific elements, and is called an *ODD* (One Document Does it all).

- It's important to know that, as of TEI P5, there is no "fixed" monolithic one-size-fits-all TEI schema. Instead, you are supposed to create your own before you can start encoding TEI texts. In this sense, customisation is a built-in prerequisite for using TEI. Testimony to this centrality is the fact that TEI maintains a specific tool for easing this customisation process. It is called "Roma," and accessible as a user-friendly web form at https://roma.tei-c.org/. Consider it an electronic librarian.

[1]

This tutorial won't discuss the different TEI schema formats, but instead focus on both the formal ODD vocabulary for expressing TEI customisations, and the process of editing an ODD file with the Roma tool. In doing so, this tutorial will be the odd one out: at the same time slightly more conceptual than the other ones, and more concrete, using and introducing the Roma web tool along the way throughout the examples.

2. Some Terminology

As we have seen in Module 0: Introduction to Text Encoding and the TEI, section 5.2, all TEI elements and their attributes are organised thematically in 21 higher-level modules. If we compare the TEI specification to a library, modules are the library shelves holding the elements and attributes. Each of these modules is documented in full in a dedicated chapter of the TEI Guidelines; for an overview of which chapters correspond to which modules, see section 1.1 TEI Modules of the TEI Guidelines. Most TEI modules define attributes and elements. During time, the TEI specification has grown into an ecosystem of hundreds of interrelated elements and attributes. In order to make for easier organisation and maintenance, they have been grouped into *classes*. Each class is a group of elements and attributes. Therefore, the TEI defines two kinds of classes:

.....

1 Currently, an update of the Roma tool is in the making. It is available at a separate URL (https://roma.tei-c.org/), until it will replace the current Roma tool. So long, the "old" Roma page at https://roma.tei-c.org/ provides a link to the beta version. As this beta version is meant to replace the current Roma version eventually, we have decided to use it in this tutorial, and to refer to it as "Roma."

Attribute classes

A group of attributes that can occur in the same place in a TEI document. The names of attribute classes all start with `att.`, followed by a name that gives an indication of the group of attributes it contains. For example, the global attributes `@rend`, `@xml:id`, and `@n` (among others) are all defined in the attribute class att.global, which name indicates that it defines global attributes, available on all TEI elements.

Model classes

A group of elements that can occur in the same place in a TEI document. The names of model classes all start with `model.`, followed by a name that gives an indication of the group of elements it contains. For example, <p> and <ab> are being grouped in the model class model.pLike, which holds all paragraph-like elements.

Classes can refer to other classes, so it becomes possible to develop a fine-grained hierarchy of element and attribute groups that all share some common features, but add their own specific features, depending on the sub-classes they belong to.

In the organisation of TEI modules, the **`tei`** module is a bit special, in that it does no more than providing the definitions of the attribute and model classes that are being used in the definitions of elements and attributes in other TEI modules. Besides these classes, the **`tei`** module defines some other low-level constructs that are being used in the definition of actual elements and attributes:

Macros

A kind of "shortcut," combining a number of frequently co-occurring model classes into a logical group of elements that can occur in the same hierarchical contexts in a TEI document. The names of macros all start with `macro.`, followed by a name that gives an indication of the logical organisation of the elements it groups. For example, the macro macro.paraContent groups elements that can occur inside paragraphs and similar elements.

Datatypes

A set of rules for the values of attributes. The names of datatypes all start with `teidata.`, followed by a name that gives an indication of the datatype. For example, the datatype teidata.count specifies a rule for attribute values that must be positive integers. When an attribute definition refers to this datatype, its value must be a positive integer.

This system of building blocks can then be used to define the core of what the TEI is all about: elements and attributes for marking up information in or about texts. For example: the TEI element <name> defines itself by:

- declaring itself as a member of the *TEI module* **`core`**. This means that when this module is included in a TEI customisation, the <name> element will be available by default.
- declaring itself as a member of the *attribute classes* att.global, att.personal, att.datable, att.editLike, and att.typed. This means that <name> will receive the attributes defined in all those classes (at least, those that are included in the TEI customisation).
- declaring itself as a member of the *model class* model.personPart. This means that it will be able to occur inside all TEI elements that define the members of this model.personPart as its contents.
- declaring its contents as the elements grouped in the *macro* macro.phraseSeq. This means that all elements included in this macro can occur inside <name>.

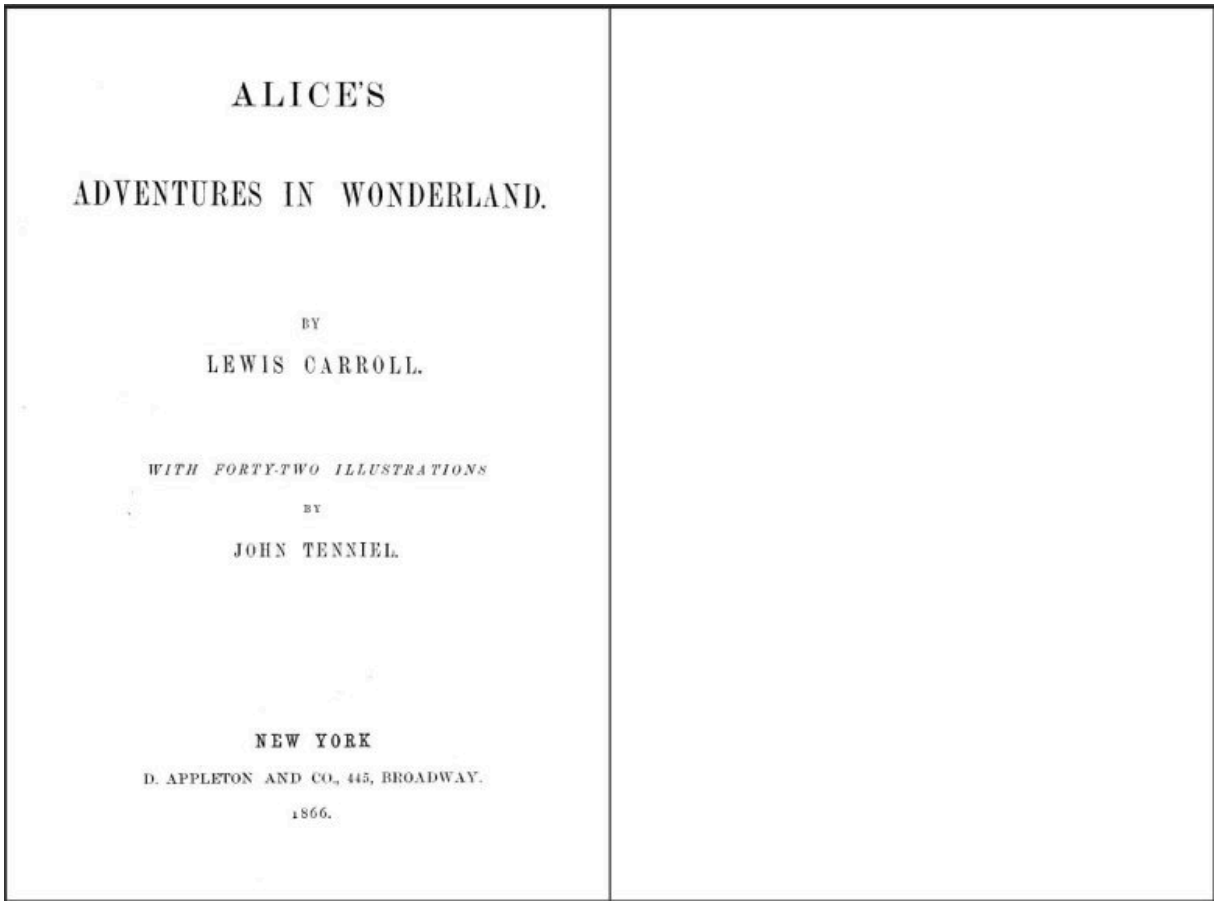
How these element and attribute are defined, is explained in this TEI by Example tutorial. It will cover some of the very specific elements that make up the **`tagdocs`** module that is documented in section 22: Documentation Elements of the TEI Guidelines. Yet, instead of a purely theoretical explanation of this rather technical part of the TEI Guidelines, we will illustrate this in a hands-on fashion, building a mini TEI customisation for a small encoding project, with the help of the Roma tool. Let's dive in!

3. Customising TEI: Why and How?

As with all other TBE tutorial modules, we'll start from a concrete text example. This time, we'll consider Lewis Carroll's *Alice in Wonderland*. Suppose we're at the start of a project that will encode this text. To get a sense of the structure of the document, here are the first pages:

ALICE'S

ADVENTURES IN WONDERLAND.



ALICE'S

ADVENTURES IN WONDERLAND.

BY

LEWIS CARROLL.

WITH FORTY-TWO ILLUSTRATIONS

BY

JOHN TENNIEL.

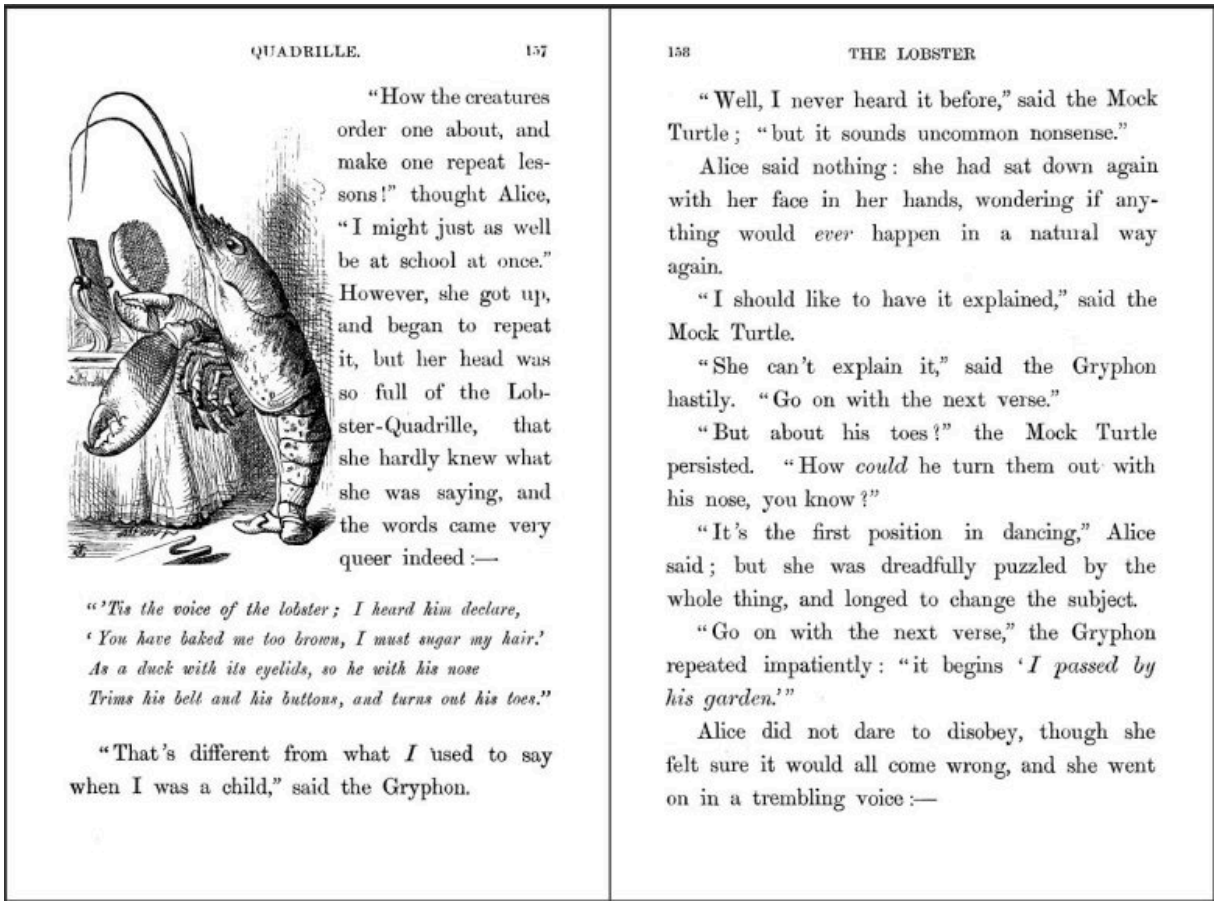
NEW YORK

D. APPLETON AND CO., 445, BROADWAY.

1866.

Figure 1. First pages of *Alice's Adventures in Wonderland*.

A typical page looks like this:



QUADRILLE. 157

"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 Lobster-Quadrille, that she hardly knew what she was saying, and the words came very queer indeed:—

"'Tis the voice of the lobster; I heard him declare,
'You have baked me too brown, I must sugar my hair.'
As a duck with its eyelids, so he with his nose
Trims his belt and his buttons, and turns out his toes."

"That's different from what *I* used to say when I was a child," said the Gryphon.

158 THE LOBSTER

"Well, I never heard it before," said the Mock Turtle; "but it sounds uncommon nonsense."

Alice said nothing: she had sat down again with her face in her hands, wondering if anything would *ever* happen in a natural way again.

"I should like to have it explained," said the Mock Turtle.

"She can't explain it," said the Gryphon hastily. "Go on with the next verse."

"But about his toes?" the Mock Turtle persisted. "How *could* he turn them out with his nose, you know?"

"It's the first position in dancing," Alice said; but she was dreadfully puzzled by the whole thing, and longed to change the subject.

"Go on with the next verse," the Gryphon 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:—

Figure 2. A typical page of *Alice's Adventures in Wonderland*.

As always, the first step in approaching the encoding of a text is a document analysis, considering this is a prose work consisting of chapters.

CHALLENGE

Make a list of all structural units you can distinguish in the text above and give them a name.

SOLUTION

Some of the significant structural elements to be distinguished are these:

- The document
- The title page
- Document title
- Chapters
- Headings
- (Sub)Divisions
- Paragraphs
- Quotations
- Citations
- Page breaks
- Figures
- Line groups

In addition, we are especially interested in the semantic encoding of the names of the different characters and places.

This document analysis allows us to get an idea of the phenomena we want to encode and how to express them in TEI; for a suggestion of the corresponding TEI elements we refer you to the other TBE tutorial modules or the full TEI Guidelines.

However, after completion of this document analysis, we're not quite ready to start encoding our TEI version of *Alice's Adventures in Wonderland*. Unless you know TEI by heart, it will be very hard to produce a valid TEI transcription, without a TEI schema.

There are two options to get a TEI schema:

1. Pick one of the sample TEI customisations, available at https://tei-c.org/Guidelines/Customization/, in the format of your choice. TEI provides a number of basic customisations, each with their own focus on different aspects of the TEI model. Depending on your needs, these may provide the elements and attributes you need, or you may want to build on them.
2. Create your own schema with the Roma web tool.

Although the existing TEI customisations in many cases provide all that's needed for the encoding of common textual phenomena, and the study of these customisations provides an excellent source of information on customising and modifying TEI, in this tutorial we'll start from scratch. This way, all concepts can be introduced one at a time, and you will get to learn how to actually interpret existing customisations. Alongside the Roma tool itself, the most important concepts of TEI customisation will be treated, split in two strands:

- selection and restriction of existing TEI elements and attributes
- extension of the TEI model with new elements and attributes

SUMMARY

Encoding TEI texts with a TEI schema involves customising the TEI. Either you use one of the pre-cooked TEI customisations, or start creating your own with the Roma web tool. Customisation can be roughly divided into selection and restriction of the existing TEI model, and extension of the TEI model.

4. Starting from a Minimal Schema

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

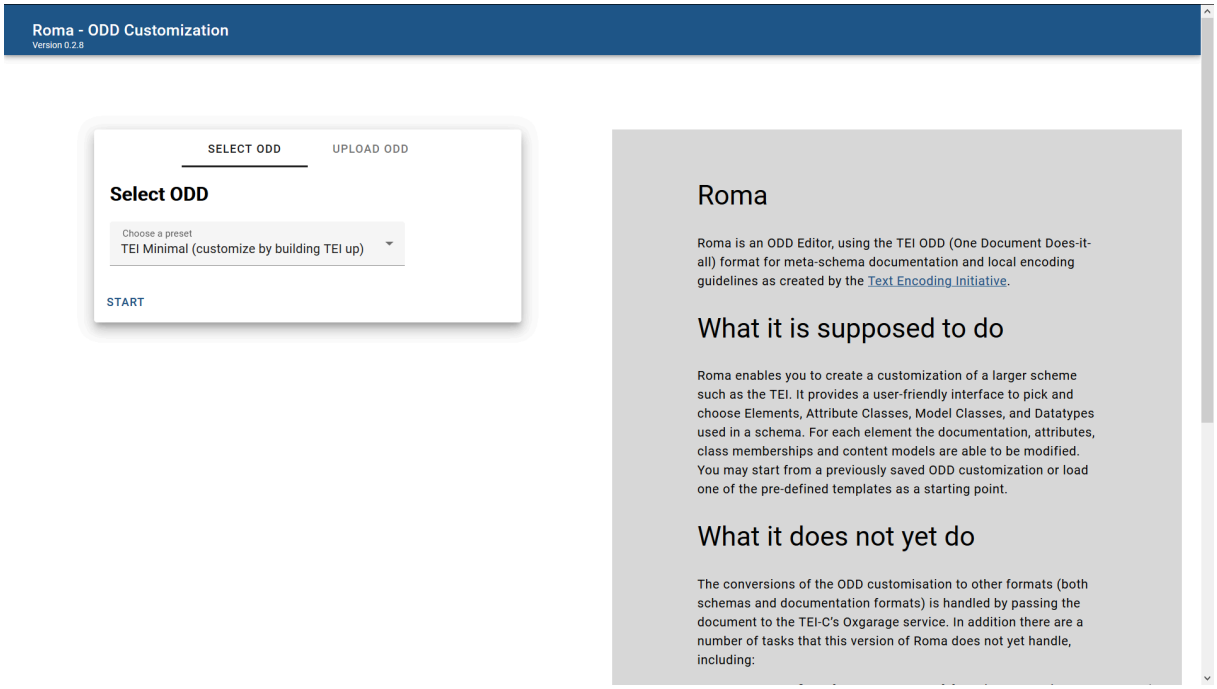


Figure 3. The start screen of Roma.

This is the start screen for new customisations, offering you a choice between two options:

- "Select ODD": start creating a customisation from one of the "official" TEI customisations
- "Upload ODD": start creating a customisation from an existing ODD file on your file system

For the purpose of this tutorial, we'll set out from a minimal customisation. Open the dropdown list and select the "TEI Minimal (customize by building TEI up)" template. This is a very simple customisation that only selects the TEI elements needed for producing valid TEI documents. Select this template and press the "Start" button.

This will produce a configuration screen for your fresh TEI customisation:

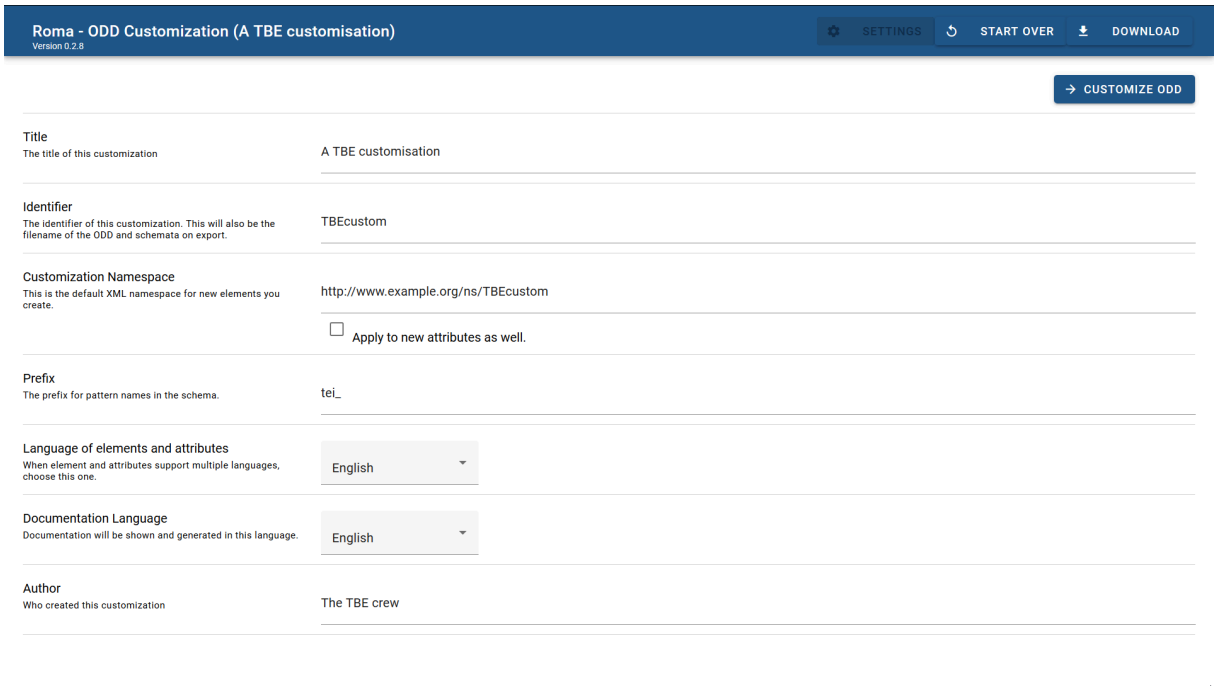


Figure 4. ODD configuration screen in Roma.

Here, you can enter details about your customisation file. 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" field.

That's it! We have created a first TEI customisation already, or at least an abstract representation in the Roma web interface. In order to use your TEI customisation in your project, you'll not only have to derive a schema to tell your XML parser what you consider a valid TEI document for your project, but most likely also human-readable documentation to tell your project team how they should encode the text phenomena they'll encounter in your project.

4.1 The Power of ODD

Remember how ODD stands for "One Document Does it all"? You can derive a number of different schema and documentation output formats from the single ODD file you're editing! The "Download" button at the top right of the Roma screen lets you choose between 3 types of output:

- ODD files
- schema files, in a number of flavours
- documentation files, in a number of output formats

Since these are frequent operations when customising TEI, they are treated in separate sections below.

4.1.1 Generating a Schema

Select the "Download" button at the top right in the Roma screen, and choose the schema language of your choice (the various schema language output formats are highlighted in yellow):

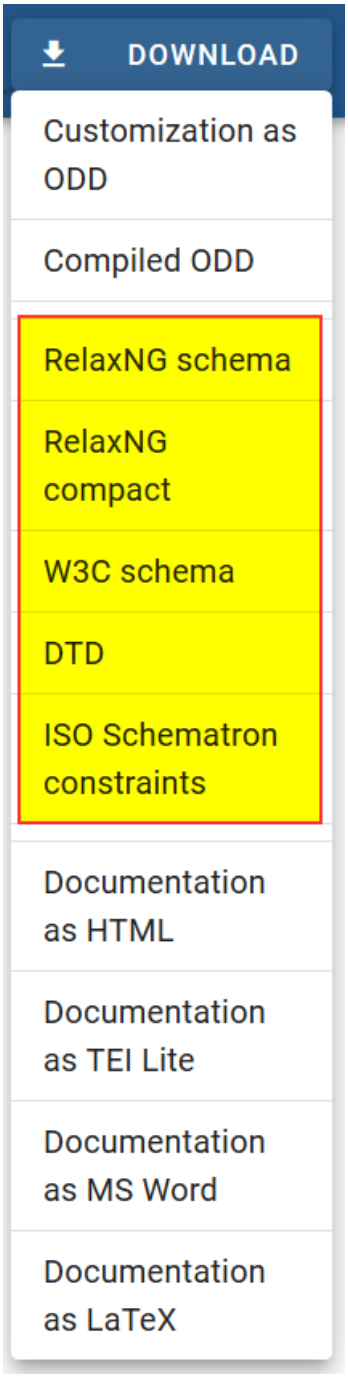


Figure 5. Schema download formats in Roma.

Roma can generate schemas in following schema languages:

- RelaxNG schema: a schema in the more verbose XML syntax of the RelaxNG language

- RelaxNG compact: a schema in the more concise compact syntax of the RelaxNG language
- W3C schema: a schema in the W3C XSD schema language
- DTD: a schema in the DTD language
- ISO Schematron constraints: a stand-alone file containing all ISO Schematron constraints that are present in the ODD file.

Although the schemas generated in these different schema languages are roughly equivalent, some schema languages offer more expressive power than others. The schema languages that offer most complete coverage of TEI features are RelaxNG (XML syntax) and W3C schema.

3

When choosing one of the schema flavours, your browser will download a file named **`TBEcustom`** (as we had specified earlier for our ODD in the "Settings" Roma tab). The file's extension depends on the schema format chosen: `.rng` (Relax NG XML), `.rnc` (Relax NG compact), `.xsd` (W3C schema), `.dtd` (DTD), or `.isosch` (ISO Schematron). You can store this file and use it to validate your TEI documents.

4.1.2 Generating Documentation for a Customisation

Select the "Download" button at the top right in the Roma screen, and choose the documentation format of your choice (the various documentation formats are highlighted in yellow):

.....

3 We'll not go into details about the differences between these XML schema languages here. For more information, Wikipedia is your friend: https://en.wikipedia.org/wiki/XML_schema.

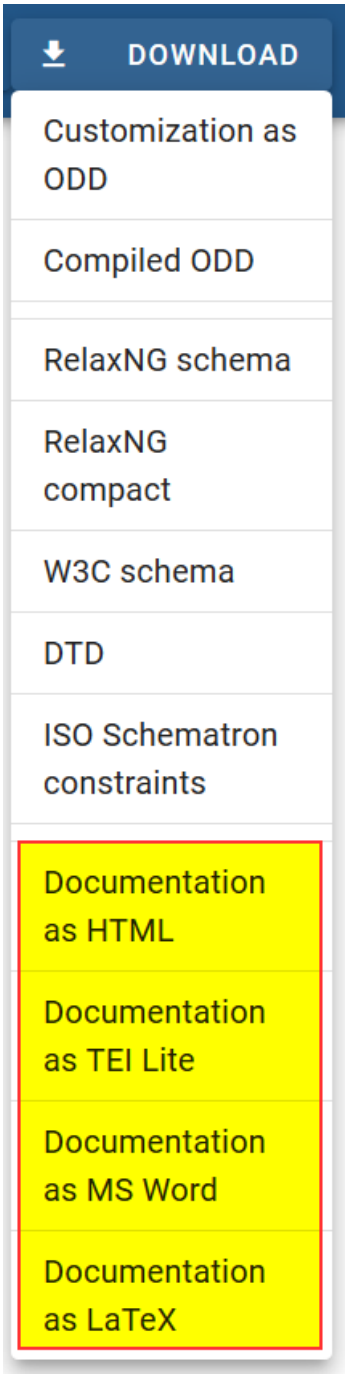


Figure 6. Documentation download formats in Roma.

Roma can generate documentation in following formats:

- HTML: a web page containing the documentation of the TEI customisation

- TEI Lite: a TEI Lite file containing the documentation of the TEI customisation
- MS Word: a Word file containing the documentation of the TEI customisation
- LaTeX: a LaTeX file containing the documentation of the TEI customisation

Make sure you save the file, and see how this produces a file named **`TBEcustom`**, either in HTML, TEI XML, DOCX, or LaTeX format. This documentation can serve as your project-specific TEI Guidelines, containing formal references for all elements in the schema, as well as any prose documentation present in the ODD file.

4.1.3 Generating an ODD File

We saved the best for last: without doubt, saving your customisation as an ODD file is *the* most important step of customising TEI with Roma. It will allow you (and others) to upload this customisation again for reuse, fine tune it further, and / or generate both schemas and documentation from this single source file again. Notice that, although all changes you make in the Roma web interface are being recorded automatically, you still have to save your customisation as an ODD file. In order to do so, all you have to do is hit the "Download" button at the top right in the Roma screen, and choose one of the ODD formats (the ODD formats are highlighted in yellow):

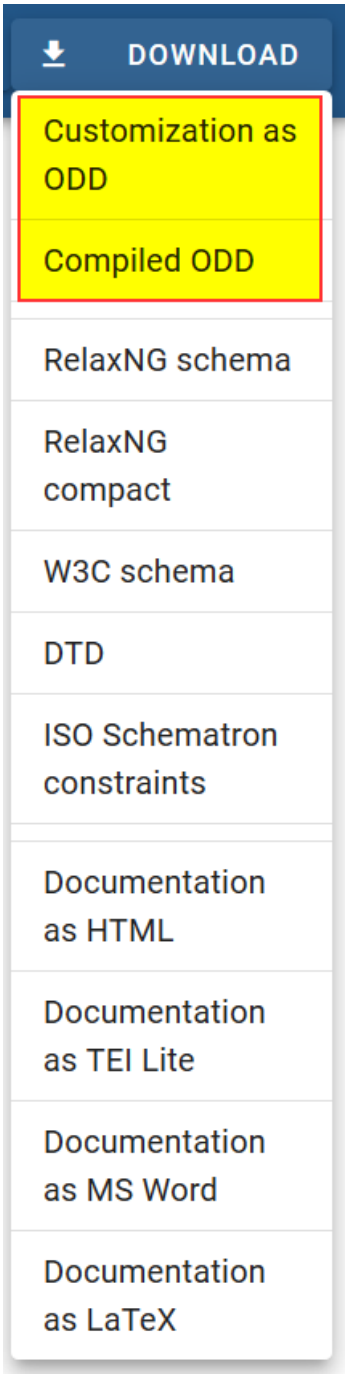


Figure 7. ODD download formats in Roma.

Roma offers following ODD formats:

- Customization as ODD: an ODD file, referencing the various TEI components

- Compiled ODD: an expanded ODD file, containing full documentation and with all references to the TEI components resolved

You'll probably need the first option: "Customization as ODD." This will create a file called **`TBEcustom.odd`**, which is a TEI XML file you can open in any plain text or XML editor. Notice, once again, how the file name corresponds to the one specified for our customisation in the "Settings" Roma tab.

SUMMARY

Roma provides a visual interface to create TEI customisations, either from any of the customisation templates provided by the TEI Consortium, or from a previously saved ODD file on your file system. Customisations can be edited in Roma and exported as an ODD (One Document Does it all) file, from which both a TEI schema and accompanying documentation can be derived in a number of output formats. The ODD file is the core of your TEI customisation.

4.2 What Does a Minimal TEI Customisation Tell Us?

Before proceeding, let's have a closer look at what we've done. As mentioned before, the **`TBEcustom.odd`** file we had stored earlier, is just a TEI XML file. Let's have a look what's inside!

```
<TEI xmlns="http://www.tei-c.org/ns/1.0" xmlns:rng="http://relaxng.org/ns/
structure/1.0" xml:lang="en">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>A TBE Customisation</title>
        <author>The TBE crew</author>
      </titleStmt>
      <publicationStmt>
        <publisher>TEI Consortium</publisher>
        <availability status="free">
          <licence target="http://creativecommons.org/licenses/by-sa/3.0/">
           Distributed under a Creative Commons Attribution-ShareAlike 3.0 Unported
           License </licence>
          <!-- ... -->
```

```
      </availability>
    </publicationStmt>
    <sourceDesc>
      <p>Created from scratch by James Cummings, but looking at previous tei_minimal
       and tei_bare exemplars by SPQR and LR.</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
<text>
  <body>
    <head>A Minimal TEI Customization</head>
    <p>This TEI ODD defines a TEI customization that is as minimal as possible
     and the schema generated from it will validate a document that is
     minimally valid against the TEI scheme. It includes only the ten required
     elements: <list rend="numbered">
      <item><gi>teiHeader</gi> from the header module to store required
       metadata</item>
      <item><gi>fileDesc</gi> from the header module to record information about this
       file</item>
      <item><gi>titleStmt</gi> from the header module to record information about the
       title</item>
      <item><gi>publicationStmt</gi> from the header module to detail how it is
       published</item>
      <item><gi>sourceDesc</gi> from the header module to record where it is
       from</item>
      <item><gi>p</gi> from the core module for use in the header and the body</item>
      <item><gi>title</gi> from the core module for use in the titleStmt</item>
      <item><gi>TEI</gi> from the textstructure module because what is a TEI file
       without that?</item>
      <item><gi>text</gi> from the textstructure module to hold some text</item>
      <item><gi>body</gi> from the textstructure module as a place to put that
       text</item>
    </list></p>
    <schemaSpec ident="TBEcustom" start="T
    EI" prefix="tei_" targetLang="en" docLang="en">
      <!-- required minimal header elements -->
      <moduleRef key="header" include="teiHeader fileDesc titleStmt publicationStmt
       sourceDesc"/>
```

```
        <!-- required core elements (p and title for use in titleStmt) -->
        <moduleRef key="core" include="p title"/>
        <!-- required textstructure elements (TEI, text, and body) -->
        <moduleRef key="textstructure" include="TEI text body"/>
        <!-- required module tei instantiates lots of classes used for further expansion
         of this odd -->
        <moduleRef key="tei"/>
      </schemaSpec>
    </body>
  </text>
</TEI>
```

Example 1. A minimal TEI customisation ODD (download).

One immediate observation is that an ODD file is indeed just a regular TEI document, with a <TEI> root element, containing a <teiHeader> element and a <text> element. Remember the metadata you entered in the "Settings" tab in Roma, and see how it is reflected at the proper places inside the <teiHeader>. However, the most interesting bits are inside 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 identification code 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 added to the names 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 valid TEI documents.

5

.....

5 Since an ODD file is just a regular TEI file, the formal schema specification in <schemaSpec> can be completed with a detailed prose documentation (and in fact, the concept of an ODD file is *intended* to contain both formal specifications and prose documentation), using the regular TEI elements in the <body> part. Our current example only contains very minimalistic prose documentation; for an excellent example, see the documentation in the TEI Lite ODD file, and compare it with the different documentation files offered at https://tei-c.org/guidelines/customization/Lite/.

The <schemaSpec> element is the core of any ODD file, specifying the formal definition of a TEI schema. A TEI customisation can be constructed by referring to definitions of existing TEI objects (element and attribute classes, datatypes, and macros), or—as will be covered later in this tutorial—by declaring new objects as well. In our example so far, the schema specification only contains references to four of the predefined TEI modules. Each module is being referred to in a separate <moduleRef> element, whose @key attribute is pointing to the formal identifier of that module in the TEI source.

When a module is selected with <moduleRef> in an ODD file, by default all elements and attributes of that module are incorporated in the customisation. In our example, this is the case with the **tei** module. As we have seen, this is a special module, which doesn't define any elements and attributes itself, but instead a lot of classes, datatypes, and macros that are being used in all other TEI modules. Three of those other modules are being referenced in our current ODD file. Yet, only a very small number of elements they define are being included in our customisation, because the <moduleRef> elements have an @include attribute, where only those elements that should be included are enumerated as a white-space separated list. If you count them, our customisation only selects the 10 elements that are required for minimally valid TEI documents.

Notice, how the @include mechanism has a counterpart: it is equally possible to specify which elements *not* to include in a module by enumerating them in an @except attribute. This will only exclude those elements, but will include all others. Both mechanisms offer convenient shortcuts to either include or exclude large numbers of elements. It's important to remember that you can only use either the @include or the @except attribute on a <moduleRef> element, but not both.

SUMMARY

TEI groups its different elements and their attributes in 21 modules. When defining a TEI customisation, these modules can be referred to in an ODD file. 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 a @key attribute. When a TEI module is included in a customisation, by default all of its elements and attributes

will be included. In order to include specific elements only, the names of these elements can be enumerated in an @include attribute. Likewise, if only some of the elements of a module should be excluded, they can be enumerated in an @except attribute on <moduleRef>.

4.3 Inspecting a Customisation in Roma

As we have introduced in section 2, the TEI is a highly sophisticated library, holding model and attribute classes, macros, and datatypes. Customising TEI requires some knowledge of how they are organised. Full documentation can be found in the TEI Guidelines, but fortunately, Roma is conceived as a convenient online store for selecting the exact TEI components you need, and doing so, it shows you around in the internals of TEI as well. Let's start editing our current customisation where we have left it by hitting the "Customize ODD" button at the top right in the Roma screen:

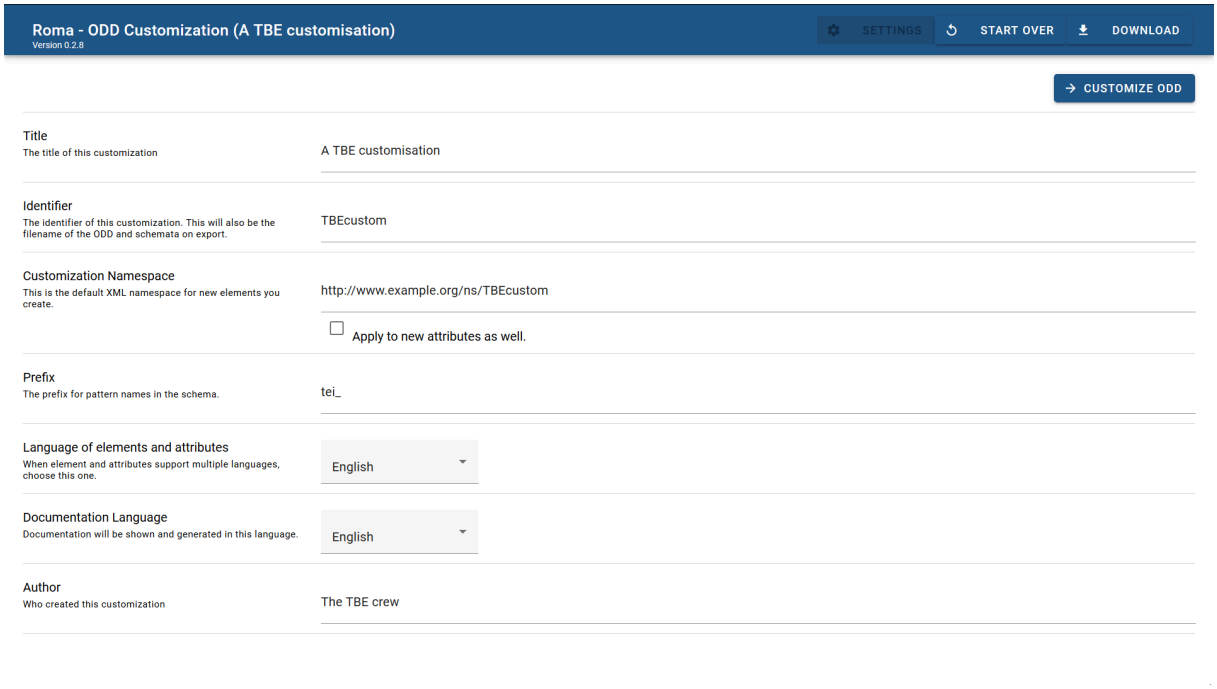


Figure 8. Start customising an ODD file with the "Customize ODD" button.

This will take you to the main Roma dashboard. You can tell you're editing the right customisation from the title at the top of the screen, which reads "Roma - ODD Customization (A TBE customisation)."

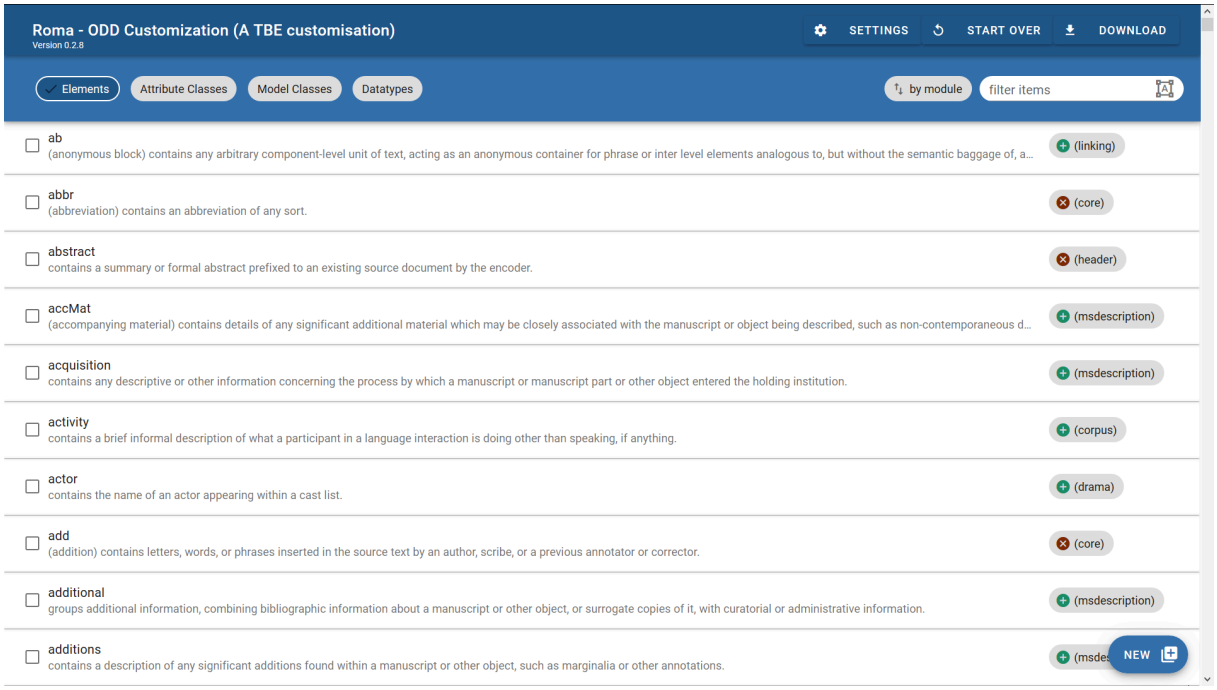


Figure 9. Roma main screen.

Below the title, you see 4 different tabs at the top of the screen:

- "Elements": an overview of all available TEI elements, allowing you to select and customise those you need
- "Attribute Classes": an overview of all available TEI attribute classes, allowing you to select and customise those you need
- "Model Classes": an overview of all available TEI model class definitions, allowing you to select and customise those you need
- "Datatypes": an overview of all available TEI datatype definitions, allowing you to select and customise those you need

Let's check how the <moduleRef> elements we've seen in our current TEI customisation are being reflected in the Roma interface. When you select the "Elements" tab, you'll notice that only a very small number of elements are ticked. Indeed, from the long list of available TEI elements, only 10 have been selected: <body>, <fileDesc>, <p>, <publicationStmt>, <sourceDesc>, <TEI>, <teiHeader>, <text>, <title>, and <titleStmt>. Notice, how the default ordering in Roma is alphabetical; you can also click the "by module" button at the top right of the Roma screen. This will order the elements (or attribute classes, or model classes, or datatypes) alphabetically: first per module, and next by name. This sorting "by module" gives you an idea of what TEI components are

being defined in each of the modules. This is broader than elements: if you look at the other tabs, you'll notice that a lot of attribute classes, model classes, and datatypes are being included automatically when a TEI module is included via <moduleRef> in a TEI customisation.

5. Selecting and Restricting the TEI Model

Back to *Alice*! As we have discussed, our minimal TEI customisation now includes 10 TEI elements, just enough to be able to create TEI-conformant documents. But does this suffice for encoding our *Alice* example? Not really, since we're still lacking a bunch of TEI elements to encode the textual phenomena we identified during our document analysis:

- The title page: <titlePage>, <docAuthor>, <byline>, <docImprint>, <publisher>, <docDate>.
- Document structure: , <lg>, <l>.
- Text elements: <emph>, <q>, <pb>, <figure>, <graphic>, <figDesc>, <fw>.
- Semantic units: <name>.

Back to the drawing board of our TEI customisation, back to Roma!

5.1 Selecting Modules and Elements

In order to encode the textual phenomena we'd identified for our example text, a number of TEI elements have to be included in our TEI customisation. This can be done easily in Roma by ticking the corresponding boxes in the "Elements" tab.

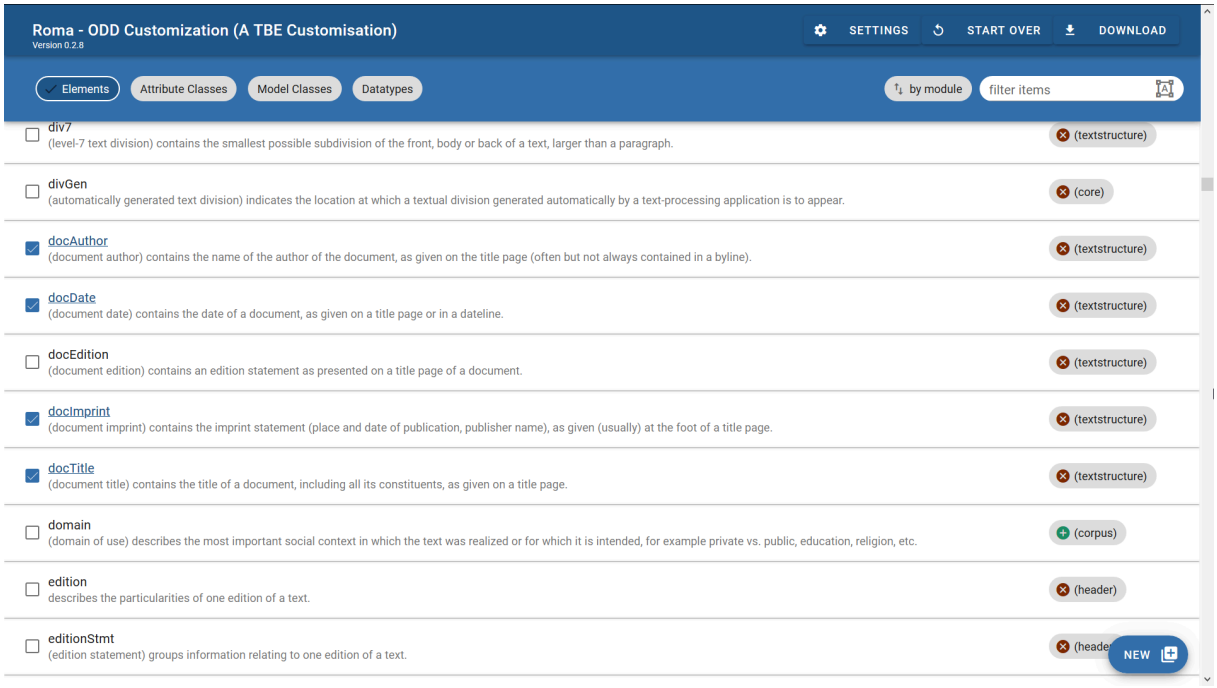


Figure 10. The "Elements" tab in Roma.

Notice how the right-hand column in Roma informs you for any element which module it belongs to. A convenient way to include or exclude all elements from a module at once, is by selecting or de-selecting that module. In order to do so, you can click the module name. For example, let's select the **`figures`** module by clicking

the plus sign next to that module name:

If we order the elements by module (by clicking the "by module" button at the top right of the Roma screen) and scroll down to the **`figures`** module, you'll see that all of its elements now have been included in your customisation. This includes some elements we didn't identify in the document analysis for our document; if we are absolutely certain we only have to encode images, but no tables, formulas, or music notations, these elements can quickly be deselected by unticking the element name in the left-hand column:

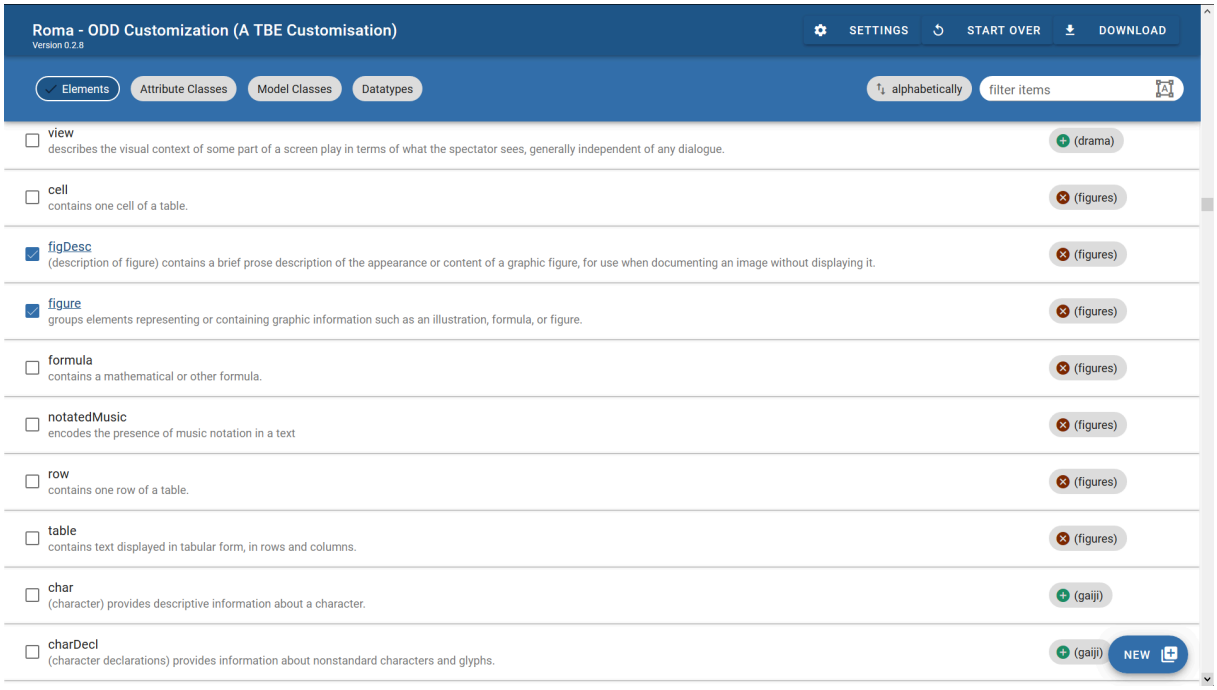


Figure 11. De/selecting elements in Roma.

Did you notice how the <graphic> element is part of the **`core`** module, and not of **`figures`**? If you want to include the images themselves in your transcription of *Alice*, you'll have to include the <graphic> element as well. Hint: you can quickly look up an element by entering its name in the search box at the top right of the Roma screen:

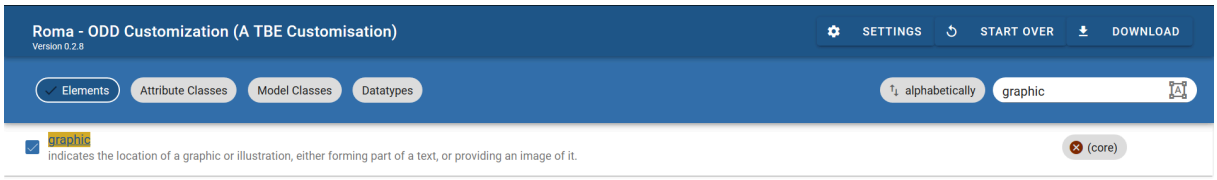


Figure 12. Filtering element names in Roma.

That's all there is to including and excluding existing TEI elements with Roma! You can also remove an entire module at once, with the caveat that this will remove all its members (and possible modifications you've made to them) from your customisation. Since you might run the risk of losing work here, Roma issues a warning in order to prevent unwanted mistakes:

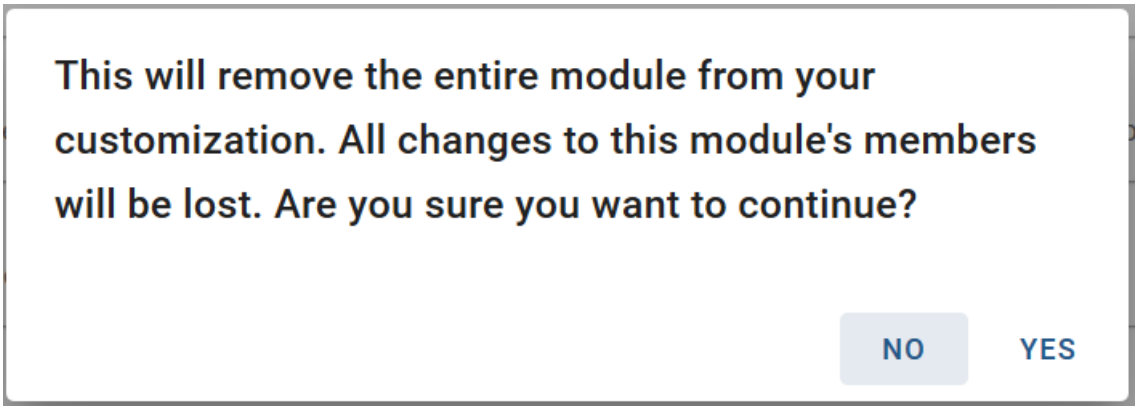


Figure 13. Warning when removing an entire module in Roma.

Now, save your customisation as an ODD file (click the "Download" > "Customization as ODD" button at the top right of the Roma screen). Its <schemaSpec> element will be updated to:

```
<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 graphic emph lg l pb pubPlace publisher q
   quote name"/>
  <moduleRef key="textstructure" include="TEI text body titlePage docTitle docImprint
   docDate docAuthor byline"/>
  <moduleRef key="tei"/>
</schemaSpec>
```

Example 2. A minimal TEI customisation with more TEI elements added (download).

7

.....

7 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.

When we generate a TEI schema from this customisation (via the "Download" button at the top right of the Roma screen), this allows us to validate a transcription of a typical page of the document (the third image from the *Alice* scans above) as follows:

```
<!-- ... -->
<div xmlns="http://www.tei-c.org/ns/1.0" type="chapter">
  <!-- ... -->
  <pb n="157"/>
  <figure>
    <graphic url="images/lobster.jpg"/>
    <figDesc>The lobster sugaring its hair.</figDesc>
  </figure>
  <p><q who="#alice">"How the creatures order one about, and make one repeat
   lessons!"</q> thought <name type="person">Alice</name>, <q who="#alice">"I
   might just as well be at school at once."</q> However, she
   got up, and began to repeat it, but her head was so full of
   the <title type="song"><name type="animal">Lobster</name>-Quadrille</title>, that
   she hardly knew what she was saying, and the words came very queer indeed:—</p>
  <q rend="blockquote" who="#alice">
    <lg>
      <l>"'Tis the voice of the <name type="animal">lobster</name>; I heard him
       declare,</l>
      <l>
        <q who="#lobster">'You have baked me too brown, I must sugar my hair.'</q>
      </l>
      <l>As a duck with its eyelids, so he with his nose</l>
      <l>Trims his belt and his buttons, and turns out his toes."</l>
    </lg>
  </q>
  <p><q who="#gryphon">"That's different from what <emph>I</emph> used to say when I
   was a child,"</q> said the <name type="animal">Gryphon</name>.</p>
  <pb n="158"/>
  <p><q who="#mockTurtle">"Well, I never heard it before,"</q> said
   the <name type="animal">Mock Turtle</name>; <q who="#mockTurtle">"but it sounds
   uncommon nonsense."</q></p>
```

```
  <p><name type="person">Alice</name> said nothing; she had sat down with her face in
   her hands, wondering if anything would <emph>ever</emph> happen in a natural way
   again.</p>
  <p><q who="#mockTurtle">"I should like to have it explained,"</q> said
   the <name type="animal">Mock Turtle</name>.</p>
  <p><q who="#gryphon">"She can't explain it,"</q> said
   the <name type="animal">Gryphon</name> hastily. <q who="#gryphon">"Go on with the
   next verse."</q></p>
  <p><q who="#mockTurtle">"But about his toes?"</q> the <name type="animal">Mock
   Turtle</name> persisted. <q who="#mockTurtle">"How <emph>could</emph> he turn them
   out with his nose, you know?"</q></p>
  <p><q who="#aliceI">"It's the first position in
   dancing."</q> <name type="person">Alice</name> said; but she was dreadfully puzzled
   by the whole thing, and longed to change the subject.</p>
  <p><q who="#gryphon">"Go on with the next verse,"</q>
   the <name type="animal">Gryphon</name> repeated impatiently: <q who="#gryphon">"it
   begins <quote>'I passed by his garden.'</quote>"</q></p>
  <p><name type="person">Alice</name> did not dare to disobey, though she felt sure it
   would all come wrong, and she went on in a trembling voice:—</p>
  <pb n="159"/>
  <!-- ... -->
</div>
<!-- ... -->
```

So far for selecting modules and elements. The obvious counterpart, adding *new* elements, will be dealt with later in this tutorial (see section 6). 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. If an @include attribute is present, only those elements that are being enumerated will be included in the customisation, and all others won't. If you want to exclude only a couple of elements from a module, but include all others, this can conveniently be done by enumerating the ones you don't want in an @except attribute. You can't use both at the same time.

5.2 Changing Attributes

5.2.1 Changing Individual Attributes

As shown in the previous encoding snippet, the <name> element definition (from the **core** module) comes with a @type attribute. By providing a keyword for this attribute, we can specify what type of name is being identified with the <name> element. 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, specified in the teidata.enumerated datatype (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 improve consistency in our *Alice* encoding project, we would like to trim down these possibilities for the @type attribute of <name>. We're only interested in the 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 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 <name>.

In order to edit its attributes, just click the "name" element in the list (make sure it is selected, before you can click it):

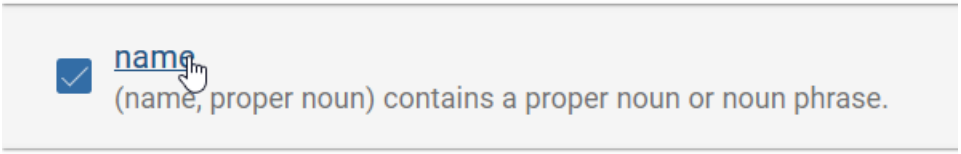


Figure 14. Navigating to an element definition by clicking its name in Roma.

Clicking an element in Roma opens an "edit" screen, where some aspects can be changed:

- "Documentation" here you can change the description of an element
- "Attributes" here you can change the attributes of an element
- "Class Membership & Content Model" here you can change the classes an element belongs to, and the elements or element classes it can contain

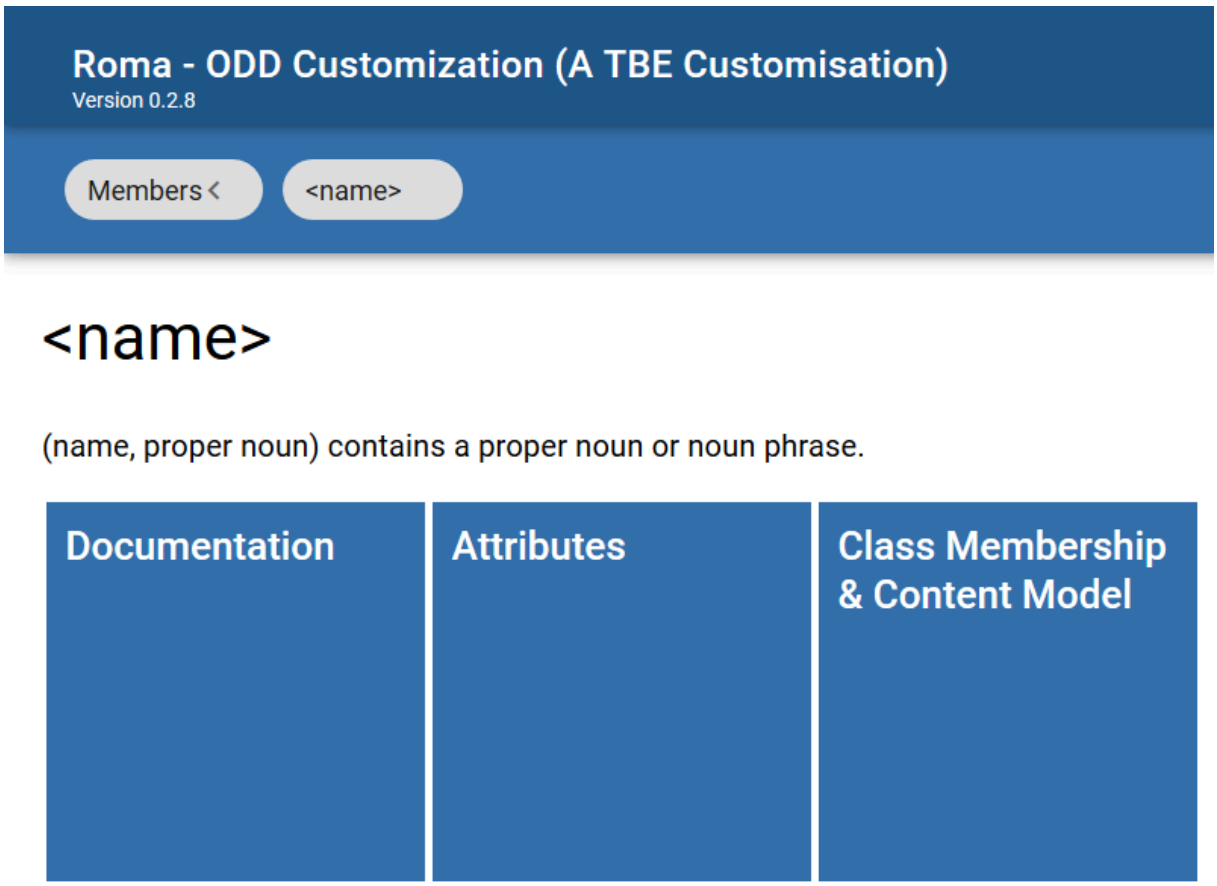


Figure 15. Element editing screen in Roma.

Since we want to restrict the values allowed for the @type attribute of <name>, hit the “Attributes” button. This produces an overview of all attributes defined for the <name> element, organised per attribute class from which they are inherited:

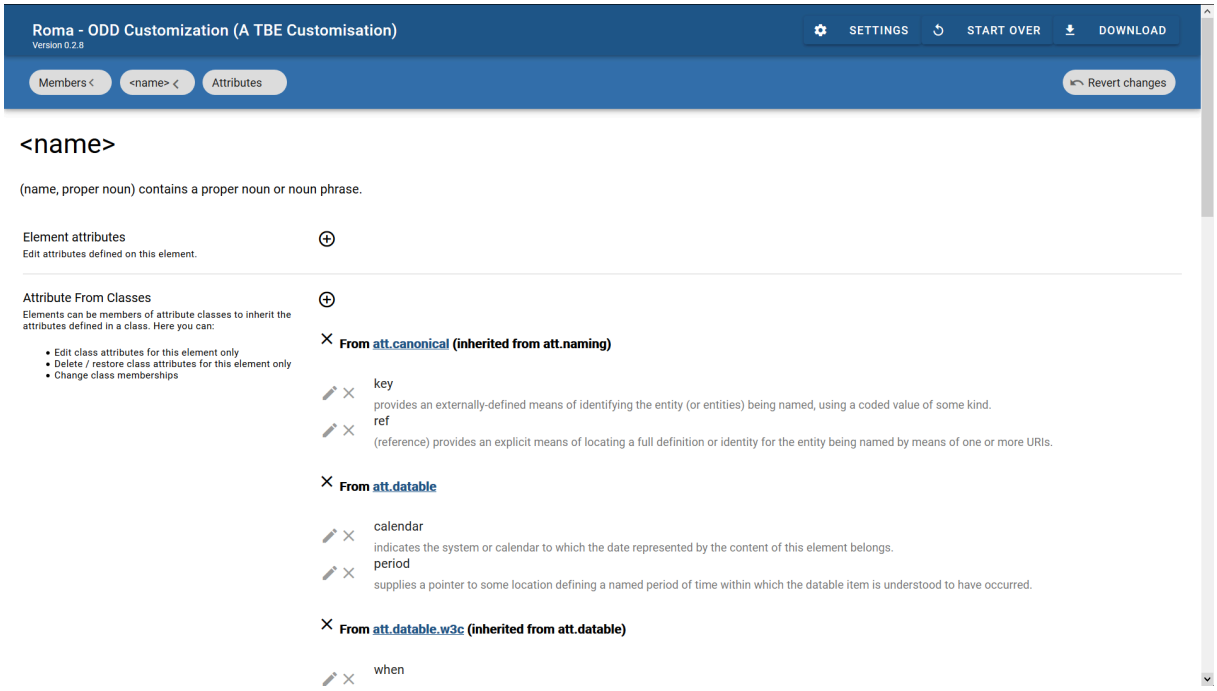


Figure 16. Display of an element's attributes in Roma.

Scroll down to the bottom, where you will find the `@type` attribute listed (in the att.typed attribute class). Left of each attribute name, you'll find a pencil icon for editing the attribute, and a cross icon for removing the attribute. Since we want to restrict the values for the `@type` attribute, just click the pencil icon next to it. This will show the current definition of that attribute. There you can determine whether the attribute should be mandatory or optional, what the datatype of its value(s) should be, and what kind of values it allows. The "Values" section is where we can restrict the possible values for the `@type` attribute for names to a closed list. In order to do so:

1. Click the drop-down box in the "Values" field, and change it to "Closed."
2. For each value we want to allow for this attribute (`"place"`, `"person"`, and `"animal"`), enter the value in the text box below, and click the "+" sign next to it.
3. For each new value, a description can be provided by clicking the "+" sign next to "Description."

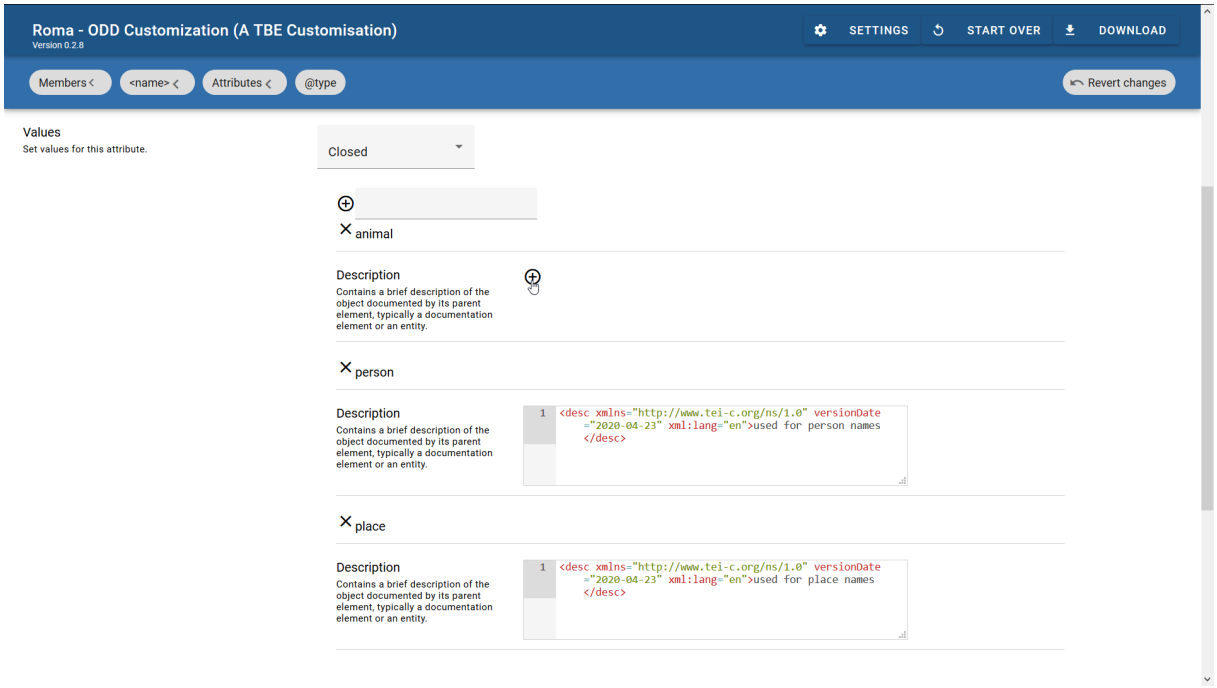


Figure 17. Changing the attributes for an element in Roma.

The changes are being recorded immediately in Roma. Clicking the "Attributes" button in the top menu of the Roma screen brings us back to the attributes list for the <name> element. Now, let's remove some attributes we don't need in our encoding project. With Roma, it's possible to remove entire attribute classes at once by clicking the "X" sign next to the attribute class names in the list. Let's do so for att.datable, att.editLike, att.global.responsibility, att.global.source, att.naming, and att.personal. Also, the @subtype and @nymRef attributes can go, just click the "X" sign next to the attribute name.

If we save the customisation at this stage (by clicking the "Download" > "Customization as ODD" button), the ODD file gets updated to:

```
<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"/>
  <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>
</schemaSpec>
```

Example 3. Attribute classes removed and attribute values changed for the `<name>` element (download).

Notice how a new element is introduced: <elementSpec>. In a schema specification, this is where elements are being defined. It has a mandatory @ident attribute, which names the element. In this case, the "name" value tells us that this element specification defines a <name> element. Although this could be specified further by adding a @module attribute with value "core", this is not mandatory, since element names are unique across all TEI modules.

Defining a TEI element that already exists might seem a bit weird at first sight. Notice, however, the @change attribute with the value "change", which is expressing that this existing TEI element is being changed in this customisation. This is an important mechanism: via the @mode attribute, ODD lets you specify a number of actions on the declaration of elements and attributes (and many of their components):

"add"

a new declaration is added to the current definitions

"delete"

an existing declaration is removed

"change"

an existing declaration is changed partly: only the customised parts are changed; all other parts are copied from their TEI definition

"replace"

an existing declaration is changed completely: only the customised parts are copied; all existing TEI definitions are ignored

In our ODD file, the <elementSpec ident="name" mode="change"> element is telling that the existing declaration of <name> (in the **core** module) should be copied, except for the components that are overridden in this customisation. In this case, there are both class-related changes, that are grouped in a <classes> element, and attribute-related changes, that are grouped in an <attList> element. The <classes> element groups class membership declarations in <memberOf> elements. In this case, three class memberships are deleted: each <memberOf> element refers to the class with the value of a @key attribute, and next states that this membership

should be removed, by the `"delete"` value for the `@mode` attribute. It's interesting to notice that, even though we removed the att.canonical class in Roma, this is not reflected in a separate `<memberOf key="att.canonical" mode="delete/>` element in the ODD file. This is not strictly needed, since att.canonical is a member of att.personal, so removal of the latter superclass automatically implies that the subclass(es) are removed as well.

Apart from class deletions, the <elementSpec> element with the definition of the customised <name> element groups attribute declaration in an <attList> element. Each attribute is declared in an <attDef> element. Here, too, the attribute is identified with an `@ident` attribute, and the customisation action is given in `@mode`. Four attributes are deleted: `@subtype`, `@nymRef`, `@cert` and `@resp` (from the att.global.responsibility attribute class), and `@source` (from the att.global.source attribute class). It's interesting to notice that Roma doesn't just remove these classes via a <memberOf> element, as with the other attribute classes we had deleted, but instead removes their attributes individually. While both ways of removing attributes in ODD are equivalent, Roma probably opts to remove "standalone" classes as such, while members of "inherited" classes are removed individually.

The definition for the `@type` attribute, has the value `"change"` for its `@mode` attribute, meaning that its contents should override the existing definitions. Since we restricted the list of possible values to a closed vocabulary, a <valList> re-defines the value list of this `@type` attribute. Its `@type` attribute with value `"closed"`, indicates that the attribute value list is a closed list. In order to indicate that only the parts defined in our customisation should be changed, a `@mode="change"` attribute is given on <valList>. The different values of this closed list are defined in a series of <valItem> elements, with the value of the attribute given in an <ident> attribute (in this case: `"place"`, `"person"`, and `"animal"`, respectively). Inside <valItem>, the description of the attribute value is given in a <desc> element. Notice that, since these attribute values are new, the `@mode` attribute on each <valItem>

element has the value `"add"`, to tell the ODD processor that they should be added to the existing definition of the @type attribute. All other parts of the existing TEI definition of the @type attribute are being copied when transforming this ODD file to a schema or documentation.[9]

SUMMARY

Elements are defined in an ODD file with the <elementSpec> element. The element must be identified in an @ident attribute, and the processing mode should be stated in a @mode attribute. Inside an element declaration, entire attribute classes can be removed by removing their membership of an attribute class. This is done inside a <classes> section, where each class membership is declared in a <memberOf> element. The value `"delete"` for the @mode attribute indicates that a class membership should be removed. Declarations for individual attributes are grouped in an <attList> element inside <elementSpec>. 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. Value lists for an attribute can be defined in a <valList> element, whose @type attribute indicates 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 the value given in an @ident attribute. The attribute value can be described in a <desc> element.

5.2.2 Changing Attribute Classes

As mentioned before, TEI modules group attribute definitions into *attribute 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 members of the att.global attribute class, which defines the global attributes @xml:id, @n, @xml:lang, @rend, @rendition, and @xml:base.

.....

9 Notice, however, that a full attribute definition consists of more fields, like a description, declarations of datatype and occurrence indicators. These are discussed later in this module (section 6.2).

As it happens, the @nymRef attribute we have 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. Actually, we've been there before:

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 <name>.

Remember how the attributes for <name> were grouped per attribute class? Those attribute class names are clickable in Roma. One way of accessing the att.naming attribute class definition in Roma, is by clicking the link with the label att.naming in the attributes list:

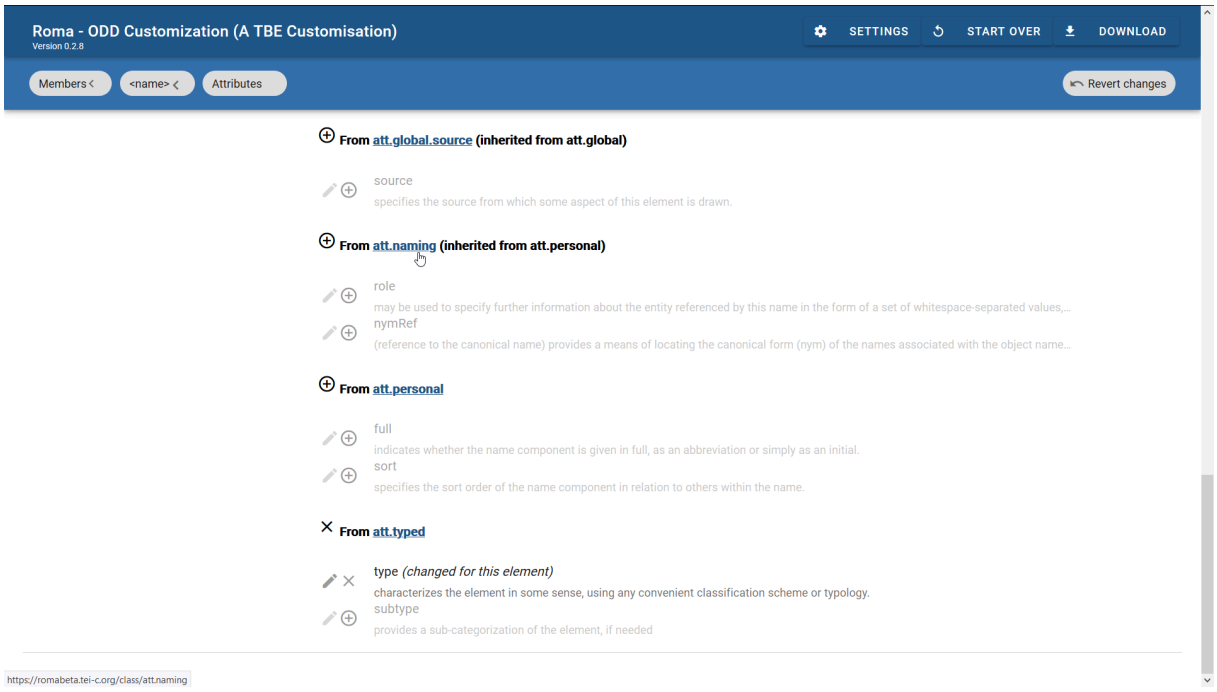


Figure 18. Accessing the definition of an attribute class via the attributes list of an element in Roma.

Another way of navigating to the attribute classes, is by selecting the "Attribute Classes" tab at the top left of the Roma screen. As with the "Elements" tab, you'll get a list of all attribute classes, with an indication of those that have been included in the current customisation, and an indication of the TEI module that defines them.

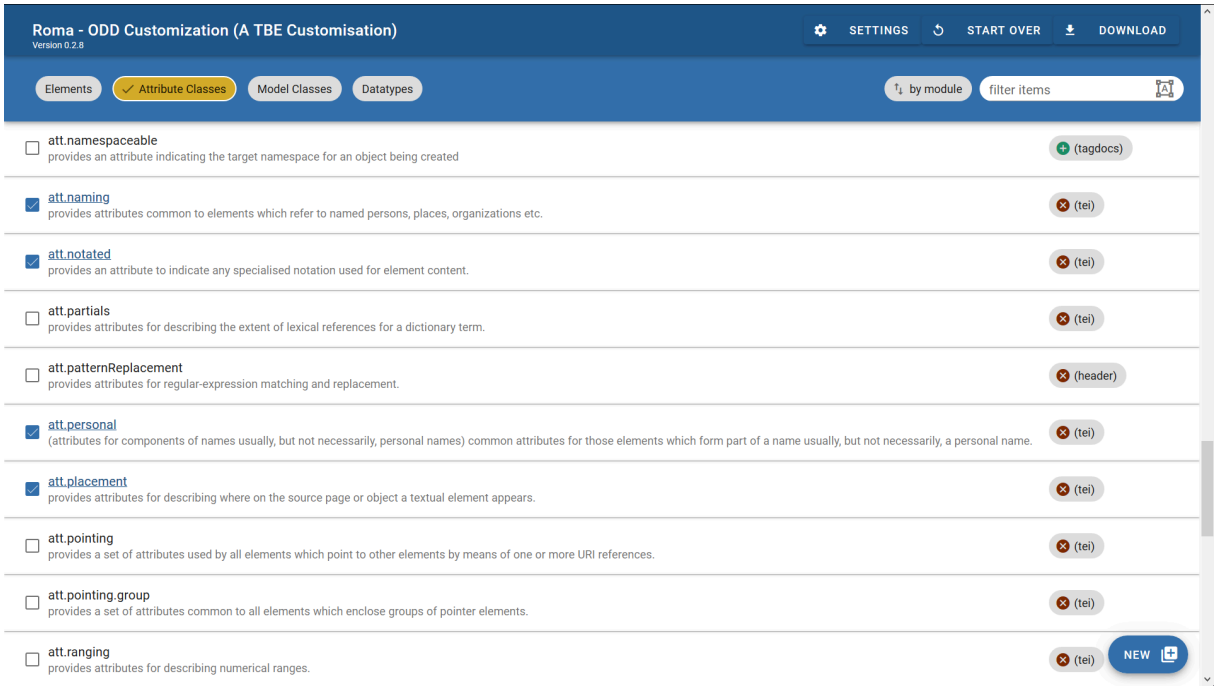


Figure 19. The “Attribute Classes” tab in Roma.

Clicking the link with the label att.naming attribute class here, will take you to the definition of this attribute class:

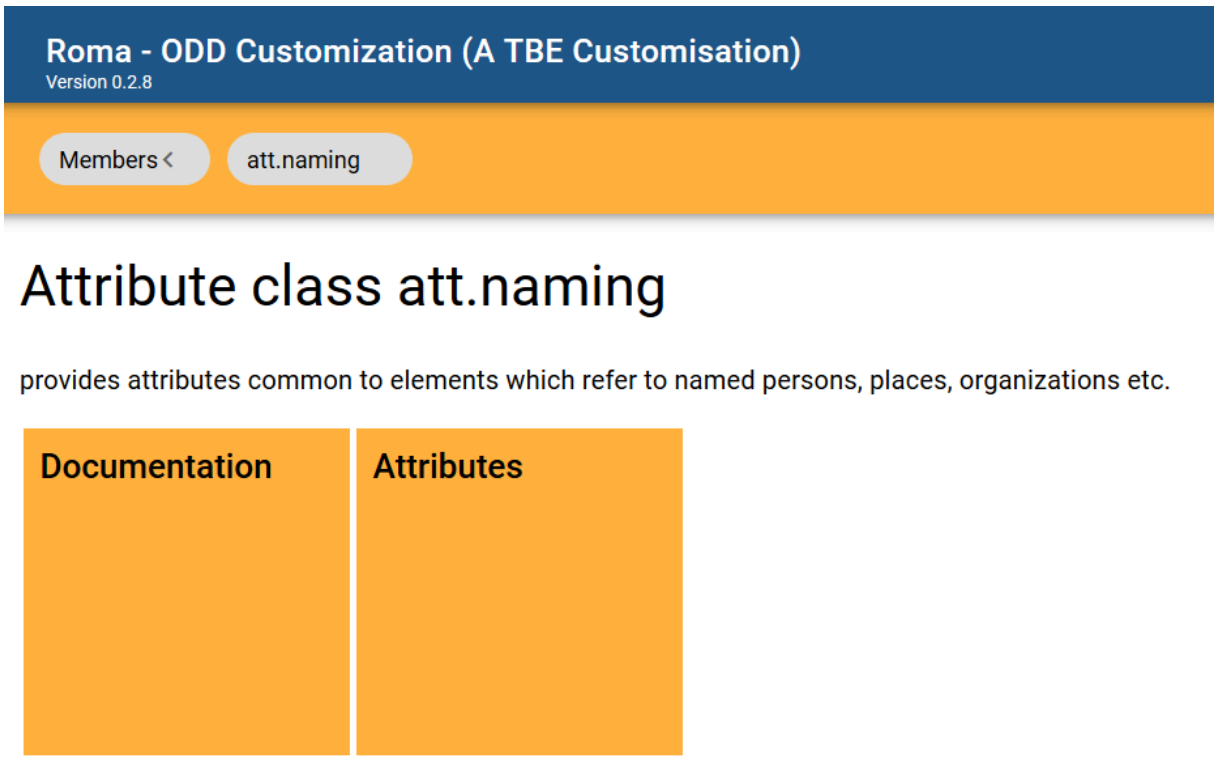


Figure 20. Attribute class editing screen in Roma.

The “Attributes” button will take us to the definition of the attributes inside this class:

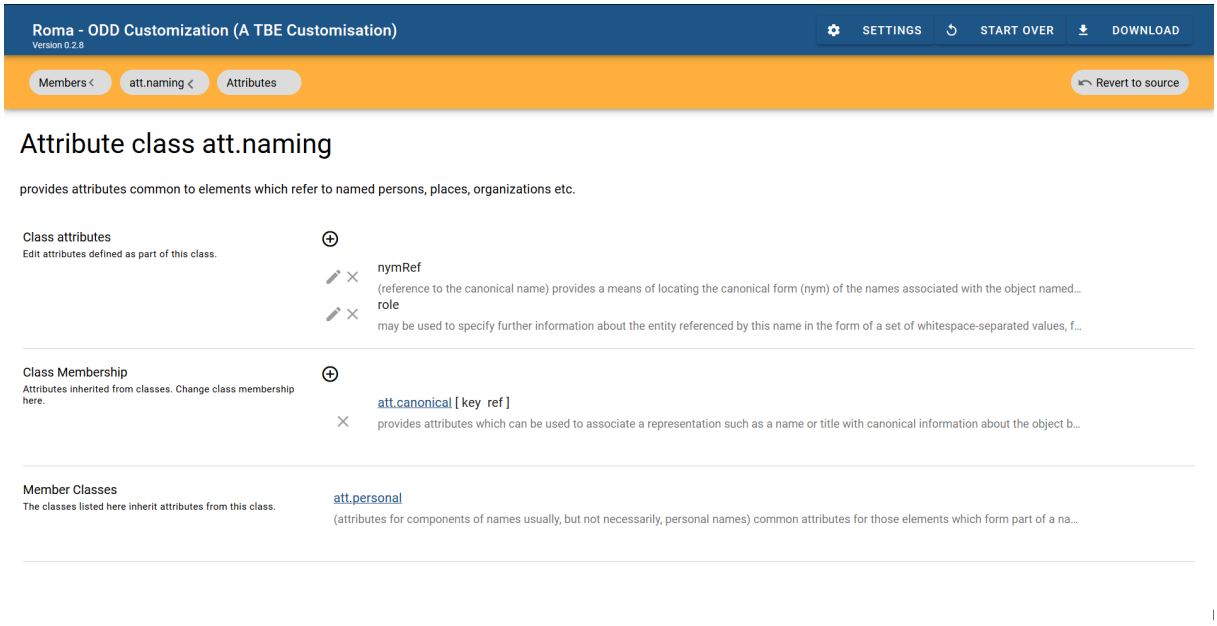


Figure 21. Display of the attributes in an attribute class in Roma.

The attribute class contains a number of sections: “Class Attributes” define the attributes of this class, in this case @nymRef and @role. In “Class Membership,” a superclass can be given; here, the att.naming attribute class is declared as a member of the att.canonical attribute class. This means that elements declaring themselves as member of (the subclass) att.naming, will also inherit the attributes defined in (the superclass) att.canonical, namely @key and @ref. Finally, the section “Member Classes” names attribute classes that are defined as subclass of att.naming, in this case att.personal.

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 elements that are member of the att.naming attribute class. This can be done simply by clicking the “X” sign next to the @nymRef attribute name.

If we save the customisation again (by clicking the “Download” > “Customization as ODD” button at the top right of the Roma screen), this produces following ODD file:

```
<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"/>
<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="cert" mode="delete"/>
    <attDef ident="resp" mode="delete"/>
    <attDef ident="source" mode="delete"/>
  </attList>
</elementSpec>
<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>
</schemaSpec>
```

Example 4. Changed attribute class (download).

Now, the <schemaSpec> element in our customisation ODD file contains another element, besides <elementSpec>: <classSpec>. This is where classes are being declared. As with element and attribute definitions, the name of the class is given in an @ident attribute, and a @mode attribute indicates the processing mode. There is an extra, mandatory, attribute, though: @type, which indicates what kind of class is being defined. Possible values are:

"atts"

for the definition of an attribute class, which defines a number of common attributes

"model"

for the definition of a model class, which groups elements that can occur in the same context in a TEI document

The content of this class specification looks familiar: an <attList> element groups all attribute declarations defined by the (attribute) class. Since the att.naming class is being processed in "change" mode, it suffices to just redefine the @nymRef attribute of this class: all other attributes of the class are being copied unmodified in our customisation. Hence, all it takes to remove @nymRef from the att.naming class, is an <attDef> element with @ident="nymRef", and @mode="delete".

Actually, the deletion of the @nymRef attribute from the att.naming attribute class obsoletes our earlier 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 one of the schema output formats in the "Download" button at the top right of the Roma screen): this will only validate documents whose name-like elements don't have a @nymRef attribute.

11

SUMMARY

Attributes that are defined in an attribute class can be changed globally by changing the class specification in a <classSpec> element. This element must identify the name of the class in an @ident 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.

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.

.....

11 Be careful, though, when changing class definitions, as this can have wide ranging (and sometimes unforeseen) effects! Always make sure to study the relevant parts of the TEI Guidelines. And make sure you can trace back when anything unexpected happens: save early, save often!

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.

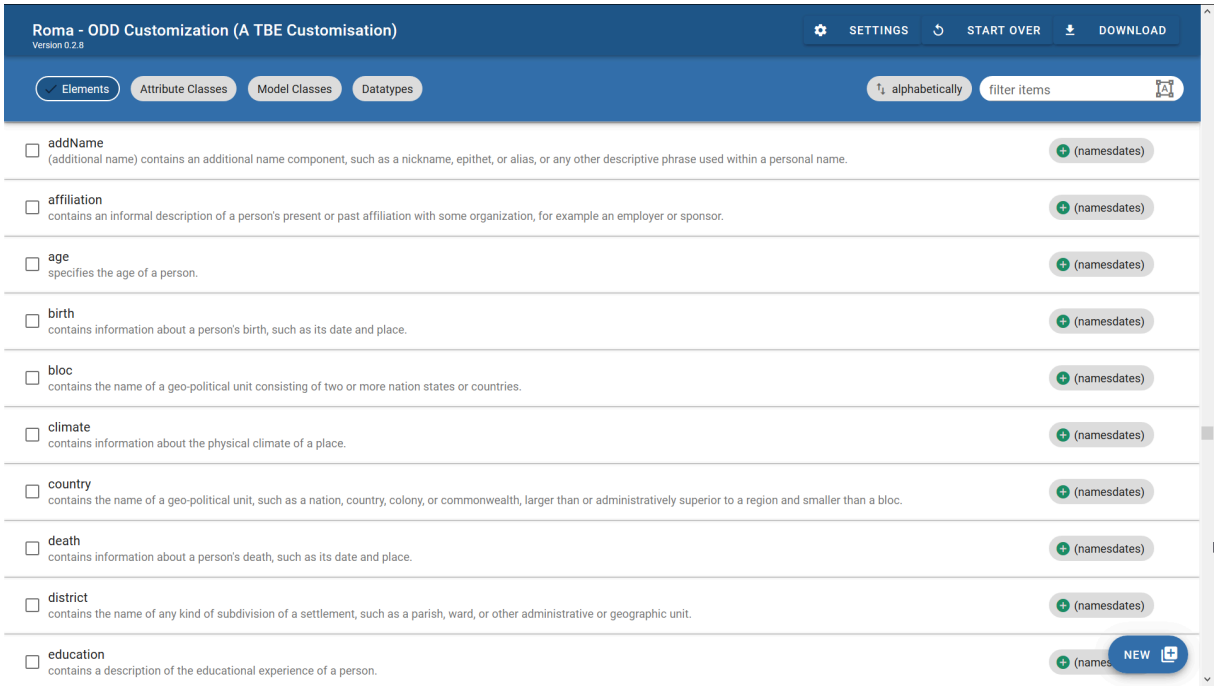


Figure 22. Grouping all elements of a TEI module in Roma.

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`**

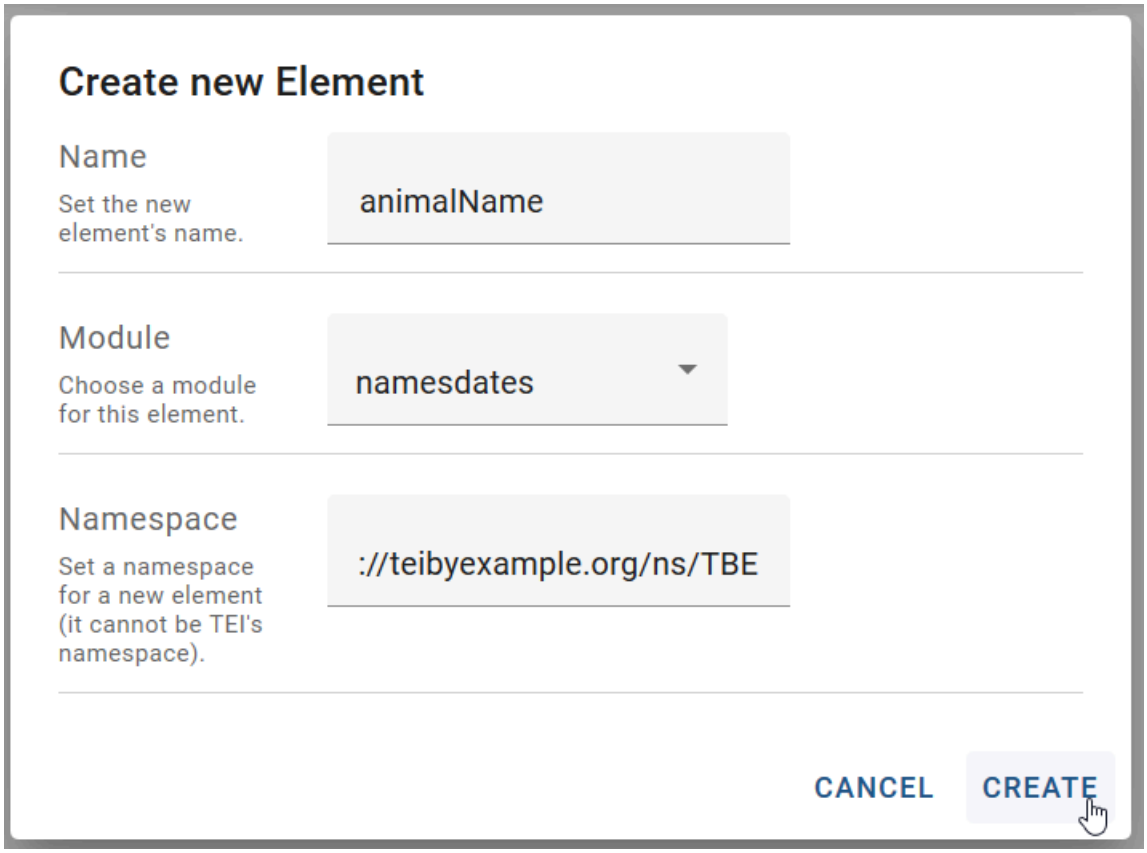


Figure 24. "Create new element" form in Roma.

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

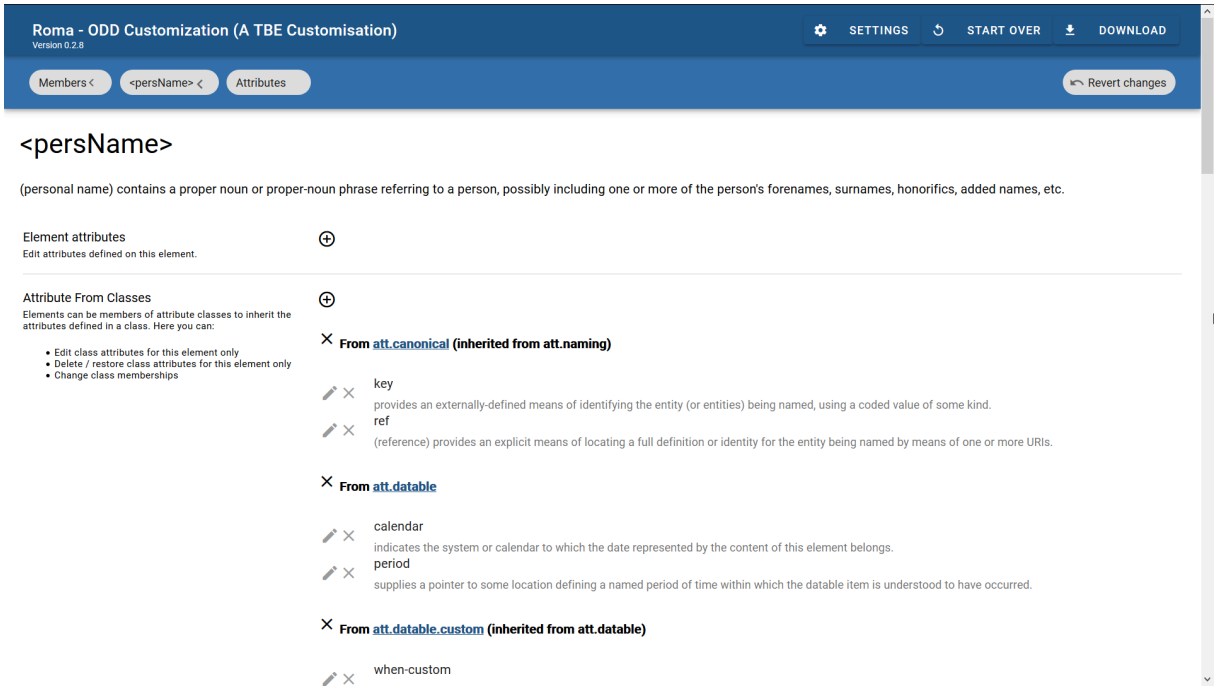


Figure 26. Display of the attributes for the `<persName>` element in Roma.

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.

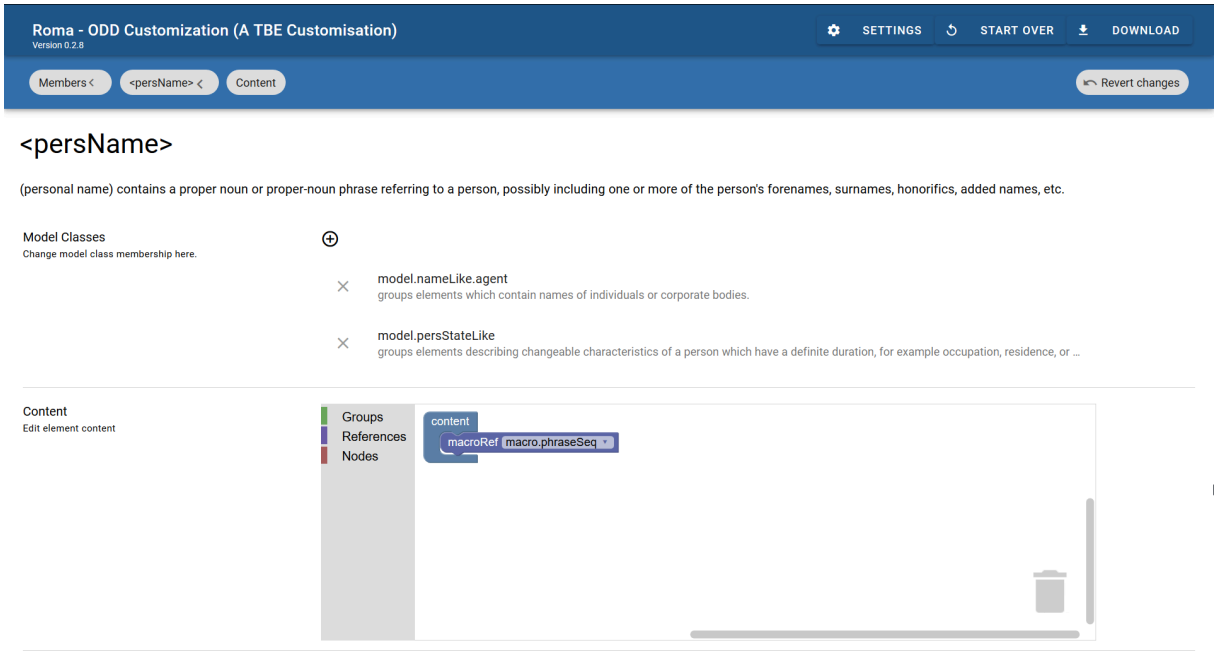


Figure 27. Display of the class membership for the `<persName>` element in Roma.

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.

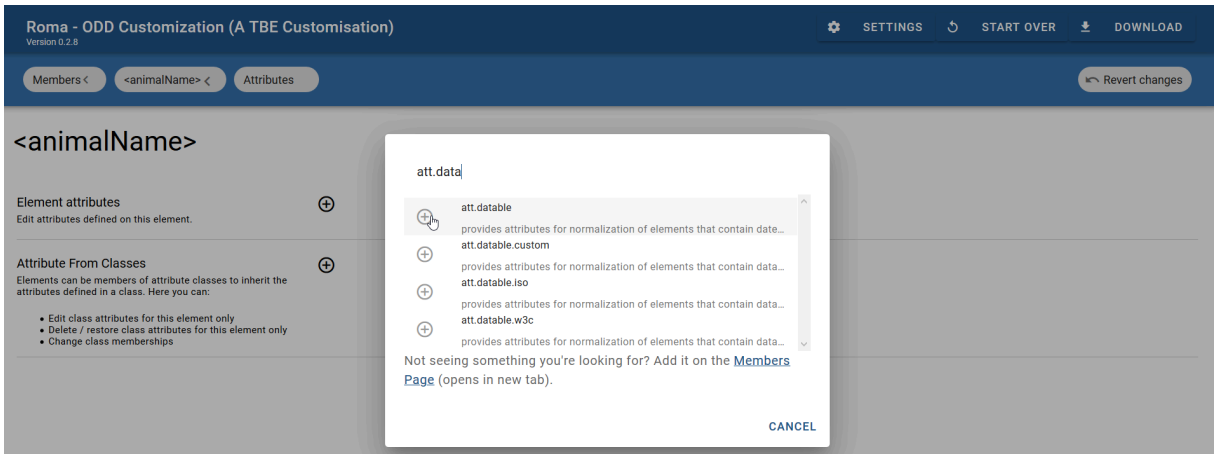


Figure 29. Adding the `<animalName>` element to attribute classes in Roma.

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.

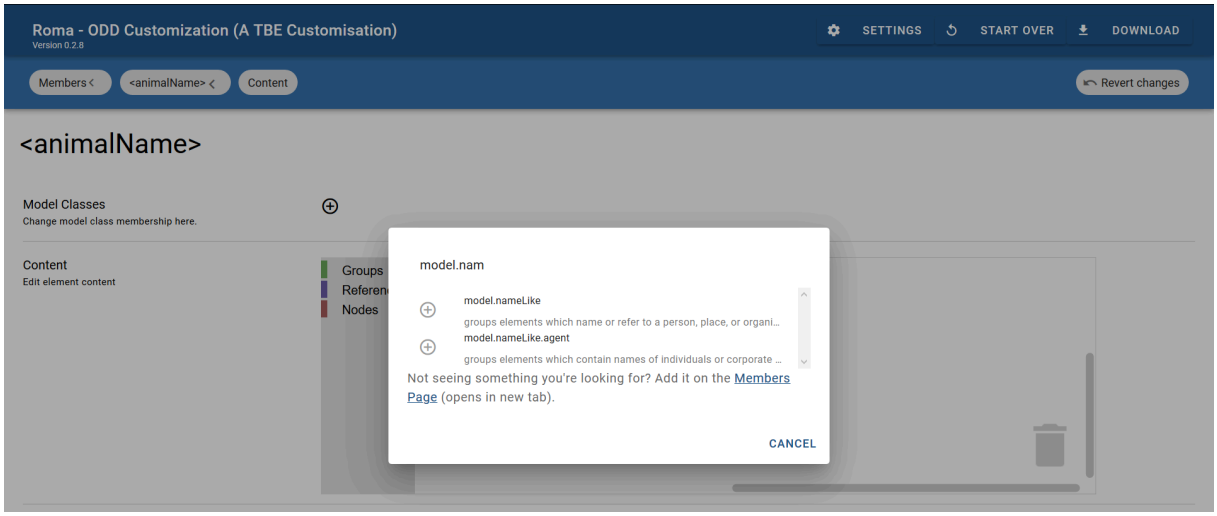


Figure 30. Editing the class membership of the `<animalName>` element in Roma.

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.

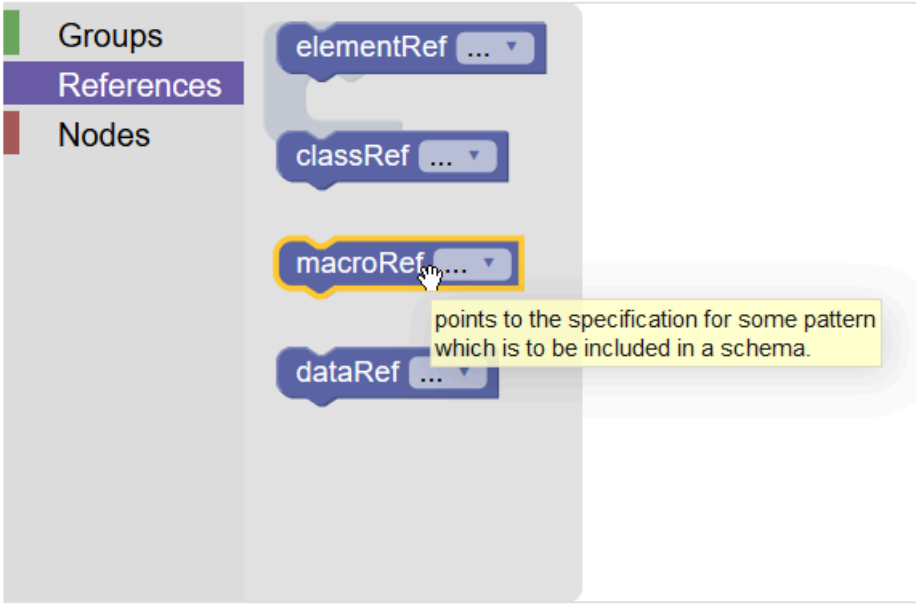


Figure 31. Selecting a macro for the content model of the `<animalName>` element in Roma.

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.

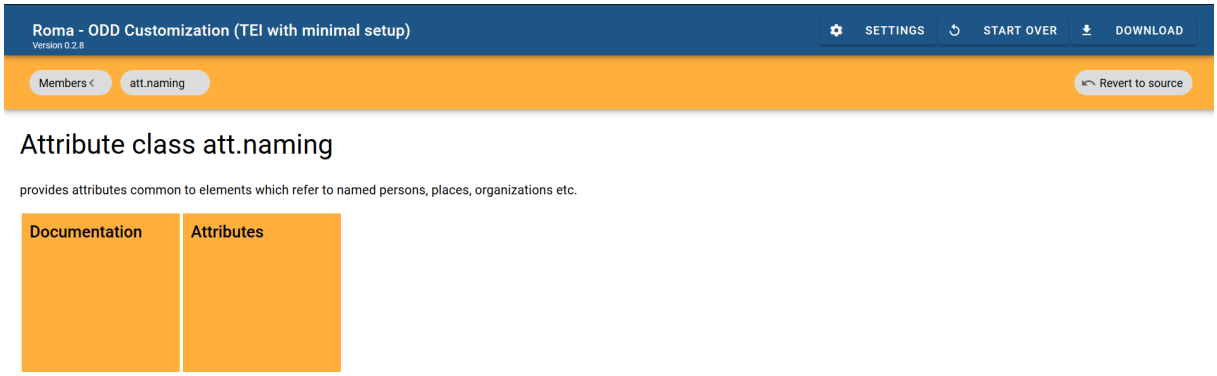


Figure 33. Editing an attribute class for a TEI customisation in Roma.

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:

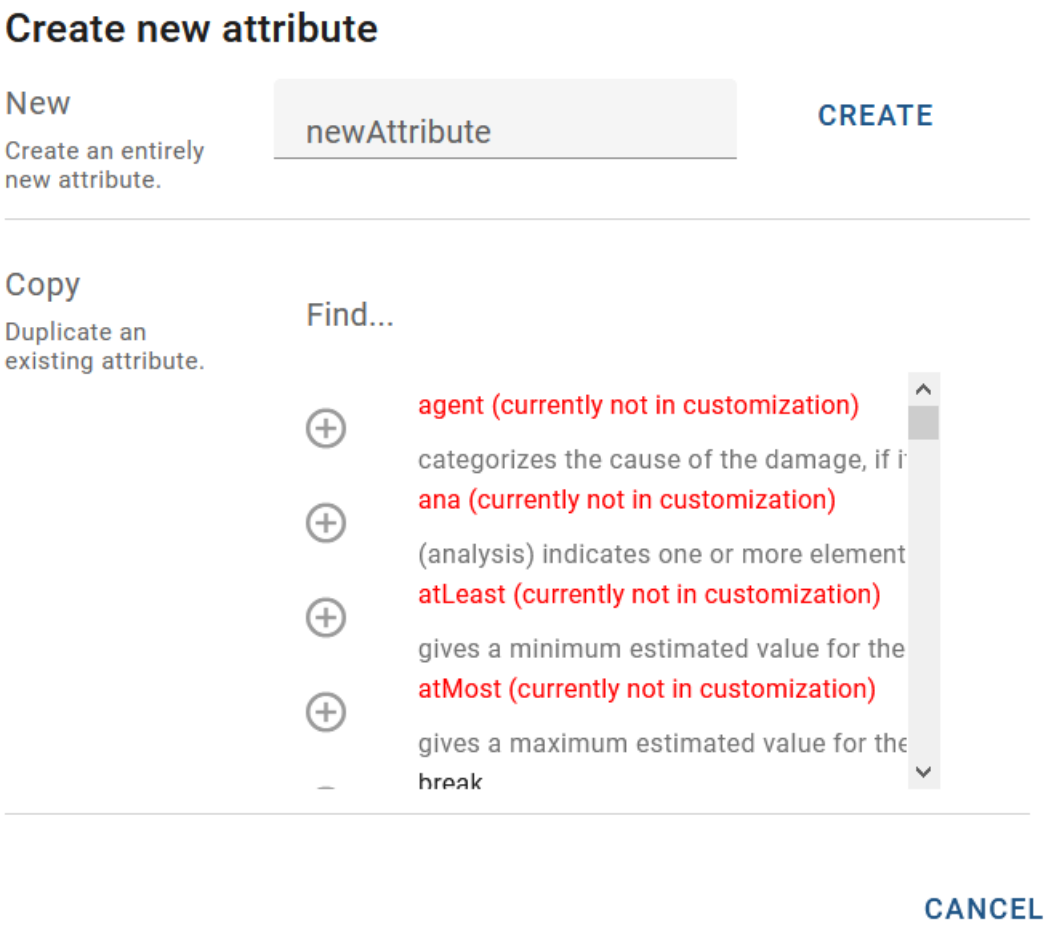


Figure 34. Creating a new attribute in Roma.

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.

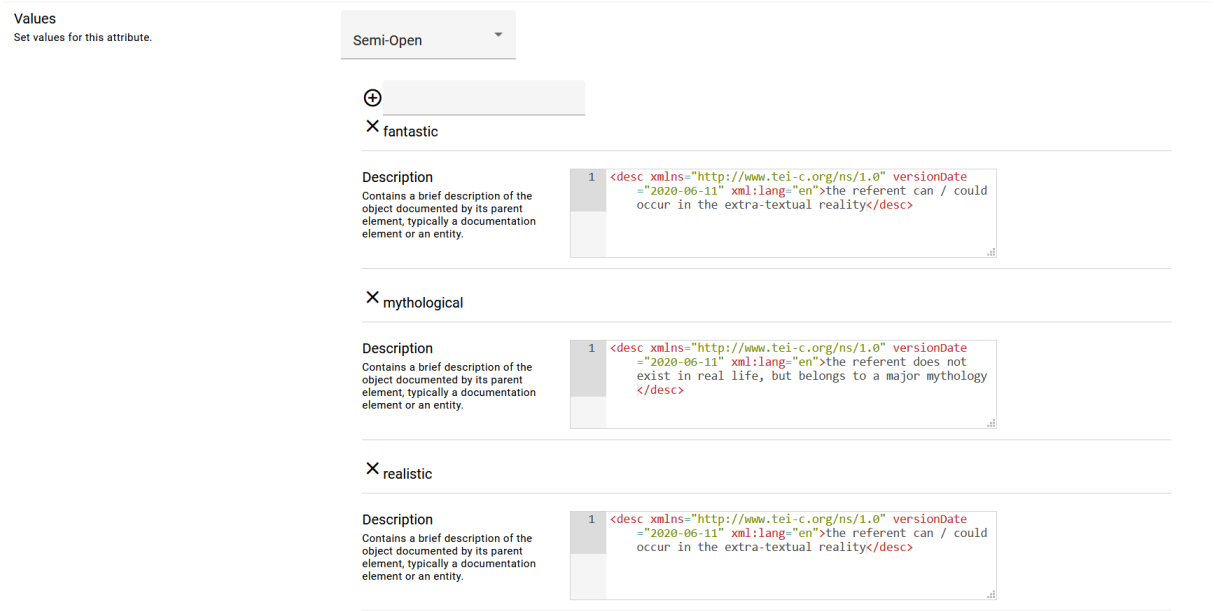


Figure 36. Defining attribute values for a new attribute in Roma.

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.

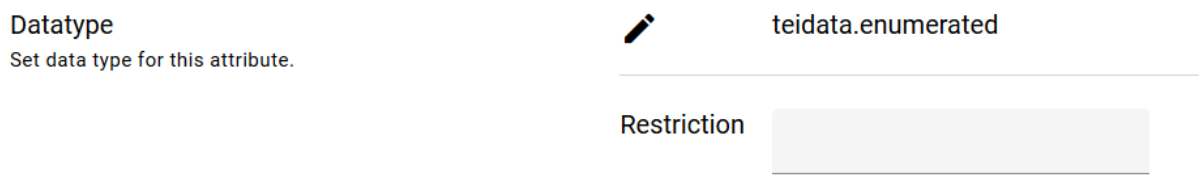


Figure 37. Defining the datatype for a new attribute in Roma.

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:

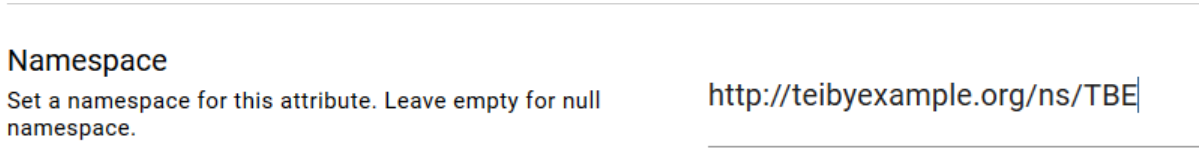


Figure 38. Defining a namespace for a new attribute in Roma.

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.

7. Summary

This tutorial started from a sample encoding project: the encoding of Lewis Carroll's novel *Alice's Adventures in Wonderland*. An analysis of this mini-project's needs identified following encoding goals:

- Encoding of structural elements: the document, title page, document title, chapters, headings, (sub)divisions, paragraphs, quotations, citations, page breaks, figures, line groups.
- Encoding of names for persons, places and animals in the story, with an additional requirement for an experimental analysis of the ontological status of their referents.

Throughout this tutorial, a TEI customisation was developed step-by-step from which TEI schemas can be generated that fit these needs. After selection of relevant TEI modules and elements, selection of element-specific and attribute class attributes, and the addition of new elements and attributes, this is the final version of the ODD file for our **`TBEcustom`** customisation:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>A TBE Customisation</title>
        <author>The TBE crew</author>
      </titleStmt>
      <publicationStmt>
        <publisher>TEI Consortium</publisher>
        <availability status="free">
          <licence target="http://creativecommons.org/licenses/by-sa/3.0/">
           Distributed under a Creative Commons Attribution-ShareAlike 3.0 Unported
           License </licence>
```

```
      <licence target="http://www.opensource.org/licenses/BSD-2-Clause">
        <p>Copyright 2013 TEI Consortium.</p>
        <p>All rights reserved.</p>
        <p>Redistribution and use in source and binary forms, with or without
         modification, are permitted provided that the following conditions are
         met:</p>
        <list>
          <item>Redistributions of source code must retain the above copyright
           notice, this list of conditions and the following disclaimer.</item>
          <item>Redistributions in binary form must reproduce the above
           copyright notice, this list of conditions and the following disclaimer
           in the documentation and/or other materials provided with the
           distribution.</item>
        </list>
        <p>This software is provided by the copyright holders and contributors
         "as is" and any express or implied warranties, including, but not
         limited to, the implied warranties of merchantability and fitness for
         a particular purpose are disclaimed. In no event shall the copyright
         holder or contributors be liable for any direct, indirect, incidental,
         special, exemplary, or consequential damages (including, but not limited to,
         procurement of substitute goods or services; loss of use, data, or profits;
         or business interruption) however caused and on any theory of liability,
         whether in contract, strict liability, or tort (including negligence or
         otherwise) arising in any way out of the use of this software, even if
         advised of the possibility of such damage.</p>
      </licence>
      <p>TEI material can be licensed differently depending on the use you intend
       to make of it. Hence it is made available under both the CC+BY and BSD-2
       licences. The CC+BY licence is generally appropriate for usages which
       treat TEI content as data or documentation. The BSD-2 licence is generally
       appropriate for usage of TEI content in a software environment. For further
       information or clarification, please contact the <ref target="mailto:info@tei
      -c.org">TEI Consortium</ref>. </p>
    </availability>
  </publicationStmt>
  <sourceDesc>
    <p>Created from scratch by James Cummings, but looking at previous tei_minimal
     and tei_bare exemplars by SPQR and LR.</p>
```

```
    </sourceDesc>
  </fileDesc>
</teiHeader>
<text>
  <body>
    <head>A Minimal TEI Customization</head>
    <p>This TEI ODD defines a TEI customization that is as minimal as possible
     and the schema generated from it will validate a document that is
     minimally valid against the TEI scheme. It includes only the ten required
     elements: <list rend="numbered">
      <item><gi>teiHeader</gi> from the header module to store required
       metadata</item>
      <item><gi>fileDesc</gi> from the header module to record information about this
       file</item>
      <item><gi>titleStmt</gi> from the header module to record information about the
       title</item>
      <item><gi>publicationStmt</gi> from the header module to detail how it is
       published</item>
      <item><gi>sourceDesc</gi> from the header module to record where it is
       from</item>
      <item><gi>p</gi> from the core module for use in the header and the body</item>
      <item><gi>title</gi> from the core module for use in the titleStmt</item>
      <item><gi>TEI</gi> from the textstructure module because what is a TEI file
       without that?</item>
      <item><gi>text</gi> from the textstructure module to hold some text</item>
      <item><gi>body</gi> from the textstructure module as a place to put that
       text</item>
    </list></p>
    <schemaSpec 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
       sourceDesc"/>
      <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 doc
      Imprint docDate docAuthor byline div"/>
      <moduleRef key="tei"/>
```

```
<moduleRef key="namesdates" include="persName placeName"/>
<elementSpec ident="name" mode="change">
  <attList>
    <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="subtype" mode="delete"/>
    <attDef ident="calendar" mode="delete"/>
    <attDef ident="cert" mode="delete"/>
    <attDef ident="evidence" mode="delete"/>
    <attDef ident="from" mode="delete"/>
    <attDef ident="full" mode="delete"/>
    <attDef ident="instant" mode="delete"/>
    <attDef ident="notAfter" mode="delete"/>
    <attDef ident="notBefore" mode="delete"/>
    <attDef ident="period" mode="delete"/>
    <attDef ident="resp" mode="delete"/>
    <attDef ident="sort" mode="delete"/>
    <attDef ident="source" mode="delete"/>
    <attDef ident="to" mode="delete"/>
    <attDef ident="when" mode="delete"/>
    <attDef ident="nymRef" mode="delete"/>
    <attDef ident="role" mode="delete"/>
  </attList>
</elementSpec>
<classSpec 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>
          <dataRef key="teidata.enumerated"/>
        </datatype>
        <valList mode="change">
          <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>
  <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.global"/>
      <memberOf key="model.persStateLike"/>
      <memberOf key="model.nameLike.agent"/>
      <memberOf key="att.datable"/>
      <memberOf key="att.editLike"/>
      <memberOf key="att.personal"/>
      <memberOf key="att.typed"/>
    </classes>
```

```
        <content>
          <macroRef key="macro.phraseSeq"/>
        </content>
      </elementSpec>
    </schemaSpec>
  </body>
 </text>
</TEI>
```

This ODD file allows the generation of a TEI schema for the encoding of the document. The following example illustrates how the encoding could make use of the features defined in the ODD file (notice how the **`http://teibyexample.org/ns/TBE`** namespace is used to distinguish the added elements and attributes, and bound to the namespace prefix “TBE”):

```
<TEI xmlns="http://www.tei-c.org/ns/1.0" xmlns:tbe="http://teibyexample.org/ns/TBE">
 <teiHeader>
  <!-- ... -->
 </teiHeader>
 <text>
  <body>
   <!-- ... -->
   <div type="chapter">
    <!-- ... -->
    <pb n="157"/>
    <figure>
     <graphic url="images/lobster.jpg"/>
     <figDesc>The lobster sugaring its hair.</figDesc>
    </figure>
    <p><q who="#alice">"How the creatures order
     one about, and make one repeat lessons!"</q>
     thought <persName tbe:ontStatus="realistic">Alice</persName>, <q who="#alice">"I
     might just as well be at school at once."</q> However, she
     got up, and began to repeat it, but her head was so full of
     the <title type="song"><tbe:animalName tbe:ontStatus="realistic">Lobster</tbe:animalName>-
    Quadrille</title>, that she hardly knew what she was saying, and the words came
     very queer indeed:—</p>
    <q rend="blockquote" who="#alice">
```

```
  <lg>
    <l>"'Tis the voice of
     the <tbe:animalName tbe:ontStatus="realistic">lobster</tbe:animalName>; I
     heard him declare,</l>
    <l>
      <q who="#lobster">'You have baked me too brown, I must sugar my hair.'</q>
    </l>
    <l>As a duck with its eyelids, so he with his nose</l>
    <l>Trims his belt and his buttons, and turns out his toes."</l>
  </lg>
</q>
<p><q who="#gryphon">"That's different from
 what <emph>I</emph> used to say when I was a child,"</q> said
 the <tbe:animalName tbe:ontStatus="mythological">Gryphon</tbe:animalName>.</p>
<pb n="158"/>
<p><q who="#mockTurtle">"Well, I never heard it before,"</q>
 said the <tbe:animalName tbe:ontStatus="fantastic">Mock
 Turtle</tbe:animalName>; <q who="#mockTurtle">"but it sounds uncommon
 nonsense."</q></p>
<p><persName tbe:ontStatus="realistic">Alice</persName> said nothing;
 she had sat down with her face in her hands, wondering if anything
 would <emph>ever</emph> happen in a natural way again.</p>
<p><q who="#mockTurtle">"I should like to have it explained,"</q> said
 the <tbe:animalName tbe:ontStatus="fantastic">Mock Turtle</tbe:animalName>.</p>
<p><q who="#gryphon">"She can't explain it,"</q> said
 the <tbe:animalName tbe:ontStatus="mythological">Gryphon</tbe:animalName>
 hastily. <q who="#gryphon">"Go on with the next verse."</q></p>
<p><q who="#mockTurtle">"But about his toes?"</q>
 the <tbe:animalName tbe:ontStatus="fantastic">Mock Turtle</tbe:animalName>
 persisted. <q who="#mockTurtle">"How <emph>could</emph> he turn them out with
 his nose, you know?"</q></p>
<p><q who="#aliceI">"It's the first position in
 dancing."</q> <persName tbe:ontStatus="realistic">Alice</persName> said;
 but she was dreadfully puzzled by the whole thing, and longed to change the
 subject.</p>
```

```
        <p><q who="#gryphon">"Go on with the next verse,"</q>
         the <tbe:animalName tbe:ontStatus="mythological">Gryphon</tbe:animalName>
         repeated impatiently: <q who="#gryphon">"it begins <quote>'I passed by his
         garden.'</quote>"</q></p>
        <p><persName tbe:ontStatus="realistic">Alice</persName> did not dare to disobey,
         though she felt sure it would all come wrong, and she went on in a trembling
         voice:—</p>
        <pb n="159"/>
        <!-- ... -->
      </div>
      <!-- ... -->
    </body>
  </text>
</TEI>
```

8. What's Next?

You have reached the end of this tutorial module covering TEI customisations and Roma. You can now either

- proceed with other TEI by Example modules
- have a look at the examples section for the Customising TEI, ODD, Roma module.
- take an interactive test. This comes in the form of a set of multiple choice questions, each providing a number of possible answers. Throughout the quiz, your score is recorded and feedback is offered about right *and* wrong choices. Can you score 100%? Test it here!