# TABLE OF CONTENTS

1. Introduction ........................................................................................................................ 1
2. Customising TEI: why and how? ...................................................................................... 2
3. Selecting and restricting the TEI model ........................................................................... 6
   3.1 Starting from a minimal schema .............................................................................. 6
      3.1.1 Generating a schema ......................................................... 9
      3.1.2 Generating documentation for a schema ........................................... 11
      3.1.3 Generating an ODD file ....................................................................... 12
   3.2 What does a minimal TEI customisation tell us? ................................................. 14
   3.3 Selecting modules and elements .............................................................................. 20
   3.4 Changing attributes ............................................................................................... 26
      3.4.1 Changing individual attributes ................................................. 26
      3.4.2 Changing attribute classes ....................................................... 30
4. Extending TEI .................................................................................................................. 37
   4.1 Adding elements .................................................................................................... 38
   4.2 Adding attributes .................................................................................................. 46
   4.3 Other types of extension ....................................................................................... 52
5. Summary ........................................................................................................................... 52
6. What's next? ..................................................................................................................... 59
Customising TEI, ODD, Roma

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, the TEI defines no less than 505 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 [your name]’s documents’. 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, labelled 'blueprint for [your name]’s TEI schema’, walk home and happily start encoding your texts with your TEI schema. In the TEI world, this library visit is called ‘customising TEI’. As it is a visit you will have to repeat often, this tutorial will guide you through the most relevant steps of customising TEI schemas.

A couple of elements in 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 Schemas, or DTDs (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 'blueprint' for your TEI schema. This will allow you to remember the choices made and facilitate you to share your schemas with others. In 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.
This tutorial won’t discuss the different TEI schema formats but instead focus on both the formal ODD way of expressing TEI customisations and the Roma tool. In doing so, it 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. Customising TEI: why and how?

This TBE tutorial module starts as any other module: from a concrete text example. This time, we’ll consider Lewis Carroll’s *Alice in Wonderland*. To get a sense of the structure of the document, here are the first pages:

![Alice's Adventures in Wonderland](image)

*Fig. 1.*
ALICE'S
ADVENTURES IN WONDERLAND.

BY
LEWIS CARROLL.

WITH FORTY-TWO ILLUSTRATIONS
BY
JOHN TENNIEL.

NEW YORK
D. APPLETON AND CO., 445, BROADWAY.
1865.

Fig. 2.

A typical page looks like this:
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
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](http://www.tei-c.org/Guidelines/Customization) 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 [http://www.tei-c.org/Guidelines/Customization](http://www.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 precooked 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.
3. Selecting and restricting the TEI model

3.1 Starting from a minimal schema

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

![Fig. 4.](image)

Roma: generating validators for the TEI

These pages will help you design your own TEI validator, as a DTD, RELAXNG or W3C Schema.

Create a new or upload existing customization

- Build schema (Create a new customisation by adding elements and modules to the smallest recommended schema)
- Reduce schema (Create a new customization by removing elements and modules from the largest possible schema)
- Create customization from template: TEI Absolutely Basic
- Open existing customization

Roma was written by Anja Mittelbach and is maintained by Sebastian Reitz; Sanity check written by Ioan Bernea. Please direct queries to the TEI@Oxford project.
This is the start screen for new customisations, offering you a choice between four options:

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

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

This will produce the main Roma dashboard:
You'll see no less than 10 different tabs at the top of the screen. They are:

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

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

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

### 3.1.1 Generating a schema

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

---

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

Fig. 6.
3.1.2 Generating documentation for a schema

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

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

3.1.3 Generating an ODD file

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

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

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

3.2 What does a minimal TEI customisation tell us?

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

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

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

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

```
```

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

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader>
 <fileDesc>
  <titleStmt>
   <title>
     <!-- Title -->
   </title>
  </titleStmt>
  <publicationStmt>
   <p>
     <!--Publication Information-->
   </p>
  </publicationStmt>
  <sourceDesc>
   <p>
     <!--Information about the source-->
   </p>
  </sourceDesc>
 </fileDesc>
</teiHeader>
<text>
 <body>
  <p>
    <!--Some text here.-->
  </p>
 </body>
</text>
</TEI>
```
On the left hand side, all TEI modules are listed. The right hand column lists all modules that are selected in the current customisation. The names of the modules are presented as hyperlinks pointing to a list of elements defined in that module. To see what elements the core module holds, just click on the ‘core’ hyperlink and see all its elements listed on the next screen:

Fig. 9.
The same can be done for all other modules on the ‘Modules’ tab. If you want more information on the modules or elements, click on the question mark to navigate to the relevant documentation in the TEI Guidelines, or on its name for technical information.

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

**Summary**

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

### 3.3 Selecting modules and elements

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

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

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

![TEI by Example](image.png)

Fig. 12.

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

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

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

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

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

Because all further changes to the ODD file in this TBE tutorial module will affect only its `<schemaSpec>` part, the example fragments will focus on this element.
<!-- ... -->
<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 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>"Lobster-Quadrille"</title>, that she hardly knew what she was saying, and the words came very queer indeed:—</q>
</p>
<q rend="blockquote" who="alice">
<T>"Tis the voice of the <name type="animal">lobster</name>; I heard him declare,"</T>
</q>
<q rend="blockquote" who="lobster">"You have baked me too brown, I must sugar my hair:"
</q>
<q rend="blockquote" who="lobster">"As a duck with its eyelids, so he with his nose
</q>
<q rend="blockquote" who="lobster">"Trims his belt and his buttons, and turns out his toes."</q>
<q rend="blockquote" who="alice">
<T>"That's different from what <em>1</em> used to say when I was a child,"</T>
said the <name type="animal">Gryphon</name>.<p>
</p>
<q rend="blockquote" who="mockTurtle">"Well, I never heard it before,"</q>
said the <name type="animal">Mock Turtle</name>.
<q rend="blockquote" who="mockTurtle">"but it sounds uncommon nonsense."</q>
<q rend="blockquote" who="person">Alice</name> said nothing; she had sat down with her face in her hands, wondering if anything would <em>ever</em> happen in a natural way again.<p>
</p>
<q rend="blockquote" who="mockTurtle">"I should like to have it explained,"</q>
said the <name type="animal">Mock Turtle</name>.<p>
</p>
<q rend="blockquote" who="grphon">"She can't explain it,"</q>
said the <name type="animal">Gryphon</name> hastily.
<q rend="blockquote" who="grphon">"Go on with the next verse."</q>
</q>
<p>
  <q who="mockTurtle">"But about his toes?"</q>
</p>

the

Mock Turtle persisted.

<q who="mockTurtle">"How <emph>could</emph> he turn them out with his nose, you know?"</q>

<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>

<p>
  <pb n="159"/>
</p>

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

---

**Summary**

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

3.4.1 Changing individual attributes

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

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

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

In order to edit its attributes, click the relevant ‘Change attributes’ hyperlink on the right hand side. This produces a similar page, only now the attributes are listed:
By clicking the ‘type’ hyperlink, a page is shown with the definition of the @type attribute. There you can determine whether the attribute should be mandatory or optional, what the datatype and occurrence of its value(s) should be, its default value, a list of possible values, and whether this list is exhaustive or not. Finally, the prose description of the attribute can be given. For our purpose, we can leave most settings unchanged but only add a comma-separated list of the values we expect, in the ‘List of values’ field:

**Fig. 13.**

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

**Fig. 14.**
Pressing the ‘Save’ button returns us to the attribute list page. Now, another change we want to make is getting rid of the @nymRef attribute. This is meant to point to a canonical or normalized form of a name, for onomastic purposes. As this is too specific for our purposes with the Alice story, we’ll delete it. This way, it won’t bother us when actually marking up the names in the text. Selection of attributes is similar to selection of elements (see the previous section (p. 20)). Just check the desired option: ‘Include’ (default) to include the attribute to this element in the schema; ‘Exclude’ to delete it. Selecting the ‘Exclude’ option next to the @nymRef attribute will do so, after pressing the ‘Save’ button.

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

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

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

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

**Summary**

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

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

### 3.4.2 Changing attribute classes

Similar to the organisation of elements in modules, attributes are grouped into classes. This facilitates the definition of elements that share the same attributes, by declaring them as members of an attribute class. For example, all TEI elements are declared as members of

---

6 Note, however, that a full attribute definition consists of more fields, like a description, declarations of datatype and occurrence indicators. These are discussed later (p. 46) in this module.
the `att.global` attribute class, which defines the global attributes `@xml:id`, `@n`, `@xml:lang`, `@rend`, `@rendition`, and `@xml:base`.

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

To find out the attribute classes an element (in this case, the `<name>` element) belongs to:

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

This calls a page with the definition of the `<name>` element. If you scroll down to the ‘Attribute classes’ section, you will see the ‘att.naming’ option selected:
As always in Roma, clicking the name of this attribute class will produce a formal definition of this attribute class:
This tells us that the `att.naming` attribute class defines the attribute `@nymRef` directly, and by reference to the `att.canonical` attribute class declares the `@key` and `@ref` attributes for a whole range of name-related elements, of which `<name>` is only one.

Now, instead of removing the `@nymRef` attribute only from the `<name>` element as we did in the previous section, we could as well delete it globally from all these elements at once. This can be done by changing the attribute class itself. In Roma, click the ‘Change Classes’ tab. This calls a list of all attribute classes defined in TEI:
Fig. 17.

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

**Summary**

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

### 4. Extending TEI

So far, all modifications described were reductions of the general TEI model: either by selecting existing modules, elements, or attributes; or reducing the possible values of attributes. These kinds of modifications can be seen as 'clean' modifications: they define true subsets of the TEI model (provided they adhere to the minimal rules sketched out above (p. 16)). Put differently: a document that is valid against such a schema will always be valid against the maximal TEI schema.

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

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

Be careful, though, when changing global attribute definitions. Some elements may use attributes that are defined in an attribute class directly, without referring to the class. For example, the `@type` attribute is defined in the class `att.typed`; however, the `<title>` element has a `@type` attribute that is defined literally. Changing something to the definition of `@type` in the `att.typed` definition will thus not affect the `@type` attribute of the `<title>` element. On the other hand, be aware that changing attribute classes can have very wide ranging effects! Always make sure to study the relevant parts of the TEI Guidelines.
In order to guarantee maximal interoperability for TEI documents, the TEI Guidelines strongly advise to formally separate added elements and attributes from the standard TEI schema. This can be done by defining them in another namespace than the TEI namespace ("http://www.tei-c.org/ns/1.0"). You can freely decide on this namespace; for the purpose of this tutorial, we'll use a dedicated TBE namespace: "http://www.teibyexample.org/".

4.1 Adding elements

As illustrated above (p. 20), the TEI core module already provides the <name> element, whose @type attribute can be used to provide more details about the type of name. However, suppose we want to categorise names along more dimensions than just the type of creature they refer to, or we are not entirely satisfied with such a mechanism of subtyping general elements for rather diverse uses. For such cases, the TEI provides a set of more specialised naming elements that add more semantic detail and leave more room for further (sub)typing. They are grouped in the namesdates module. Let's have a look at what namesdates has to offer:

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

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

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

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

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

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

- **Name** the name of the element
- **Namespace** the namespace of the (non-TEI) element
- **Description** a prose description of the element's meaning
Model classes  

a formal declaration of the 'behaviour' of the element: assigning it to a model class will determine the contexts in which it may occur

Attribute classes  

a formal declaration of the attributes that will be assigned to the element

Contents  

a formal declaration of the content type for the element, either by

- selecting one of the TEI defined classes in the dropdown list
- providing a custom Relax NG definition in the text box below

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

Crosslink

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

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

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

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

model.nameLike  

groups elements which name or refer to a person, place, or organisation

model.nameLike.agent  

groups elements which contain names of individuals or corporate bodies
model.persStateLike  groups elements describing changeable characteristics of a person which have a definite duration, for example occupation, residence, or name.

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

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

These attribute classes define all attributes that can occur on the `<persName>` element.

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

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

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

Next, we must define how the `<animalName>` element will behave. Copy the ‘Model Classes’ from `<persName>`: tick the boxes next to ‘model.nameLike’, ‘model.nameLike.agent’, and ‘model.persStateLike’. For the ‘Attribute Classes’, select the ‘att.datable’, ‘att.editLike’, ‘att.personal’, and ‘att.typed’ options. The contents of `<animalName>` will consist of the elements and text defined in the `macro.specialPara` macro. This macro is included in the dropdown list, so we can suffice with selecting ‘macro.specialPara’ from this list.
Save your changes by pressing the ‘Save’ button. This returns us to the ‘Add Elements’ tab, which now consists of a list of added elements:

---

8 Note that the ‘Contents’ dropdown list on this page not only includes content macros (starting with `macro.`), but also attribute datatypes (starting with `data.`). These are strictly speaking irrelevant in this context, as attribute datatypes only apply to attribute definitions, not to the definition of an element’s contents. You can safely ignore them here, and scroll down to the content macros (starting with `macro.`).
Fig. 21.

Now let's have a look at the underlying ODD file (click the ‘Save Customisation’ tab):
<schemaSpec ident="TBEcustom" docLang="en" prefix="tei_" start="TEI" xml:lang="en">
  <moduleRef key="core"/>
  <moduleRef key="tei"/>
  <moduleRef key="header"/>
  <moduleRef key="textstructure"/>
  <moduleRef key="figures"/>
  <elementSpec module="figures" ident="cell" mode="delete"/>
  <elementSpec module="figures" ident="formula" modes="delete"/>
  <elementSpec module="figures" ident="row" mode="delete"/>
  <elementSpec module="figures" ident="table" modes="delete"/>
  <elementSpec ident="name" modules="core" mode="change">
    <attList>
      <attDef ident="type" mode="change">
        <defaultVal>person</defaultVal>
        <valList type="open" mode="replace">
          <valItem ident="person"/>
          <valItem ident="place"/>
          <valItem ident="animal"/>
        </valList>
      </attDef>
      <attDef ident="nymRef" mode="delete"/>
    </attList>
  </elementSpec>
  <classSpec ident="att.naming" module="tei" mode="change" type="atts">
    <attList>
      <attDef ident="nymRef" mode="delete"/>
    </attList>
  </classSpec>
  <moduleRef key="namesdates"/>
  <elementSpec module="namesdates" ident="addName" modes="delete"/>
  <elementSpec module="namesdates" ident="affiliation" mode="delete"/>
  <elementSpec module="namesdates" ident="age" mode="delete"/>
  <elementSpec module="namesdates" ident="birth" mode="delete"/>
  <elementSpec module="namesdates" ident="bloc" mode="delete"/>
  <elementSpec module="namesdates" ident="climate" mode="delete"/>
  <elementSpec module="namesdates" ident="country" mode="delete"/>
  <elementSpec module="namesdates" ident="death" mode="delete"/>
  <elementSpec module="namesdates" ident="district" mode="delete"/>
  <elementSpec module="namesdates" ident="education" modes="delete"/>
  <elementSpec module="namesdates" ident="event" mode="delete"/>
  <elementSpec module="namesdates" ident="faith" mode="delete"/>
  <elementSpec module="namesdates" ident="floruit" mode="delete"/>
</schemaSpec>
<elementSpec module="namesdates" ident="forename" mode="delete"/>
<elementSpec module="namesdates" ident="genName" mode="delete"/>
<elementSpec module="namesdates" ident="geo" mode="delete"/>
<elementSpec module="namesdates" ident="geogFeat" mode="delete"/>
<elementSpec module="namesdates" ident="geogName" mode="delete"/>
<elementSpec module="namesdates" ident="langKnowledge" mode="delete"/>
<elementSpec module="namesdates" ident="langKnown" mode="delete"/>
<elementSpec module="namesdates" ident="listNym" mode="delete"/>
<elementSpec module="namesdates" ident="listOrg" mode="delete"/>
<elementSpec module="namesdates" ident="listPerson" mode="delete"/>
<elementSpec module="namesdates" ident="listPlace" mode="delete"/>
<elementSpec module="namesdates" ident="location" mode="delete"/>
<elementSpec module="namesdates" ident="nameLink" mode="delete"/>
<elementSpec module="namesdates" ident="nationality" mode="delete"/>
<elementSpec module="namesdates" ident="nym" mode="delete"/>
<elementSpec module="namesdates" ident="occupation" mode="delete"/>
<elementSpec module="namesdates" ident="offset" mode="delete"/>
<elementSpec module="namesdates" ident="org" mode="delete"/>
<elementSpec module="namesdates" ident="orgName" mode="delete"/>
<elementSpec module="namesdates" ident="person" mode="delete"/>
<elementSpec module="namesdates" ident="personGrp" mode="delete"/>
<elementSpec module="namesdates" ident="place" mode="delete"/>
<elementSpec module="namesdates" ident="population" mode="delete"/>
<elementSpec module="namesdates" ident="region" mode="delete"/>
<elementSpec module="namesdates" ident="relation" mode="delete"/>
<elementSpec module="namesdates" ident="relationGrp" mode="delete"/>
<elementSpec module="namesdates" ident="residence" mode="delete"/>
<elementSpec module="namesdates" ident="roleName" mode="delete"/>
<elementSpec module="namesdates" ident="settlement" mode="delete"/>
<elementSpec module="namesdates" ident="sex" mode="delete"/>
<elementSpec module="namesdates" ident="socsecStatus" mode="delete"/>
<elementSpec module="namesdates" ident="state" mode="delete"/>
<elementSpec module="namesdates" ident="surname" mode="delete"/>
<elementSpec module="namesdates" ident="terrain" mode="delete"/>
<elementSpec module="namesdates" ident="trait" mode="delete"/>
<elementSpec ident="animalName" ns="http://www.teibyexample.org/" mode="add">
  <desc>contains a proper noun referring to an animal</desc>
  <classes>
    <memberOf key="model.nameLike"/>
    <memberOf key="model.nameLike.agent"/>
    <memberOf key="model.persStateLike"/>
    <memberOf key="att.datable"/>
  </classes>
</elementSpec>
As could be expected, this time the `namesdates` module is included by a `<moduleRef/>` element. Since we only retained the `<persName>` and `<placeName>` elements from this module in our `TBEcustom` customisation, all other 48 elements of this module are explicitly deleted by a dedicated `<elementSpec>` element. Each of these has the value `delete` for its `@mode` attribute, and identifies the element in the `@ident` attribute.

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

9 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. As with other element specifications, 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.

9 Syntactically, the TEI model does not require you to use different namespaces for non-TEI elements, but strongly advises you to: this is the safest way to avoid name collisions. You can for example define a `<name xmlns="http://www.teibyexample.org"/>` variant that differs from the standard TEI `<name>` element. For the sake of clarity, however, this is not really advisable.
containing `<memberOf>` declarations for each model or attribute class to which the element is added. These TEI classes are identified with a `@key` attribute. The content of the element is declared in the `<content>` element, containing either new Relax NG definitions, or Relax NG references to existing TEI macros.

### 4.2 Adding attributes

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

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

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

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

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

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

```
<persName fantastic="no">Alice</persName>
<animalName realistic="0.5">Mock Turtle</animalName>
<animalName 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'. In Roma, this can be declared next to the field labelled ‘Add a new attribute’. In the ‘Description’ field we'll describe it as:

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

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

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

However, it is prone to be extended with other categories, and would probably allow more categories to be applied simultaneously, for names referring to ambiguous creatures or places.

This analysis obviously translates into an open list (option 'no' for ‘Closed list?’) of these values (‘List of values’):

```
realistic,mythological,fantastic
```

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

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

**Fig. 22.**
After saving the ODD file (by clicking the ‘Save Customisation’ tab), we'll notice that the `<classSpec>` element is updated to:

---

**Fig. 23.**

---

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

This introductory tutorial doesn’t cover the advanced inner mechanisms of TEI nor Relax NG; for more information you can read section 1.4.2 of the TEI Guidelines, or the reference section in Appendix E.

Finally, the list of possible values is given inside `<valList>`, which is declared as an open list (`@type = open`).

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

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

Summary

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

4.3 Other types of extension

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

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

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

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

5. Summary

This tutorial started from a sample encoding project: 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.

The realisation of these encoding goals allow encoders to mark up the text's basic structure, and support a specific (tentative) analysis of the names in the story, as exemplified by the encoded fragment at the end of this section. The encoded text could be used to generate an edition, analyse the distribution of realistic vs fantastic vs mythological characters throughout the story, isolate the quotations from the different characters (for a qualitative analysis of their language), and so on.

Throughout this tutorial, a TEI customisation was developed step-by-step that should be able to generate TEI schemas that fit these needs. After selection of relevant TEI modules and elements, selecting individual attributes within the declarations of elements and attribute classes, and adding new elements and attributes, this is the final version of the ODD file for our **TBEcustom** customisation:
<teiHeader>
<fileDesc>
<titleStmt>
<title>A TBE customisation</title>
<author>The TBE Crew</author>
</titleStmt>
<pubstmt>
<p>for use by whoever wants it</p>
</pubstmt>
<sourceDesc>
<p>created on Wednesday 05th November 2008 09:03:56 AM</p>
</sourceDesc>
</fileDesc>
</teiHeader>
<text>
<front>
<divGen type="toc"/>
</front>
<body>
<p>My TEI Customization starts with modules tei, core, textstructure and header</p>
<schemaSpec id="TBEcustom" docLang="en" prefix="tei_" start="TEI" xml:lang="en">
<moduleRef key="core"/>
<moduleRef key="tei"/>
<moduleRef key="header"/>
<moduleRef key="textstructure"/>
<moduleRef key="figures"/>
<elementSpec module="figures" ident="cell" mode="delete"/>
<elementSpec module="figures" ident="formula" mode="delete"/>
<elementSpec module="figures" ident="row" mode="delete"/>
<elementSpec module="figures" ident="table" mode="delete"/>
<elementSpec ident="name" module="core" mode="change">
<attList>
<attDef ident="type" mode="change">
<defaultVal>person</defaultVal>
<valList type="open" mode="replace">
<valItem ident="person"/>
<valItem ident="place"/>
<valItem ident="animal"/>
</valList>
</attDef>
</atts>
</elementSpec>
</schemaSpec>
</body>
</text>

5. Summary
<attList>
</elementSpec>
<classSpec ident="att.naming" module="tei" mode="change" type="atts">
<attList>
<attDef ident="nymRef" mode="delete"/>
<attDef ident="ontStatus" mode="add" ns="http://www.teibyexample.org/">
<desc>describes the ontological status of a name’s referent</desc>
<datatype minOccurs="1" maxOccurs="unbounded">
<rng:ref name="data.enumerated"/>
</datatype>
</attDef>
</attList>
</classSpec>
<moduleRef key="namesdates"/>
<elementSpec module="namesdates" ident="addName" mode="delete"/>
<elementSpec module="namesdates" ident="affiliation" mode="delete"/>
<elementSpec module="namesdates" ident="age" mode="delete"/>
<elementSpec module="namesdates" ident="birth" mode="delete"/>
<elementSpec module="namesdates" ident="bloc" mode="delete"/>
<elementSpec module="namesdates" ident="climate" mode="delete"/>
<elementSpec module="namesdates" ident="country" mode="delete"/>
<elementSpec module="namesdates" ident="death" mode="delete"/>
<elementSpec module="namesdates" ident="district" mode="delete"/>
<elementSpec module="namesdates" ident="education" mode="delete"/>
<elementSpec module="namesdates" ident="event" mode="delete"/>
<elementSpec module="namesdates" ident="faith" mode="delete"/>
<elementSpec module="namesdates" ident="floruit" mode="delete"/>
<elementSpec module="namesdates" ident="forename" mode="delete"/>
<elementSpec module="namesdates" ident="genName" mode="delete"/>
<elementSpec module="namesdates" ident="geo" mode="delete"/>
<elementSpec module="namesdates" ident="geogFeat" mode="delete"/>
<elementSpec module="namesdates" ident="geogName" mode="delete"/>
<elementSpec module="namesdates" ident="langKnowledge" mode="delete"/>
<elementSpec module="namesdates" ident="langKnown" mode="delete"/>
<elementSpec module="namesdates" ident="listNym" mode="delete"/>
<elementSpec module="namesdates" ident="listOrg" mode="delete"/>
<elementSpec module="namesdates" ident="listPerson" mode="delete"/>

5. Summary
<elementSpec module="namesdates" ident="listPlace" mode="delete"/>
<elementSpec module="namesdates" ident="location" mode="delete"/>
<elementSpec module="namesdates" ident="nameLink" mode="delete"/>
<elementSpec module="namesdates" ident="nationality" mode="delete"/>
<elementSpec module="namesdates" ident="nym" mode="delete"/>
<elementSpec module="namesdates" ident="occupation" mode="delete"/>
<elementSpec module="namesdates" ident="offset" mode="delete"/>
<elementSpec module="namesdates" ident="org" mode="delete"/>
<elementSpec module="namesdates" ident="orgName" mode="delete"/>
<elementSpec module="namesdates" ident="person" mode="delete"/>
<elementSpec module="namesdates" ident="personGrp" mode="delete"/>
<elementSpec module="namesdates" ident="place" mode="delete"/>
<elementSpec module="namesdates" ident="population" mode="delete"/>
<elementSpec module="namesdates" ident="region" mode="delete"/>
<elementSpec module="namesdates" ident="relation" mode="delete"/>
<elementSpec module="namesdates" ident="relationGrp" mode="delete"/>
<elementSpec module="namesdates" ident="residence" mode="delete"/>
<elementSpec module="namesdates" ident="roleName" mode="delete"/>
<elementSpec module="namesdates" ident="settlement" mode="delete"/>
<elementSpec module="namesdates" ident="sex" mode="delete"/>
<elementSpec module="namesdates" ident="socSecStatus" mode="delete"/>
<elementSpec module="namesdates" ident="state" mode="delete"/>
<elementSpec module="namesdates" ident="surname" mode="delete"/>
<elementSpec module="namesdates" ident="terrain" mode="delete"/>
<elementSpec module="namesdates" ident="trait" mode="delete"/>
<elementSpec module="namesdates" ident="animalName" ns="http://www.teibyexample.org/" mode="add">
  <desc>contains a proper noun referring to an animal</desc>
  <classes>
    <memberOf key="model.nameLike"/>
    <memberOf key="model.nameLike.agent"/>
    <memberOf key="model.persStateLike"/>
    <memberOf key="att.datable"/>
    <memberOf key="att.editLike"/>
    <memberOf key="att.naming"/>
    <memberOf key="att.typed"/>
  </classes>
  <content>
    <rng:ref name="macro.specialPara"/>
  </content>
</elementSpec>
</schemaSpec>

5. Summary
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 (note how the 'http://www.teibyexample.org/' namespace is used to distinguish the added elements and attributes, and bound to the namespace prefix "TBE"): 

```xml
<text>
</TEI>
```
5. Summary
The lobster sugaring its hair.

"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 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.

"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
  <TBE:animalName TBE:ontStatus="realistic 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>
  <q who="aliceI">"It's the first position in dancing."</q>
  <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>
<p>
  <pb n="159"/>
    <!-- ... -->
  </div>
  <!-- ... -->
</body>
</text>
</TEI>

6. 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!