How is it done? Exploring XML Authoring Features

George Bina / oXygen XML Editor

george@oxygenxml.com

@georgebina

oXygen Users Meetup @ XML Prague 2019

Overview

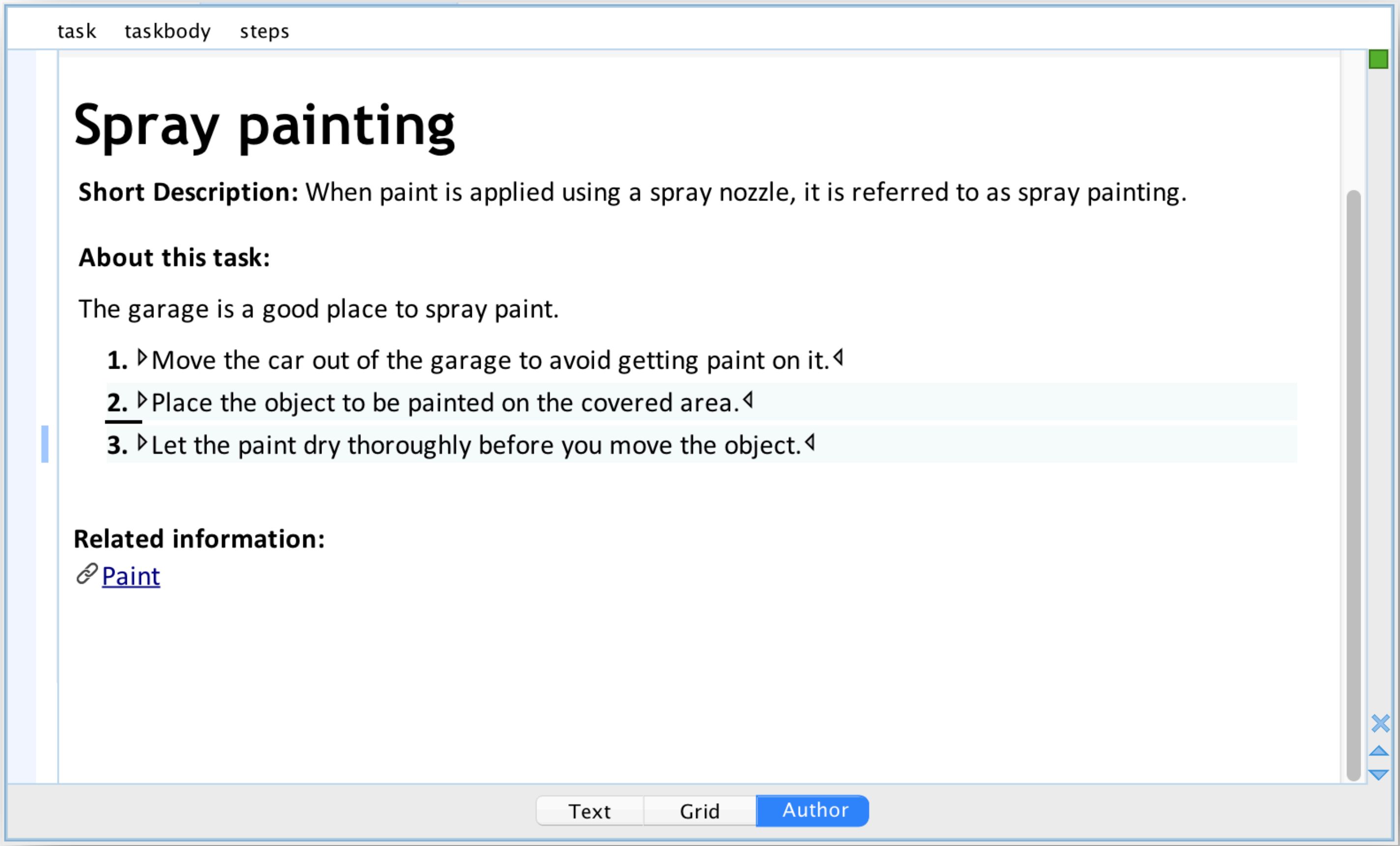
- Inline actions
- Inline hints
- Dynamic views – tabs
- Out-of-flow rendering (present side notes on the side)

Inline actions

Provide actions directly within the document to help the user add new structures.

Samples

- DITA tasks
- DITA troubleshooting
- DITA maps



Spray painting

Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.

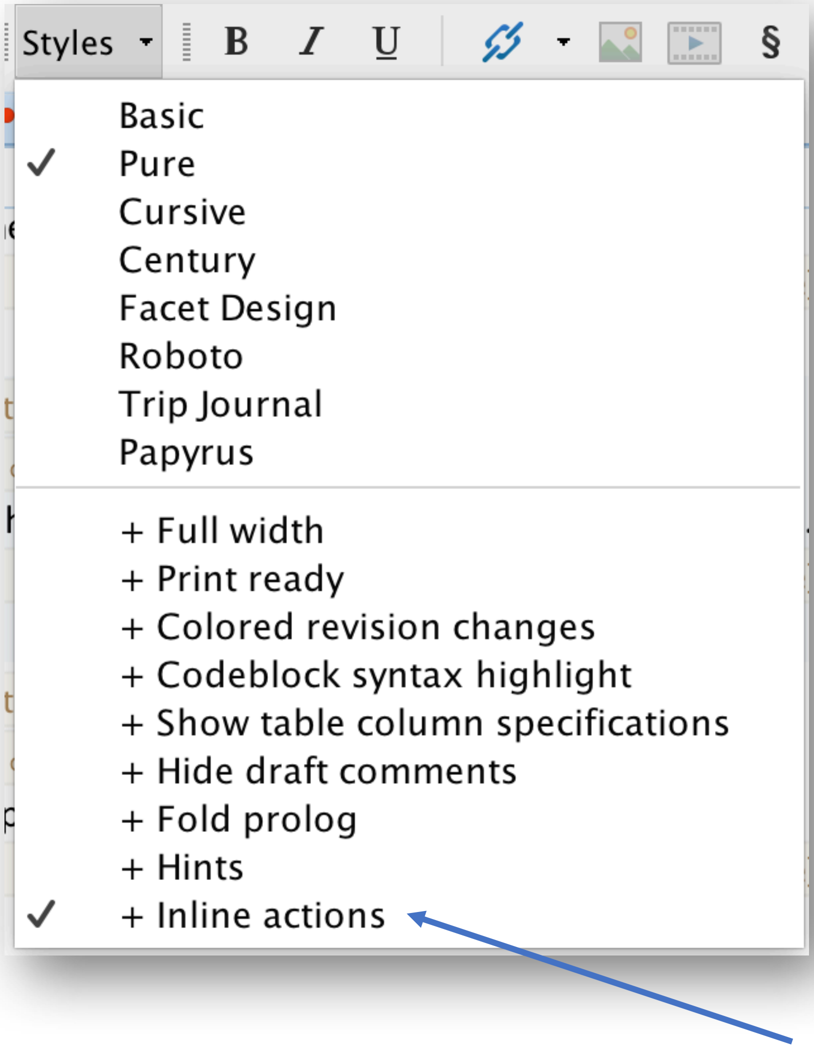
About this task:

The garage is a good place to spray paint.

1. Move the car out of the garage to avoid getting paint on it.
2. Place the object to be painted on the covered area.
3. Let the paint dry thoroughly before you move the object.

Related information:

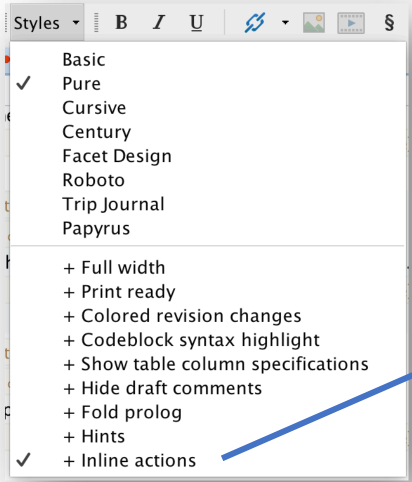
Paint



Styles

Basic
✓ Pure
Cursive
Century
Facet Design
Roboto
Trip Journal
Papyrus

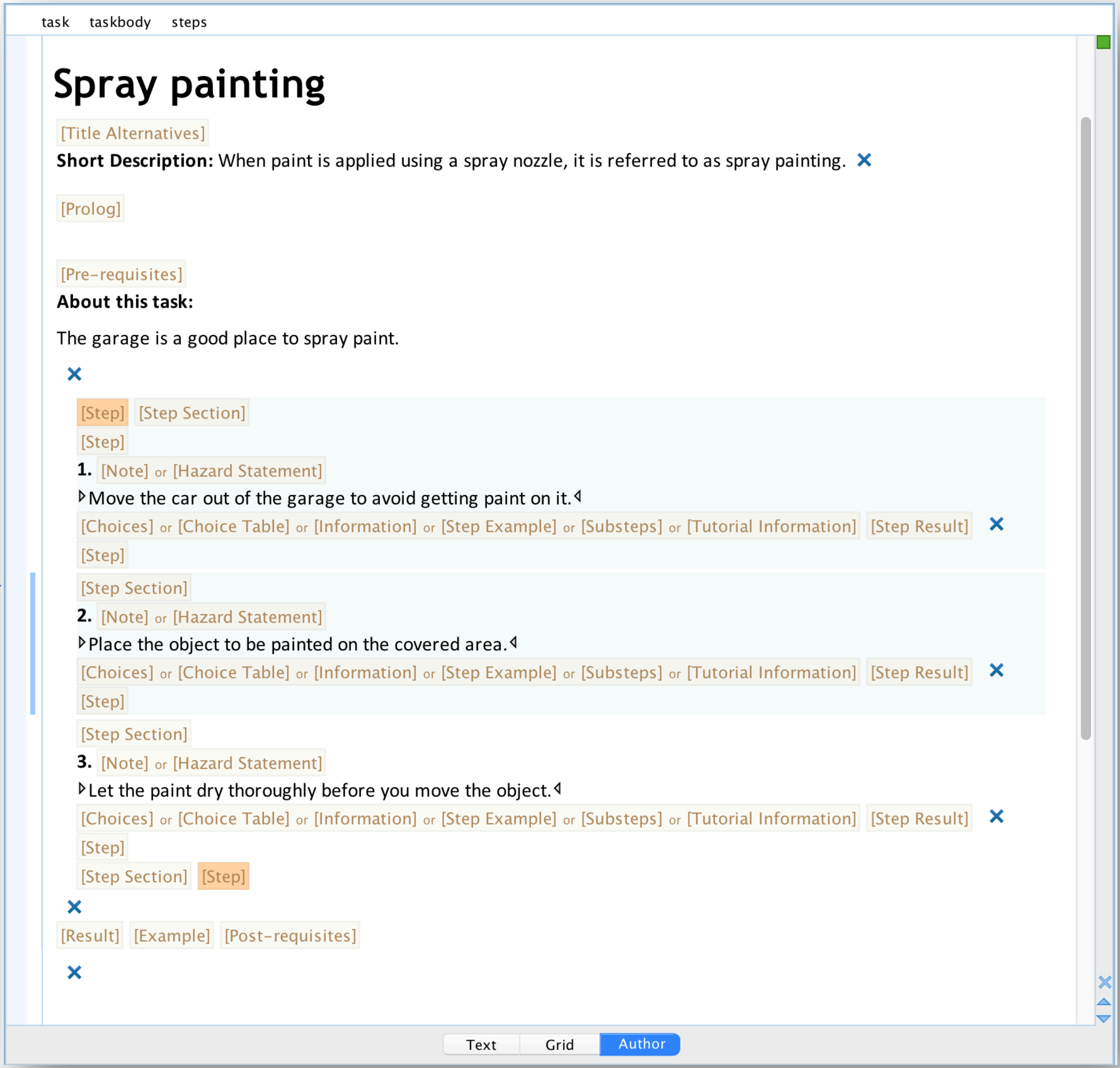
+ Full width
+ Print ready
+ Colored revision changes
+ Codeblock syntax highlight
+ Show table column specifications
+ Hide draft comments
+ Fold prolog
+ Hints
✓ + Inline actions



Styles **B** *I* U §

Basic
✓ Pure
Cursive
Century
Facet Design
Roboto
Trip Journal
Papyrus

+ Full width
+ Print ready
+ Colored revision changes
+ Codeblock syntax highlight
+ Show table column specifications
+ Hide draft comments
+ Fold prolog
+ Hints
✓ + Inline actions



task taskbody steps

Spray painting

[Title Alternatives]

Short Description: When paint is applied using a spray nozzle, it is referred to as spray painting.

[Prolog]

[Pre-requisites]

About this task:

The garage is a good place to spray paint.

[Step] [Step Section]
[Step]
1. [Note] or [Hazard Statement]
Move the car out of the garage to avoid getting paint on it.
[Choices] or [Choice Table] or [Information] or [Step Example] or [Substeps] or [Tutorial Information] [Step Result]
[Step]
[Step Section]
2. [Note] or [Hazard Statement]
Place the object to be painted on the covered area.
[Choices] or [Choice Table] or [Information] or [Step Example] or [Substeps] or [Tutorial Information] [Step Result]
[Step]
[Step Section]
3. [Note] or [Hazard Statement]
Let the paint dry thoroughly before you move the object.
[Choices] or [Choice Table] or [Information] or [Step Example] or [Substeps] or [Tutorial Information] [Step Result]
[Step]
[Step Section] [Step]

[Result] [Example] [Post-requisites]

Text Grid Author

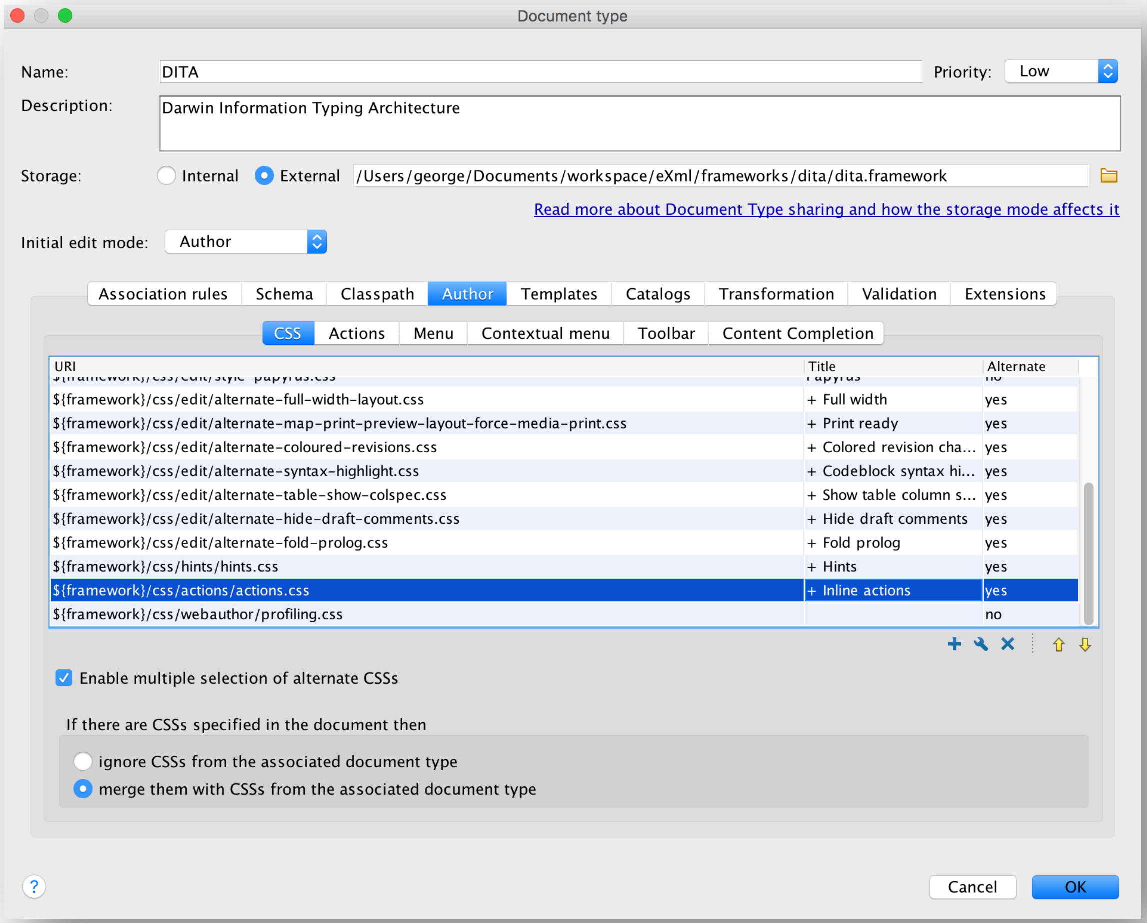
Technology

- Alternate CSS styles for providing different view layers
- Multiple before/after pseudo-element levels
- Inline actions, specified in the CSS
- CSS generated from XSLT based on a configuration XML file

Implementation

- New “+ Inline actions” alternate style
- Use different before/after levels to place different actions without interacting with others
- Generate inline actions in CSS based on a description of the content model, using an XSLT script

Alternate style as an on/off layer



Name: DITA

Priority: Low

Description: Darwin Information Typing Architecture

Storage: Internal [✓ External] /Users/george/Documents/workspace/eXml/frameworks/dita/dita.framework

Read more about Document Type sharing and how the storage mode affects it

Initial edit mode: Author

Association rules | Schema | Classpath | Author | Templates | Catalogs | Transformation | Validation | Extensions
CSS | Actions | Menu | Contextual menu | Toolbar | Content Completion

URI	Title	Alternate
${framework}/css/edit/style-papyrus.css	Papyrus	no
${framework}/css/edit/alternate-full-width-layout.css	+ Full width	yes
${framework}/css/edit/alternate-map-print-preview-layout-force-media-print.css	+ Print ready	yes
${framework}/css/edit/alternate-coloured-revisions.css	+ Colored revision cha...	yes
${framework}/css/edit/alternate-syntax-highlight.css	+ Codeblock syntax hi...	yes
${framework}/css/edit/alternate-table-show-colspec.css	+ Show table column s...	yes
${framework}/css/edit/alternate-hide-draft-comments.css	+ Hide draft comments	yes
${framework}/css/edit/alternate-fold-prolog.css	+ Fold prolog	yes
${framework}/css/hints/hints.css	+ Hints	yes
${framework}/css/actions/actions.css	+ Inline actions	yes
${framework}/css/webauthor/profiling.css		no

[✓] Enable multiple selection of alternate CSSs

If there are CSSs specified in the document then

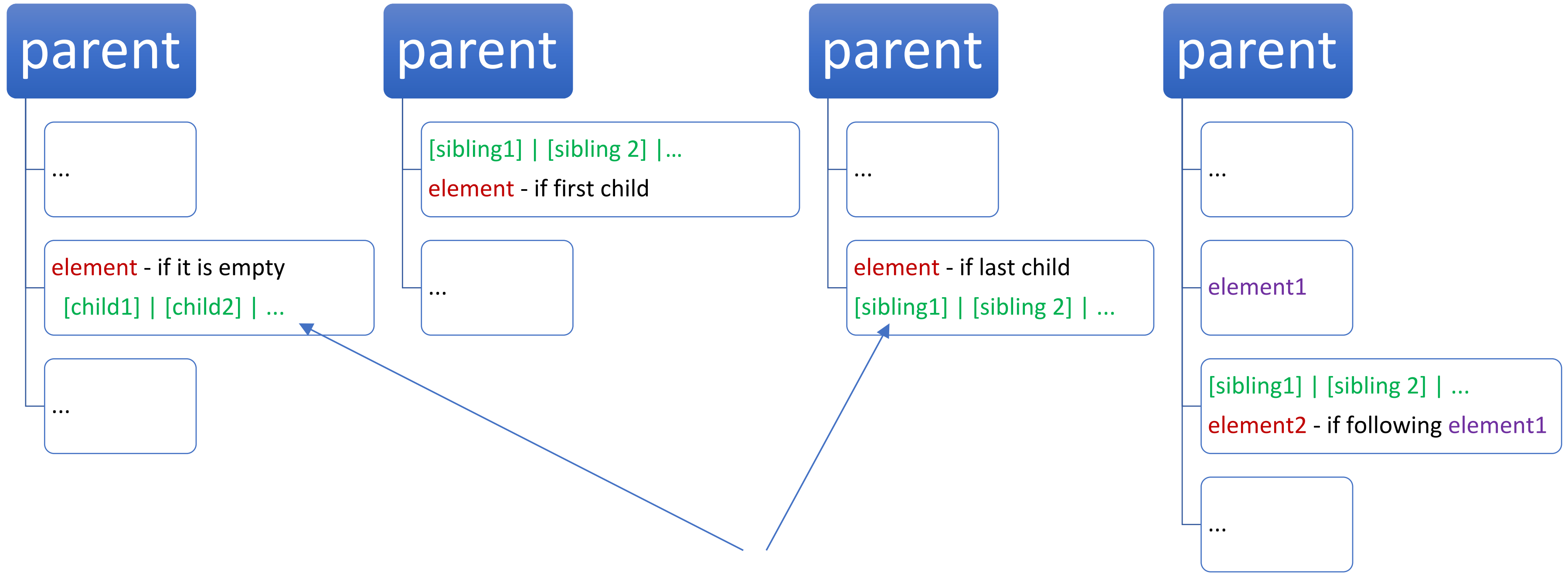
- ignore CSSs from the associated document type
- [✓ merge them with CSSs from the associated document type]

Inline action for [Navigation Title]

```
titlealts:empty:after(311) {
…
content:

    oxy_button(
        color, #B08A5D,
        action,
        oxy_action(
            name, '[Navigation Title]',
            description, 'Insert navigation title',
            operation, 'ro.sync.ecss.extensions.commons.operations.InsertFragmentOperation',
            arg-fragment, '<navtitle>${caret}</navtitle>',
            arg-insertLocation, '.',
            arg-schemaAware, false
        ),
        transparent, true,
        actionContext, element,
        showIcon, true
    )
```

How actions are generated


parent
...
element - if it is empty
[child1] | [child2] | ...
...
parent
[sibling1] | [sibling 2] | ...
element - if first child
...
parent
...
element - if last child
[sibling1] | [sibling 2] | ...
parent
...
element1
[sibling1] | [sibling 2] | ...
element2 - if following element1
...

We need different before/ after layers
to avoid interference in these cases

XML descriptor file

```
<actions>
    <contentModel parent="titlealts">
        <!-- (navtitle?, searchtitle?) -->
        <element name="navtitle" longName="Navigation Title" occurs="?"/>
        <element name="searchtitle" longName="Search Title" occurs="?"/>
    </contentModel>
    …
```

User vs Developer perspective

- Edit the XML descriptor file to specify the content models for different elements
- Apply generateActions.xsl script on an XML descriptor file to generate the corresponding CSS
- Use the generated CSS with XML files to provide inline actions

- Imagine how the CSS may look like
- Develop a prototype manually for a simple case to validate the concept
- Provide a way to generate the CSS in the general case from a configuration file that can be specified by a user

Relates examples

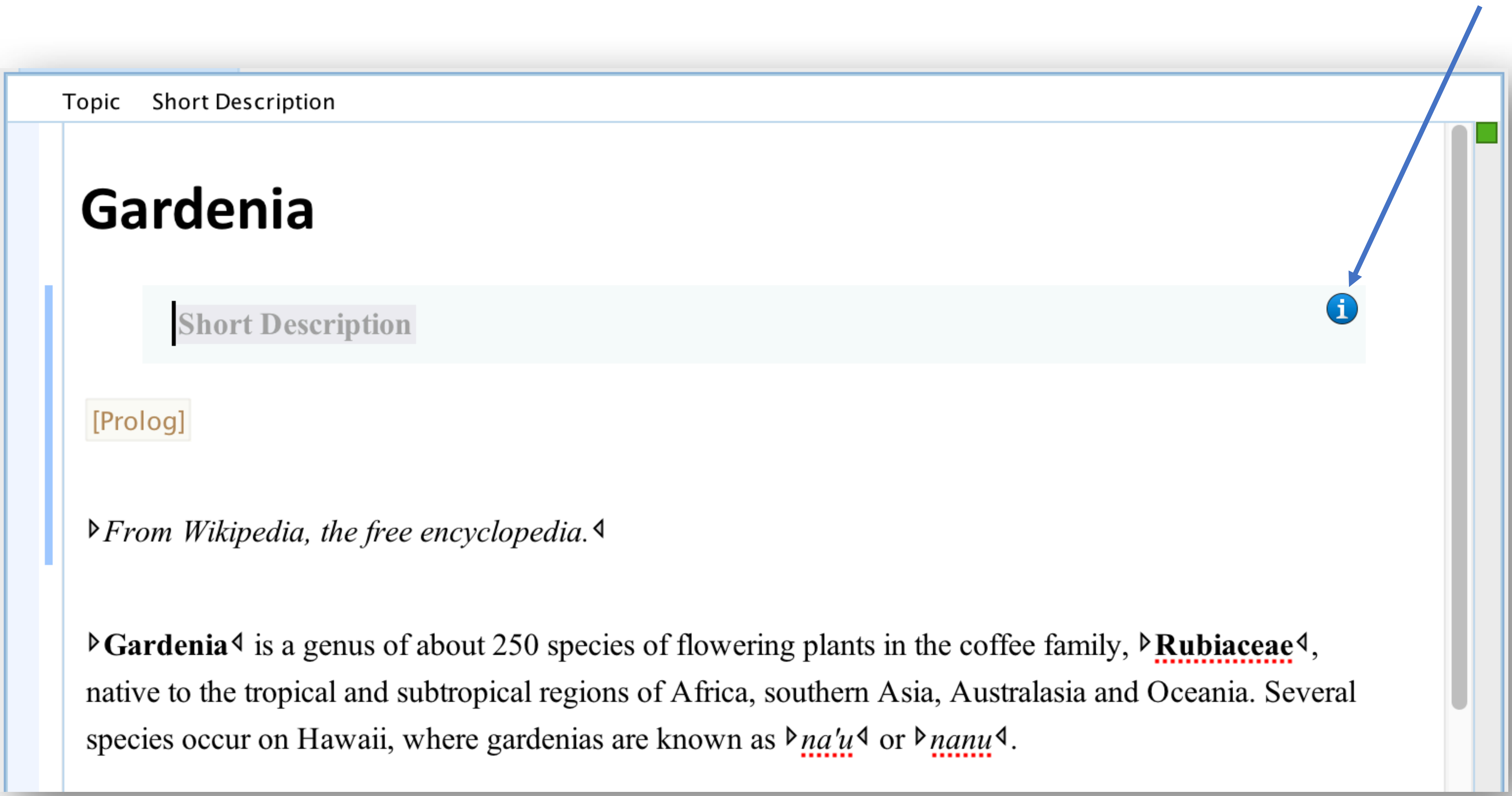
- Inline actions for Lightweight DITA topics
- Saxon configuration files
- DITAVAL filters

Inline hints

Guide the user with inline hints to enable incremental discovery and learning.

Sample

On demand hints in Lightweight DITA topics


Topic Short Description

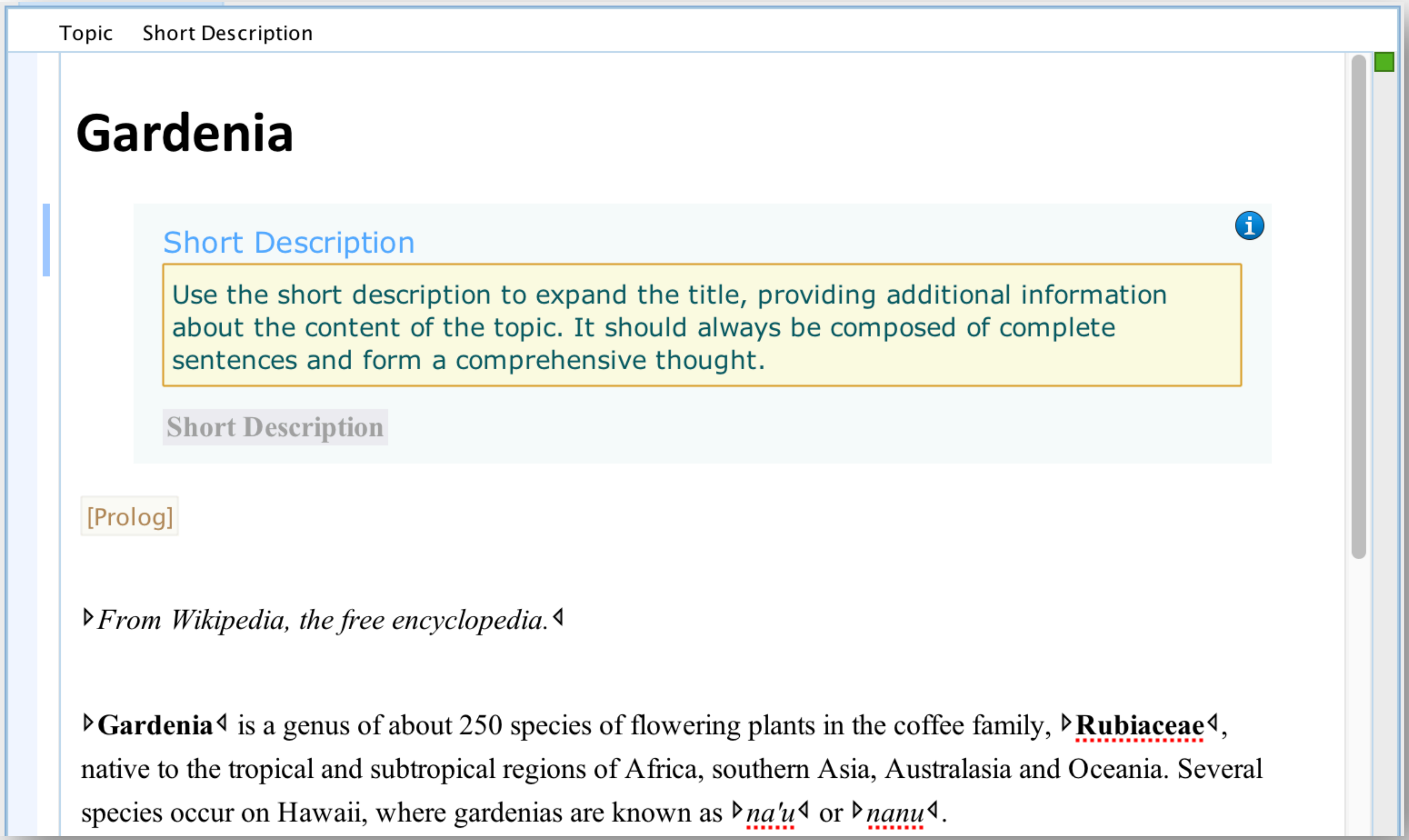
Gardenia

Short Description

[Prolog]

From Wikipedia, the free encyclopedia.

Gardenia is a genus of about 250 species of flowering plants in the coffee family, **Rubiaceae**, native to the tropical and subtropical regions of Africa, southern Asia, Australasia and Oceania. Several species occur on Hawaii, where gardenias are known as *na'u* or *nanu*.



Gardenia

Short Description

Use the short description to expand the title, providing additional information about the content of the topic. It should always be composed of complete sentences and form a comprehensive thought.

Short Description

[Prolog]

From Wikipedia, the free encyclopedia.

Gardenia is a genus of about 250 species of flowering plants in the coffee family, **Rubiaceae**, native to the tropical and subtropical regions of Africa, southern Asia, Australasia and Oceania. Several species occur on Hawaii, where gardenias are known as *na'u* or *nanu*.

Technology

- Form controls
- HTML labels
- Multiple before/after pseudo-element levels
- Inline actions
- Custom pseudo-classes

Implementation

- Define mix-ins in a LESS library to instantiate hints
- Use the mix-ins to declare hints for some elements
- Make sure we have the HTML content for each element we want hints for in the corresponding HTML resource file
- Use the CSS generated from LESS to style the document

Rendering a hint

```
.base-hint(@category, @layer, @borderColor, @bgColor) {
    ...
    &:before(@{layer}) {
        display:block;
        border:1px solid @borderColor;
        background-color: @bgColor;
        …
        content:
            oxy_htmlContent(
                href, 'hints.html',
                id, oxy_concat(@category, '-', oxy_local-name()),
                width, 100%
            );
    }
}
```

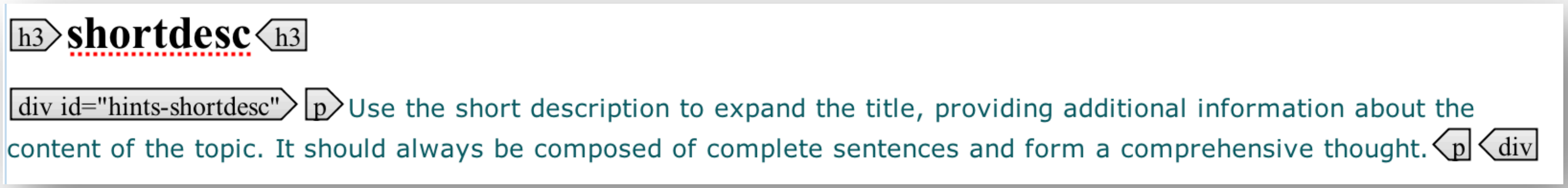
Hint with behaviour

```
.hint(@category: 'hints', @layer:@hintsLayer,
@borderColor:@borderColor, @bgColor:@hintsBgNormal) {
    .infoButton();
    &:-oxy-hints {
        .base-hint(@category, @layer, @borderColor, @bgColor);
    }
}
```

Declare hints

```
/* hints */
topic > title    {.hint();}
shortdesc        {.hint();.markSection("Short Description");}
prolog           {.hint();.markSection("Prolog Information");}
section          {.hint();.markSection("Section");}
fig              {.hint();}
object           {.hint();}
simpletable      {.hint();}
dl               {.hint();}
body             {.hint();.markSection("Content");}
```

Short description hint


h3 shortdesc h3

div id="hints-shortdesc" p Use the short description to expand the title, providing additional information about the content of the topic. It should always be composed of complete sentences and form a comprehensive thought. p div

User vs Developer perspective

- Edit the LESS file to declaratively specify the elements you want hints for
- Add the corresponding entries in the HTML resource file that contain the hints content
- Use the generated CSS from LESS with XML files to provide inline hints (alternatively you can use the LESS file directly)

- Imagine how the CSS may look like
- Develop a prototype manually for a simple case to validate the concept
- Provide a way to generate the CSS in the general case, for example using LESS to just declare the elements you want hints for

Relates examples

- Inline hints layer for DITA topics
- Saxon configuration files

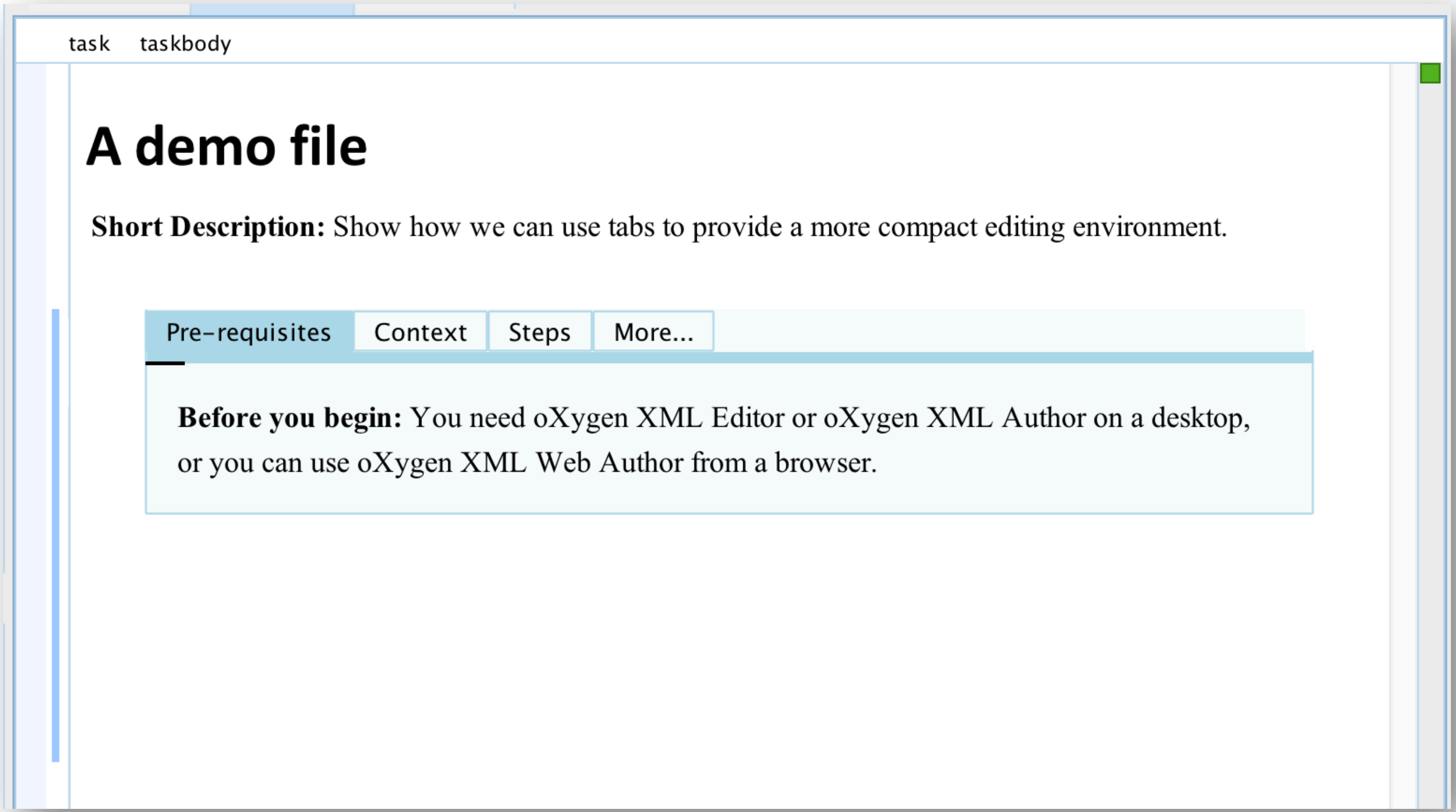
Dynamic views - tabs

Show the content of a document organizing some elements to appear in different tabs, similar to a tabbed pane control

Samples

- Present a DITA task content on multiple tabs
- Present sample code in multiple languages in separate tabs

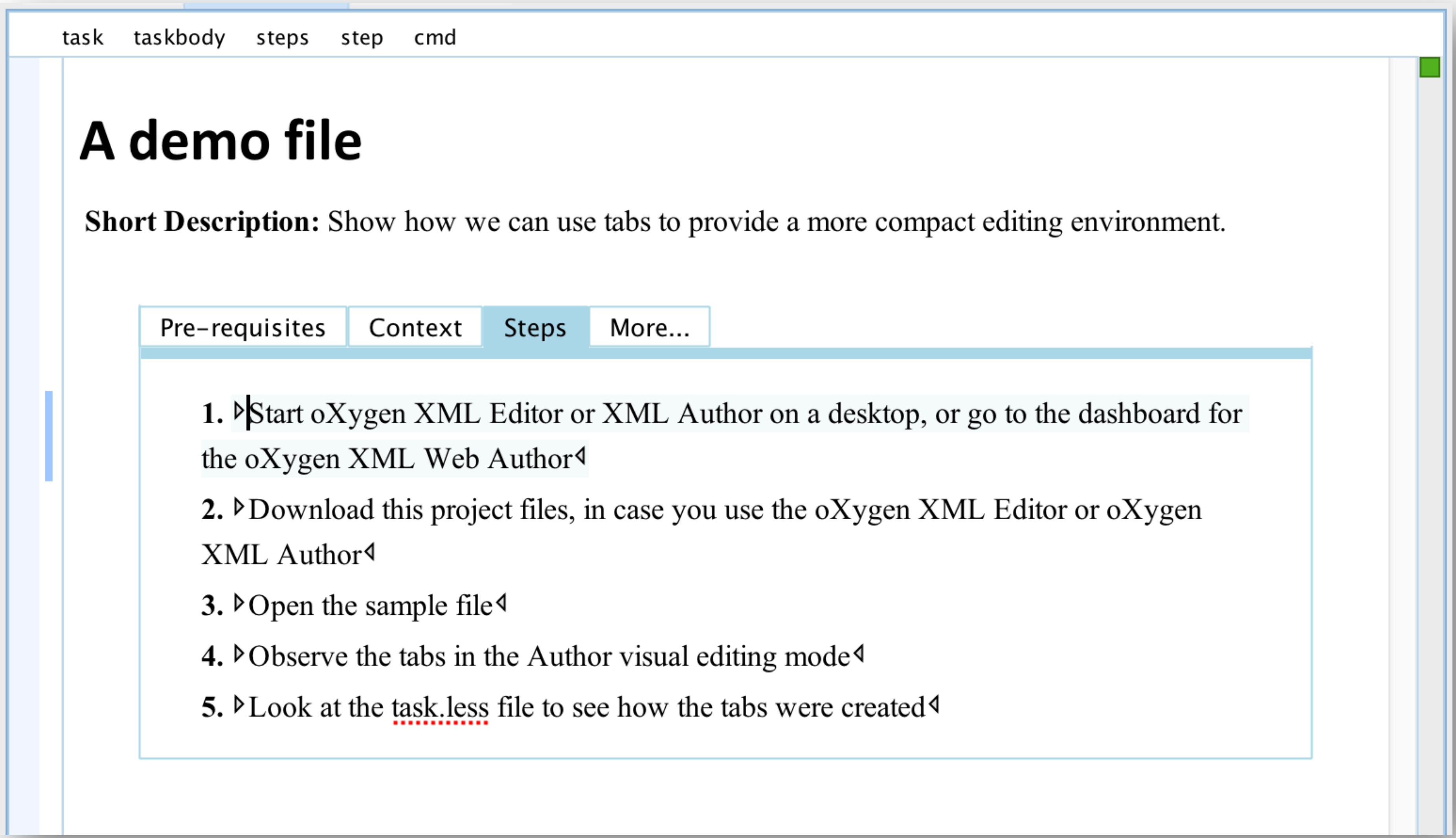
https://www.oxygenxml.com/webapp-demo-aws/app/oxygen.html?url=github%3A%2F%2FgetFileContent%2Foxygenxml%2Fax%2Fmaster%2Ftabs%2Fsample%2Ftask.dita



A demo file

Short Description: Show how we can use tabs to provide a more compact editing environment.

Before you begin: You need oXygen XML Editor or oXygen XML Author on a desktop, or you can use oXygen XML Web Author from a browser.



A demo file

Short Description: Show how we can use tabs to provide a more compact editing environment.

1. Start oXygen XML Editor or XML Author on a desktop, or go to the dashboard for the oXygen XML Web Author
2. Download this project files, in case you use the oXygen XML Editor or oXygen XML Author
3. Open the sample file
4. Observe the tabs in the Author visual editing mode
5. Look at the task.less file to see how the tabs were created

Technology

- Custom pseudo-classes
- Inline actions to set/remove custom pseudo-classes
- Use LESS to generate the CSS

Implementation

- Use a custom pseudo-class for each tab -oxy-visible-N and match on them to control the display property
- Use inline actions as tabs to set the -oxy-visible-N pseudo-class for the corresponding tab
- Use less to provide a declarative approach to define the tabs

Available as Oxygen Authoring eXperience project on GitHub: https://github.com/oxygenxml/ax

prereq and steps appears as 1st and 3rd tabs

```
taskbody:-oxy-visible-1 > * {
  display: none;
}
taskbody:-oxy-visible-1 > prereq {
  display: block;
}

taskbody:-oxy-visible-3 > * {
  display: none;
}
taskbody:-oxy-visible-3 > steps {
  display: block;
}
```

Action to set/remove pseudo-classes

```
content:
        " " oxy_button(
            transparent, true,
            action, oxy_action(
                name, ' Pre-requisites ',
                description, 'Edit Pre-requisites',
                operation,
'ro.sync.ecss.extensions.commons.operations.ChangePseudoClassesOperation',
                arg-removePseudoClassNames,
                    ' -oxy-visible-2 -oxy-visible-3 -oxy-visible-4 ',
                arg-setPseudoClassNames,
                    '-oxy-visible -oxy-visible-1’
            )
        ) " ";
```

Define use of tabs in LESS in declarative way

```
@import "../library/library-tabs.less";

taskbody{
    .tabsWithOthers(
        prereq, context, steps;
        Pre-requisites, Context, Steps;
        'More...'
    );
}

div{
    .tabs(
        "codeblock[outputclass~='language-java']",
        "codeblock[outputclass~='language-cpp']",
        "codeblock[outputclass~='language-xml']";
        Java, CPP, XML;
    );
}
```

User vs Developer perspective

- Edit the LESS file to declaratively specify the elements you want tabs for
- Use the generated CSS from LESS with XML files to provide editing in tabs (alternatively you can use the LESS file directly)

- Imagine how the CSS may look like
- Develop a prototype manually for a simple case to validate the concept
- Provide a way to generate the CSS in the general case, for example using LESS to just declare the elements you want tabs for

Relates examples

- "The Language of Content Strategy" book topics
  https://github.com/oxygenxml/languageBook

- Folding side notes for JTEI
  https://github.com/georgebina/jteiPlus
  https://www.oxygenxml.com/webapp-demo-aws/app/oxygen.html?url=https://github.com/georgebina/jteiPlus/blob/master/samples/JTEI/jtei_8_eide_source/DEMOjtei-8-eide-source.xml

Out-of-flow rendering

Show notes on the side of the editing page, out of the normal in-document flow.

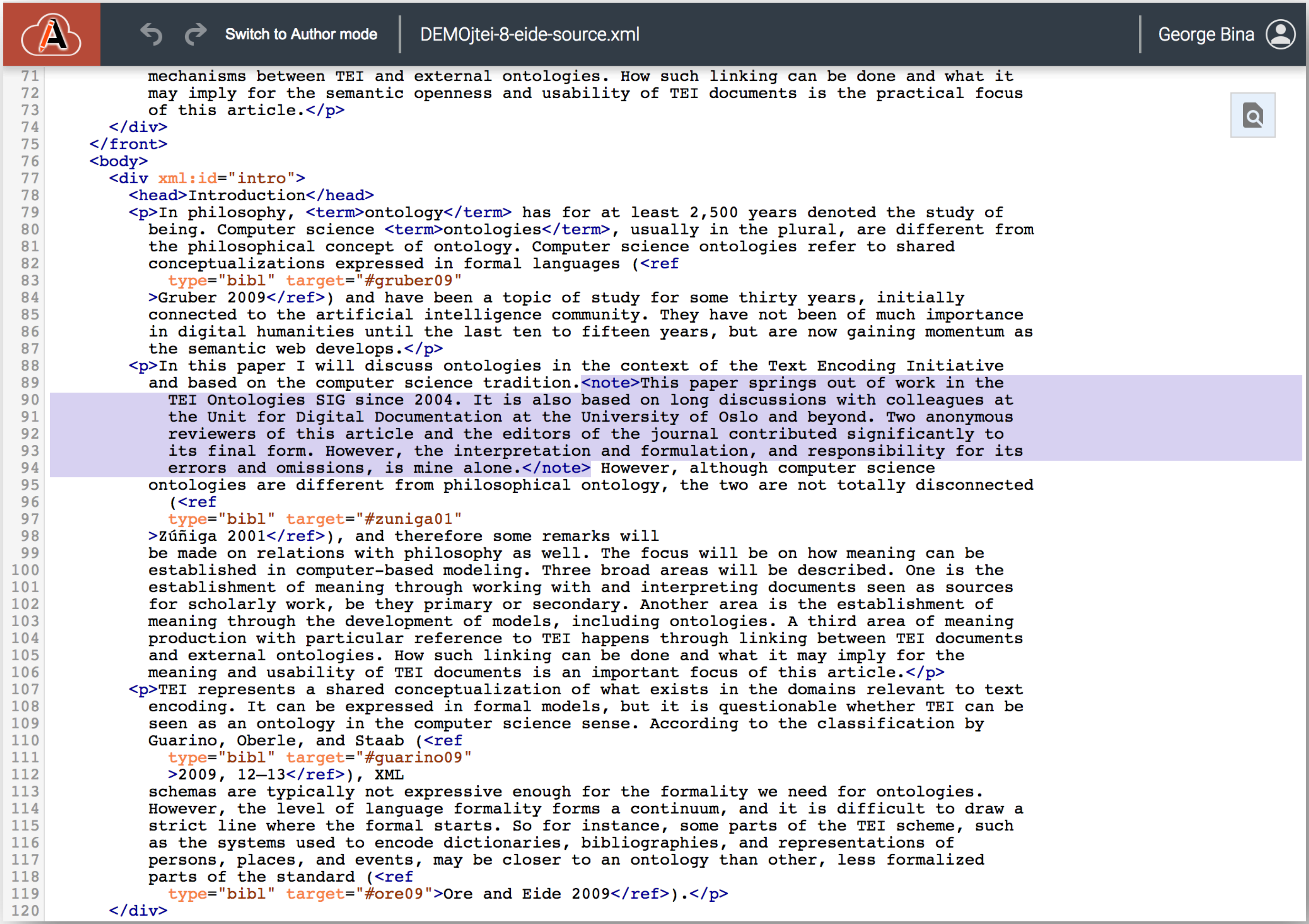
Samples

Journal of TEI article

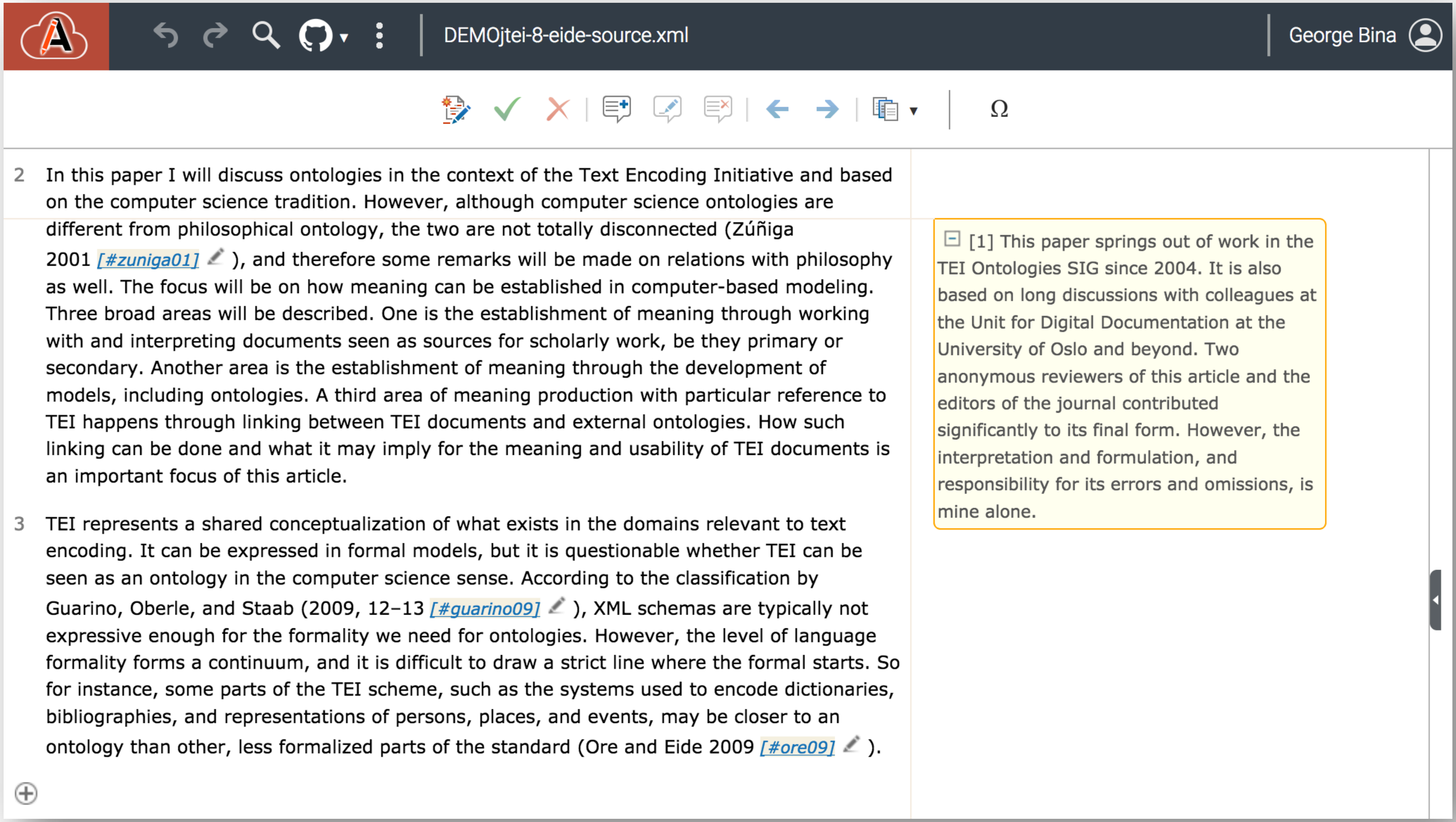
https://www.oxygenxml.com/webapp-demo-aws/app/oxygen.html?url=https://github.com/georgebina/jteiPlus/blob/master/samples/JTEI/jtei_8_eide_source/DEMOjtei-8-eide-source.xml

Available as a GitHub project at https://github.com/georgebina/jteiPlus

How is it done? Exploring XML Authoring Features



```
        mechanisms between TEI and external ontologies. How such linking can be done and what it
        may imply for the semantic openness and usability of TEI documents is the practical focus
        of this article.</p>
    </div>
  </front>
  <body>
    <div xml:id="intro">
      <head>Introduction</head>
      <p>In philosophy, <term>ontology</term> has for at least 2,500 years denoted the study of
        being. Computer science <term>ontologies</term>, usually in the plural, are different from
        the philosophical concept of ontology. Computer science ontologies refer to shared
        conceptualizations expressed in formal languages (<ref
          type="bibl" target="#gruber09"
        >Gruber 2009</ref>) and have been a topic of study for some thirty years, initially
        connected to the artificial intelligence community. They have not been of much importance
        in digital humanities until the last ten to fifteen years, but are now gaining momentum as
        the semantic web develops.</p>
      <p>In this paper I will discuss ontologies in the context of the Text Encoding Initiative
        and based on the computer science tradition.<note>This paper springs out of work in the
          TEI Ontologies SIG since 2004. It is also based on long discussions with colleagues at
          the Unit for Digital Documentation at the University of Oslo and beyond. Two anonymous
          reviewers of this article and the editors of the journal contributed significantly to
          its final form. However, the interpretation and formulation, and responsibility for its
          errors and omissions, is mine alone.</note> However, although computer science
        ontologies are different from philosophical ontology, the two are not totally disconnected
          (<ref
          type="bibl" target="#zuniga01"
        >Zúñiga 2001</ref>), and therefore some remarks will
        be made on relations with philosophy as well. The focus will be on how meaning can be
        established in computer-based modeling. Three broad areas will be described. One is the
        establishment of meaning through working with and interpreting documents seen as sources
        for scholarly work, be they primary or secondary. Another area is the establishment of
        meaning through the development of models, including ontologies. A third area of meaning
        production with particular reference to TEI happens through linking between TEI documents
        and external ontologies. How such linking can be done and what it may imply for the
        meaning and usability of TEI documents is an important focus of this article.</p>
      <p>TEI represents a shared conceptualization of what exists in the domains relevant to text
        encoding. It can be expressed in formal models, but it is questionable whether TEI can be
        seen as an ontology in the computer science sense. According to the classification by
        Guarino, Oberle, and Staab (<ref
          type="bibl" target="#guarino09"
          >2009, 12–13</ref>), XML
        schemas are typically not expressive enough for the formality we need for ontologies.
        However, the level of language formality forms a continuum, and it is difficult to draw a
        strict line where the formal starts. So for instance, some parts of the TEI scheme, such
        as the systems used to encode dictionaries, bibliographies, and representations of
        persons, places, and events, may be closer to an ontology than other, less formalized
        parts of the standard (<ref
          type="bibl" target="#ore09">Ore and Eide 2009</ref>).</p>
    </div>
```



2 In this paper I will discuss ontologies in the context of the Text Encoding Initiative and based on the computer science tradition. However, although computer science ontologies are different from philosophical ontology, the two are not totally disconnected (Zúñiga 2001 [#zuniga01]), and therefore some remarks will be made on relations with philosophy as well. The focus will be on how meaning can be established in computer-based modeling. Three broad areas will be described. One is the establishment of meaning through working with and interpreting documents seen as sources for scholarly work, be they primary or secondary. Another area is the establishment of meaning through the development of models, including ontologies. A third area of meaning production with particular reference to TEI happens through linking between TEI documents and external ontologies. How such linking can be done and what it may imply for the meaning and usability of TEI documents is an important focus of this article.

[1] This paper springs out of work in the TEI Ontologies SIG since 2004. It is also based on long discussions with colleagues at the Unit for Digital Documentation at the University of Oslo and beyond. Two anonymous reviewers of this article and the editors of the journal contributed significantly to its final form. However, the interpretation and formulation, and responsibility for its errors and omissions, is mine alone.

3 TEI represents a shared conceptualization of what exists in the domains relevant to text encoding. It can be expressed in formal models, but it is questionable whether TEI can be seen as an ontology in the computer science sense. According to the classification by Guarino, Oberle, and Staab (2009, 12–13 [#guarino09]), XML schemas are typically not expressive enough for the formality we need for ontologies. However, the level of language formality forms a continuum, and it is difficult to draw a strict line where the formal starts. So for instance, some parts of the TEI scheme, such as the systems used to encode dictionaries, bibliographies, and representations of persons, places, and events, may be closer to an ontology than other, less formalized parts of the standard (Ore and Eide 2009 [#ore09]).

Technology

- Use the support for fixed, absolute and relative positions of blocks
- Multiple before/after pseudo-element levels
- Use z-index to bring to front current note

Implementation

- JTEI additional CSS that renders notes on the side and provides expand/collapse support to show/hide their content

Place notes on the side using absolute position

```
note  {
    margin:2px;
    padding:2px;
    …
    border-radius:0 5px 5px 5px;
    border:1px solid orange;
    …
    position:absolute;
    right:-50%;
    width:45%;
    background-color:#FFFFF0;
    color:#555555;
}
```

Bring current note on top in Web Author

```
note {z-index:1;}

note:hover {z-index:10;}
```

Need additional inspiration?

Here you can find more samples for you to play with and explore how they are built

More samples/frameworks to explore

- Using Google Maps to select a point in order to edit XML coordinates
- Use JavaScript to create custom actions
- Use XSLT to create custom actions (and test them with XSpec)

Thank you!

Questions?

george@oxygenxml.com | @georgebina | http://www.oxygenxml.com