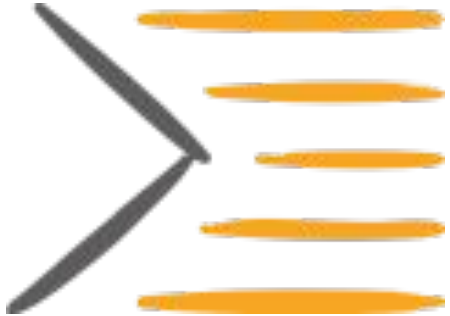
Introduction to TEI Publisher: how to make a scholarly edition in 60 minutes

Helena Bermúdez Sabel (Jinntec)

TEI Publisher



open source project developed since 2015

grassroots community efforts coordinated by e-editiones

Why TEI Publisher?

- Free and open-source software
- Rooted in open standards, with sensible default options that facilitate both the annotation and publication without programming
- Community: e-editiones (grassroots organization with the goal of supporting editions through the promotion of open standards and community collaboration)

Demo

- Alfred Escher
- Register rokopisov slovenskega slovstva
- LGPN-Ling
- EDEp
- Grimm Fairy Tales

TEI Processing Model

- The keystone of TEI Publisher: the implementation of the TEI Processing Model
- It formalizes the description of transformation rules in TEI (implementation independent)
- Officially part of the TEI Guidelines since 2016
- More sustainable and usually faster than customized, project-specific transformations
- It is the source of all TEI Publisher textual transformations

TEI Source

```
<TEI>
  <text>
    <body>
```

TEI Fragment

```
<div type="chapter">
  <head>
    <p>
```

```
<div type="chapter">
  <head>
    <list>
      <item>
```

ODD

```
<elementSpec ident="div">
  <model predicate="@type='chapter'"
    behaviour="section">
  <model behaviour="block">

<elementSpec ident="head">
  <model behaviour="heading">

<elementSpec ident="list">
  <model behaviour="list">
```

HTML FO

LaTeX ePub

More than transforming TEI to HTML

Input formats:

- TEI
- DocBook
- MS Word
- JATS
- Markdown

Publication formats:

- Web application, HTML
- EPUB
- PDF (and LaTeX source)
- Markdown

```
<elementSpec ident="name" mode="change">
    <model predicate="@type='place'" behaviour="inline">
        <outputRendition>
            font-variant: small-caps;
        </outputRendition>
    </model>
    <model behaviour="inline">
        <outputRendition>
            color: #CC6600;
        </outputRendition>
    </model>
</elementSpec>
```

Task 1: MS Word import and annotation

Using MS Word styles for annotation

- The file `test.docx` is a sample that shows how you can make annotations in MS Word using style so that they get transformed into TEI elements.
- The document `cuento.docx` contains already two types of styles: a paragraph style to create the TEI element `<head>`, and a inline style that will be processed as a TEI element `<title>`.
- If you have MS Word, you can add annotations using the style functionality.
- Open the TEI Publisher app, click on “Playground” and import the MS Word document **after you change its name adding your name or initials**.
- Open the edition of the text, and then open the generated TEI-XML file.

Annotation in TEI Publisher

- Open the TEI Publisher app, click on “Annotation Samples” and import a MS Word document, e.g. `letter.docx` **after you change its name adding your name or initials**.
- Open the document to annotate some of the named entities (e.g. “Thom”, “Elaine”).
- Save the annotations and open the generated TEI-XML file. Then open the register file: Open `eXide` > `Directory` (left side panel) > `tei-publisher` > `data` > `registers` > `persons.xml`

Task 2: creation of a web application

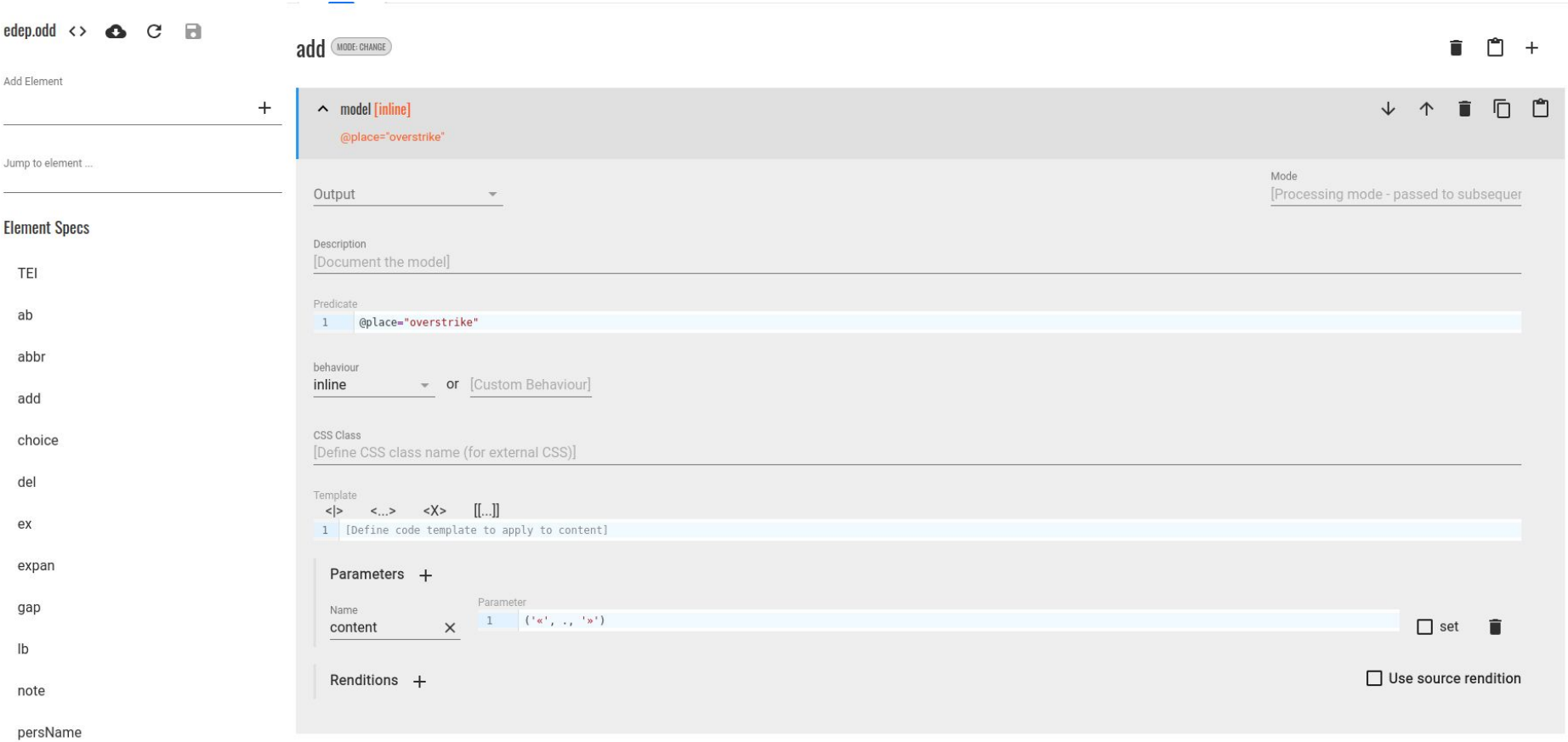
App generation

- Open the TEI Publisher app, make sure that you are logged in, and on the top menu click on `Admin` > `App generator`.
- Fill in the form (wait for instructions)
- Save and open the new app

Customization

- Using the ODD editor (accessible through a URL like: https://tei.dh.unibe.ch/exist/apps/myapp/odd-editor.html?odd=myodd.odd)

```
<elementSpec ident="name" mode="change">
    <model predicate="@type='place'" behaviour="inline">
        <outputRendition>
            font-variant: small-caps;
        </outputRendition>
    </model>
    <model behaviour="inline">
        <outputRendition>
            color: #CC6600;
        </outputRendition>
    </model>
</elementSpec>
```


edep.odd
Add Element
Jump to element ...
Element Specs
TEI
ab
abbr
add
choice
del
ex
expan
gap
lb
note
persName
add
MODE: CHANGE
model [inline]
@place="overstrike"
Output
Mode
[Processing mode - passed to subsequer
Description
[Document the model]
Predicate
1 @place="overstrike"
behaviour
inline
or
[Custom Behaviour]
CSS Class
[Define CSS class name (for external CSS)]
Template
<|> <...> <X> [[...]]
1 [Define code template to apply to content]
Parameters
Name
content
Parameter
1 ('«', ., '»')
set
Renditions
Use source rendition

Behaviours

- alternate
- anchor
- block
- body
- break
- cell
- cit
- document
- figure
- graphic
- heading
- inline
- list
- listItem
- metadata
- note
- omit
- paragraph
- pass-through
- row
- section
- table
- text
- title
- webcomponent

Predicate

- In the `@predicate` attribute we establish the condition that must be met for that model to be applied
- The value of `@predicate` is an XPath expression

XPath

Expression language for navigating the XML document

outputRendition

- Recommended only for editorial aspects that express specific features of the text
- The overall appearance should be determined by the external theme
- You can assign a CSS class through the `cssClass` attribute to be used in external CSS stylesheets

Multiple models

- The first model with a matching predicate or without a predicate will be applied.
- Important: Following models will be ignored even if they match

Parameters

```
<elementSpec ident="date" mode="change">
    <model output="web" predicate="@when" behaviour="alternate">
        <param name="default" value="."/>
        <param name="alternate" value="format-date(@when, '[FNn], [D1o] [MNn], [Y]')"/>
    </model>
    <model predicate="text()" behaviour="inline"/>
</elementSpec>
```

Parameters

- All behaviors take at least one implicit parameter: `content`, that corresponds to the content to be processed
  - If the parameter `content` is not specified, the current node will be processed
- Some behaviours need additional parameters, e.g.:
  - link: `uri`
  - alternate: `default` and `alternate`

modelSequence

```
<elementSpec ident="title" mode="change">
    <modelSequence predicate="parent::titleStmt/parent::fileDesc">
        <model predicate="preceding-sibling::title" behaviour="text">
            <param name="content" value="' - '"/>
        </model>
        <model behaviour="inline"/>
    </modelSequence>
</elementSpec>
```