<oXygen/> XML Editor Developer Guide

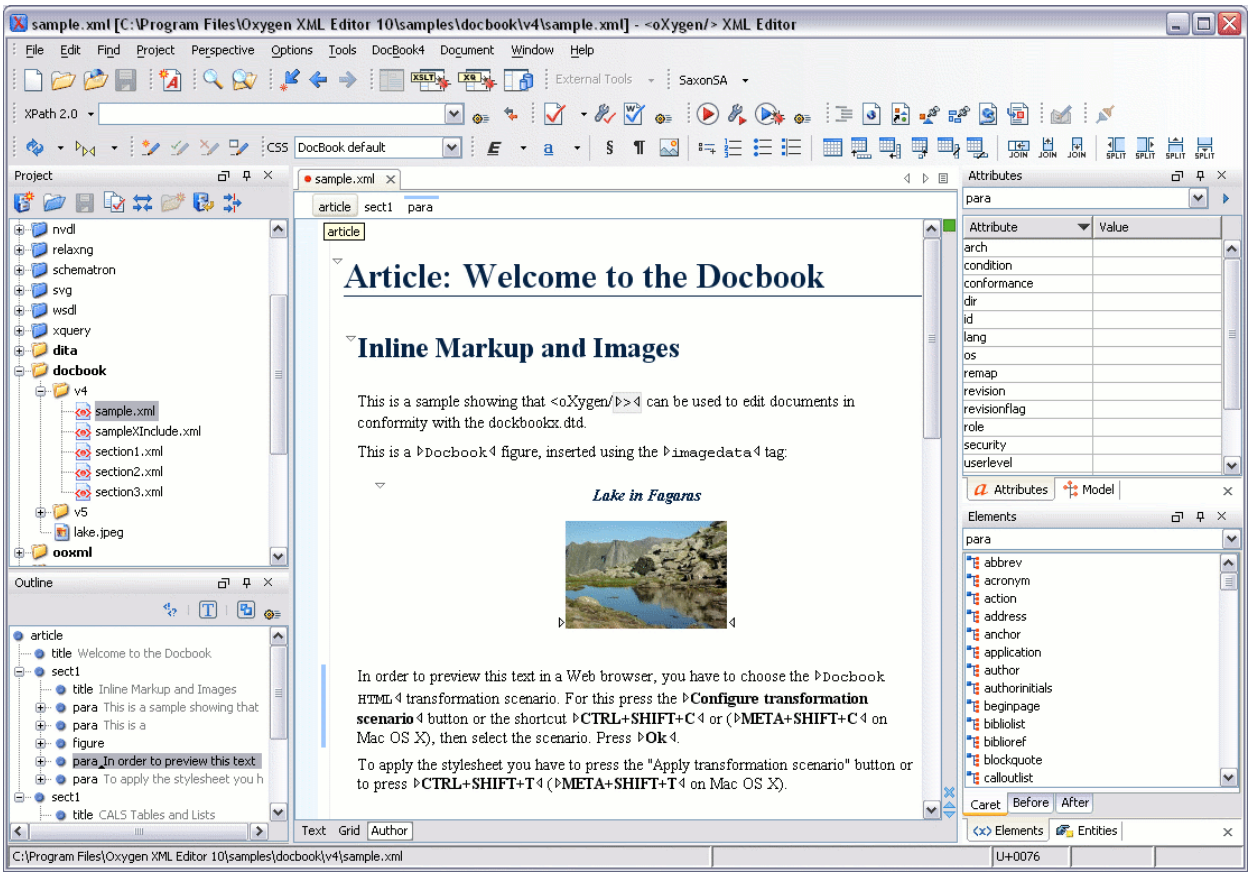
Table of Contents

Introduction

Starting with version 9, <oXygen/> adds extensive support for customization.

The Author mode from <oXygen/> was designed for bridging the gap between the XML source editing and a friendly user interface. The main achievement is the fact that the Author combines the power of the source editing and the intuitive interface of a text editor.

Figure 1. oXygen Author Editor



Although <oXygen/> comes with already configured frameworks for DocBook, DITA, TEI, XHTML, you might need to create a customization of the editor to handle other types of documents. For instance in the case you have a collection of XML document types used to define the structure of the documents that are used in your organisation and you want them visually edited by people who are not experienced in using XML.

There are several ways to customize the editor:

1. Create a CSS file defining styles for the XML elements the user will work with, and create XML files that refer the CSS through an `xml-stylesheet` processing instruction.

2. Fully configure a document type association. This involves putting together the CSS files, the XML schemes, actions, menus, etc, bundling them and distributing an archive. The CSS and the GUI elements are settings of the <oXygen/> Author. The other settings like the templates, catalogs, transformation scenarios are general settings and are enabled whenever the association is active, no matter the editing mode (Text, Grid or Author).

We will discuss both approaches in the following sections.

Simple Customization Tutorial

XML Schema

Let's consider the following XML Schema, `test_report.xsd` defining a report with results of a testing session. The report consists of a title, few lines describing the test suite that was run and a list of test results, each with a name and a boolean value indicating if the test passed or failed.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="report">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="title"/>
                <xs:element ref="description"/>
                <xs:element ref="results"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="description">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="line">
                    <xs:complexType mixed="true">
                        <xs:sequence minOccurs="0"
                            maxOccurs="unbounded">
                            <xs:element name="important"
                              type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="results">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="entry">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="test_name"
                              type="xs:string"/>
                            <xs:element name="passed"
                               type="xs:boolean"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Our use-case is that several users are testing a system and must send report results to a content management system. The Author customization should provide a visual editor for this kind of documents.

Writing the CSS

We have to define a set of rules describing how the XML document is to be rendered into the <oXygen/> Author. This is done using Cascading Style Sheets or CSS on short. CSS is a language used to describe how an HTML or XML document should be formatted by a browser. CSS is widely used in the majority of websites.

Note

For more information regarding CSS, please read the specification http://www.w3.org/Style/CSS/. A tutorial is available here : http://www.w3schools.com/css/css_intro.asp

The elements from an XML document are displayed in the layout as a series of boxes. Some of the boxes contain text and may flow one after the other, from left to right. These are called in-line boxes. There are also other type of boxes that flow one below the other, like paragraphs. These are called block boxes.

For example consider the way a traditional text editor arranges the text. A paragraph is a block, because it contains a vertical list of lines. The lines are also blocks. But any block that contains inline boxes is arranging its children in a horizontal flow. That is why the paragraph lines are also blocks, but the traditional "bold" and "italic" sections are represented as inline boxes.

The CSS allows us to specify that some elements are displayed as tables. In CSS a table is a complex structure and consists of rows and cells. The "table" element must have children that have "table-row" style. Similarly, the "row" elements must contain elements with "table-cell" style.

To make it easy to understand, the following section describes the way each element from the above schema is formatted using a CSS file. Please note that this is just one from an infinite number of possibilities of formatting the content.

report — This element is the root element of the report document. It should be rendered as a box that contains all other elements. To achieve this we set its display type to **block**. Additionally we are setting some margins for it. The CSS rule that matches this element is:

```
report{
    display:block;
    margin:1em;
}
```

title

The title of the report. Usually titles have a larger font. We should use also the **block** display - the next elements will be placed below it, and change its font to double the size of the normal text.

```
title {
    display:block;
    font-size:2em;
}
```

description

This element contains several lines of text describing the report. The lines of text are displayed one below the other, so the description will have the same **block** display. To make it standout we are changing its background.

```
description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}
```

line

A line of text in the description. We do not define a specific aspect for it, just indicating that the display should be **block**.

```
line {
    display:block;
}
```

important

The `important` element defines important text from the description. Because it can be mixed with text, its display property must be set to **inline**. To make it easier to spot, we will emphasize its text.

```
important {
    display:inline;
    font-weight:bold;
}
```

results

The `results` element shows the list of test_names and the result for each one. To make it easier to read, we choose to display it as a **table** with a green border and margins.

```
results{
    display:table;
    margin:2em;
    border:1px solid green;
}
```

entry — An item in the results element. Because we chose the results to be a table, the entry is the row in the table. Thus, the display is **table-row**.

```
entry {
    display:table-row;
}
```

test_name, passed — The name of the individual test, and its result. They are cells in the results table with display set to **table-cell**. Padding and a border are added to emphasize the table grid.

```
test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}

passed{
    font-weight:bold;
}
```

The full content of the CSS file `test_report.css` is:

```
report {
    display:block;
    margin:1em;
}

description {
    display:block;
    background-color:#EEEEFF;
    color:black;
}

line {
    display:block;
}

important {
    display:inline;
    font-weight:bold;
}

title {
    display:block;
    font-size:2em;
}

results{
    display:table;
    margin:2em;
    border:1px solid green;
```

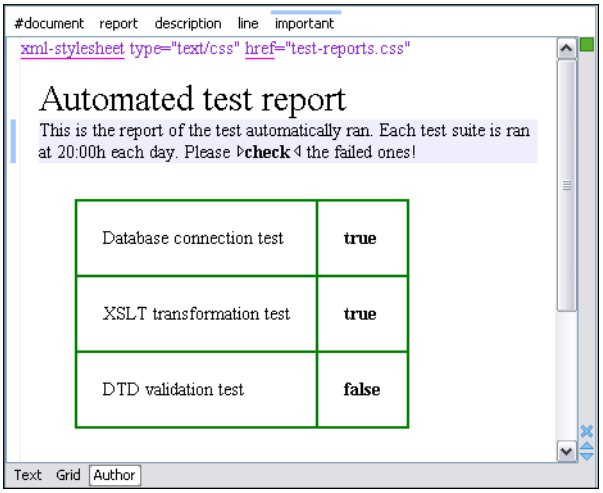
```
}

entry {
    display:table-row;
}

test_name, passed{
    display:table-cell;
    border:1px solid green;
    padding:20px;
}

passed{
    font-weight:bold;
}
```

Figure 2. A report opened in the Author



The XML Instance Template

Now we have the XML Schema and the CSS file. Based on these files, the <oXygen/> Author can help the content author in loading, editing and validating the test reports. We have to create an XML file template, a kind of skeleton, that the users can use as a starting point for creating new test reports.

The template must be generic enough and refer the XML Schema file and the CSS stylesheet. This is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="test_report.xsd">
    <title>Test report title</title>
    <description>
        <line>This is the report
                  <important>description</important>.</line>
    </description>
```

```
    <results>
        <entry>
            <test_name>Sample test1</test_name>
            <passed>true</passed>
        </entry>
        <entry>
            <test_name>Sample test2</test_name>
            <passed>true</passed>
        </entry>
    </results>
</report>
```

The processing instruction `xml-stylesheet` associates the CSS stylesheet to the XML file. The href pseudo attribute contains the URI reference to the stylesheet file. In our case the CSS is in the same directory as the XML file.

The next step is to place the XSD file and the CSS file on a web server and modify the template to use the HTTP URLs, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
        href="http://www.mysite.com/reports/test_report.css"?>
<report xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
        "http://www.mysite.com/reports/test_report.xsd">
    <title>Test report title</title>
    <description>
.......
```

The alternative is to create an archive containing the `test_report.xml`, `test_report.css` and `test_report.xsd` and send it to the content authors.

Advanced Customization Tutorial - Document Type Associations

<oXygen/> Author is highly customizable. Practically you can associate an entire class of documents (grouped logically by some common features like namespace, root element name or filename) to a bundle consisting of a CSS stylesheets, validation schemas, catalog files, templates for new files, transformation scenarios and even custom actions. This is called a **Document Type Association**.

Creating the Basic Association

In this section we will create a **Document Type Association** for a set of documents. As an example we will create a light documentation framework, similar to DocBook and create a complete customization of the Author editor.

You can find the complete files that were used in this tutorial in the Example Files Listings.

First step. XML Schema.

Our documentation framework will be very simple. The documents will be either `articles` or `books`, both composed of `sections`. The `sections` may contain `titles`, `paragraphs`, `figures`, `tables` and other `sections`. To complete the picture, each section will include a `def` element from another namespace.

The first schema file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oxygenxml.com/sample/documentation"
    xmlns:doc="http://www.oxygenxml.com/sample/documentation"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
    elementFormDefault="qualified">

    <xs:import namespace=
    "http://www.oxygenxml.com/sample/documentation/abstracts"
     schemaLocation=
    "abs.xsd"/>
```

The namespace of our documents will be `http://www.oxygenxml.com/sample/documentation`. The namespace of the `def` element is `http://www.oxygenxml.com/sample/documentation/abstracts`.

Now let's define the structure of the sections. They all start with a title, then have the optional `def` element then either a sequence of other sections, or a mixture of paragraphs, images and tables.

```
<xs:element name="book" type="doc:sectionType"/>
<xs:element name="article" type="doc:sectionType"/>
<xs:element name="section" type="doc:sectionType"/>

<xs:complexType name="sectionType">
    <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element ref="abs:def" minOccurs="0"/>
        <xs:choice>
            <xs:sequence>
                <xs:element ref="doc:section" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="doc:para"/>
                <xs:element ref="doc:image"/>
                <xs:element ref="doc:table"/>
            </xs:choice>
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

The paragraph contains text and other styling markup, such as bold (`b`) and italic (`i`) elements.

```
<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b"/>
        <xs:element name="i"/>
    </xs:choice>
</xs:complexType>
```

The `image` element has an attribute with a reference to the file containing image data.

```
<xs:element name="image">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

The `table` contains a header row and then a sequence of rows (`tr` elements) each of them containing the cells. Each cell has the same content as the paragraphs.

```
 <xs:element name="table">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="header">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" maxOccurs="unbounded"
                            type="doc:paragraphType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tr" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td" type="doc:tdType"
                             maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:complexType name="tdType">
    <xs:complexContent>
        <xs:extension base="doc:paragraphType">
            <xs:attribute name="row_span" type="xs:integer"/>
            <xs:attribute name="column_span" type="xs:integer"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

The `def` element is defined as a text only element in the imported schema `abs.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
     "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>
```

Now that we defined our XML data structure, let's start styling it...

Second step. The CSS.

If you read the Simple Customization Tutorial then you already have some basic notions about creating simple styles. Our document contains elements from different namespaces, so we will use CSS Level 3 extensions supported by the <oXygen/> layout engine to associate specific properties with that element.

Note

Please note that the CSS Level 3 is a standard under development, and has not been released yet by the W3C. However, it addresses several important issues like selectors that are namespace aware and values for the CSS properties extracted from the attributes of the XML documents. Although not (yet) conforming with the current CSS standard these are supported by the <oXygen/> Author.

Defining the General Layout.

We are now creating the basic layout of the rendered documents.

Elements that are stacked one on top of the other are: `book`, `article`, `section`, `title`, `figure`, `table`, `image`. These elements are marked as having `block` style for display. Elements that are placed one after the other in a flowing sequence are: `b`, `i`. These will have `inline` display.

```
/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
}
```

Important

Having `block` display children in an `inline` display parent, makes <oXygen/> Author change the style of the parent to `block` display.

Styling the `section` Element.

The title of any section must be bold and smaller than the title of the parent section. To create this effect we have to create a sequence of CSS rules. The `*` operator matches any element, so we can use it to match titles having progressive depths in the document.

```
title{
    font-size: 2.4em;
    font-weight:bold;
}
* * title{
    font-size: 2.0em;
}
```

```
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}
```

Note

CSS rules are combined as follows:

- All the rules that match an element are kept as a list. The more specific the rule is, the further it will be placed to the end of the list.
- If there is no difference in the specificity of the rules, they are placed in the list in the same order as they appear in the CSS document.
- The list is then iterated, and all the properties from the rules are collected, overwriting the already collected values from the previous rules. That is why the font-size is changed depending on the depth of the element, while the font-weight property remains unchanged - no other rule is overwriting it.

It's useful to have before the title a constant text, indicating that it refers to a section. This text can include also the current section number. To achieve we have to use the `:before` and `:after` pseudo elements, plus the CSS counters.

We declare a counter named `sect` for each `book` or `article`. The counter is set to zero at the beginning of each such element:

```
book,
article{
    counter-reset:sect;
}
```

The `sect` counter is incremented with each `section`, that is the a direct child of a `book` or an `article` element.

```
book > section,
article > section{
    counter-increment:sect;
}
```

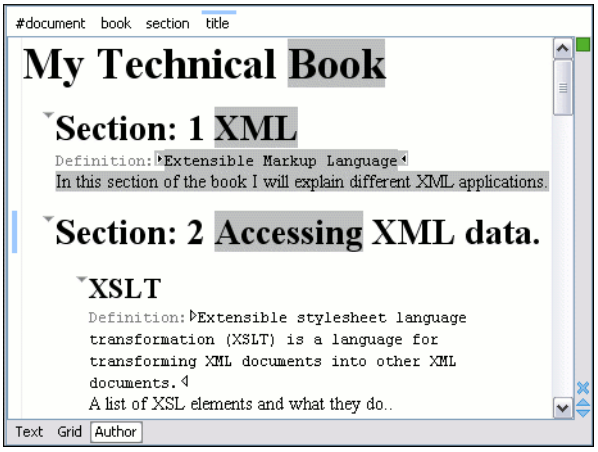
The "static" text that will prefix the section title is composed of the constant "Section ", followed by the decimal value of the `sect` counter and a dot.

```
book > section > title:before,
article > section > title:before{
    content: "Section " counter(sect) ". ";
}
```

To make the documents easy to read, we will add a margin to the sections. In this way the higher nesting level, the larger the left side indent. The margin is expressed relatively to the parent bounds:

```
section{
    margin-left:1em;
    margin-top:1em;
}
```

Figure 3. A sample of nested sections and their titles.



In the above screenshot you can see a sample XML document rendered by our CSS. The selection "avoids" the text that is generated by the CSS "content" property. This happens because the CSS generated text is not present in the XML document and is just a visual aid.

Styling the `table` Element.

There are standard CSS properties used to indicate what elements are tables, table rows and table cells. What CSS is missing is the possibility to indicate the cell spanning. <oXygen/> Author offers support for adding an extension to solve this problem. This will be presented in the next chapters.

The table in this example is a simple one. The header must be formatted in a different way than the ordinary rows, so it will have a background color.

```
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
```

```
  padding:1em;
}
```

Note

Children elements with `block` or `table-caption` display placed at the beginning or the end of an element displayed as a `table`, will be grouped and presented as blocks at the top or the bottom of the table.

Note

Mixing elements having `table-cell`, `table-group`, `table-row`, etc.. display type with others that have `block` or `inline` display or with text content breaks the layout of the table. In such cases the table is shown as a `block`.

Note

Having child elements that do not have `table-cell` or `table` display in a parent with `table-row` display breaks the table layout. In this case the `table` display is supported for the children of the `table-row` element in order to allow sub-tables in the parent table.

Note

<oXygen/> Author can automatically detect the spanning of a cell, without the need to write a Java extension for this.

This happens if the span of the cell element is specified using the **colspan** and **rowspan** attributes, just like in HTML, or **cols** and **rows** attributes.

For instance, the following XML code:

```
  <table>
        <tr>
            <td>Cell 1.1</td>
            <td>Cell 1.2</td>
            <td>Cell 1.3</td>
        </tr>
        <tr>
            <td>Cell 2.1</td>
            <td colspan="2" rowspan="2">
            Cell spanning 2 rows and 2 columns.
            </td>
        </tr>
        <tr><td>Cell 3.1</td></tr>
  </table>
```

using the CSS:

```
table{
   display: table;
}
tr{
   display: table-row;
}
td{
```

```
    display: table-cell;
}
```

is rendered correctly:

Table 1. Built-in Cell Spanning

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell spanning 2 rows and 2 columns	
Cell 3.1		

Because in our schema the `td` tag has the attributes **row_span** and **column_span** that are not automatically recognized by <oXygen/> Author, we will implement a Java extension which will provide information about the cell spanning. See the section Configuring a Table Cell Span Provider.

Because the column widths are specified by the attributes **width** of the elements `customcol` that are not automatically recognized by <oXygen/> Author, it is necessary to implement a Java extension which will provide information about the column widths. See the section Configuring a Table Column Width Provider.

Styling the Inline Elements.

The "bold" style is obtained by using the `font-weight` CSS property with the value `bold`, while the "italic" style is specified by the `font-style` property:

```
b {
    font-weight:bold;
}

i {
    font-style:italic;
}
```

Styling Elements from other Namespace

In the CSS Level 1, 2, and 2.1 there is no way to specify if an element X from the namespace Y should be presented differently from the element X from the namespace Z. In the upcoming CSS Level 3, it is possible to differentiate elements by their namespaces. <oXygen/> Author supports this CSS Level 3 functionality. For more information see the Namespace Selectors section.

To match our `def` element we will declare its namespace, bind it to the *abs* prefix, and then write a CSS rule:

```
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}
```

Styling images

The CSS 2.1 does not specify how an element can be rendered as an image. To overpass this limitation, <oXygen/> Author supports a CSS Level 3 extension allowing to load image data from an URL. The URL of the image must be specified by one of the element attributes and it is resolved through the catalogs specified in <oXygen/>.

Note

<oXygen/> Author recognizes the following image file formats: JPEG, GIF, PNG and SVG. The oXygen Author for Eclipse does not render the SVG files.

```
image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

Our `image` element has the required attribute `href` of type `xs:anyURI`. The `href` attribute contains an image location so the rendered content is obtained by using the function:

```
attr(href, url)
```

Important

The first argument is the name of the attribute pointing to the image file. The second argument of the `attr` function specifies the type of the content. If the type has the `url` value, then <oXygen/> identifies the content as being an image. If the type is missing, then the content will be the text representing the attribute value.

Important

<oXygen/> Author handles both absolute and relative specified URLs. If the image has an *absolute* URL location (e.g: "http://www.oasis-open.org/images/standards/oasis_standard.jpg") then it is loaded directly from this location. If the image URL is *relative* specified to the XML document (e.g: "images/my_screenshot.jpg") then the location is obtained by adding this value to the location of the edited XML document.

An image can also be referenced by the name of a DTD entity which specifies the location of the image file. For example if the document declares an entity **graphic** which points to a JPEG image file:

```
<!ENTITY graphic SYSTEM "depo/keyboard_shortcut.jpg" NDATA JPEG>
```

and the image is referenced in the XML document by specifying the name of the entity as the value of an attribute:

```
<mediaobject>
    <imageobject>
        <imagedata entityref="graphic" scale="50"/>
    </imageobject>
</mediaobject>
```

The CSS should use the functions **url**, **attr** and **unparsed-entity-uri** for displaying the image in the Author mode:

Note

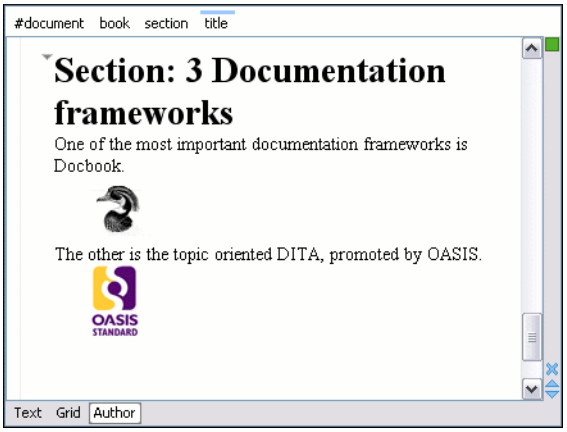
Note that the scale attribute of the imagedata element will be considered without the need of a CSS customization and the image will be scaled accordingly.

```
imagedata[entityref]{
    content: url(unparsed-entity-uri(attr(entityref)));
}
```

To take into account the value of the `width` attribute of the `imagedata` and use it for resizing the image, the CSS can define the following rule:

```
imagedata[width]{
  width:attr(width, length);
}
```

Figure 4. Samples of images in Author

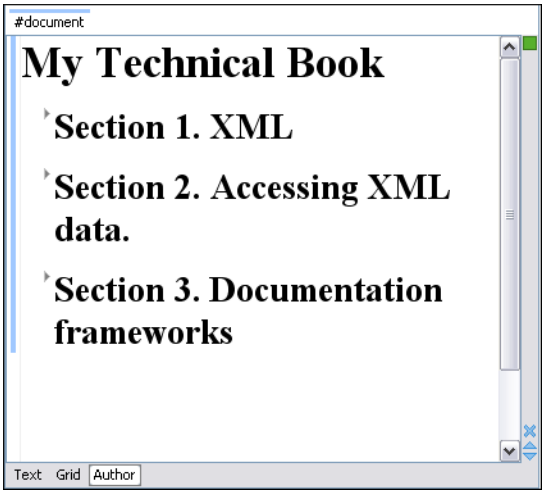


Marking elements as foldable

You can specify what elements are collapsible. The collapsible elements are rendered having a small triangle icon in the top left corner. Clicking on this icon hides or shows the children of the element. In our case, we will mark the `section` elements as foldable. We will leave only the `title` child elements visible.

```
section{
    foldable:true;
    not-foldable-child: title;
}
```

Figure 5. Folded Sections



Marking elements as links

You can specify what elements are links. The text content specified in the `:before` pseudo element will be underlined. When hovering the mouse over that content the mouse pointer will change to indicate that it can follow the link. Clicking on a link will result in the referred resource being opened in an editor. In our case we will mark the `link` elements as links with the `href` attribute indicating the referred location.

```
link[href]:before{
    display:inline;
    link:attr(href);
    content: "Click to open: " attr(href);
}
```

Note

If you plan to use IDs as references for links, the value of the link property should start with a sharp sign(#). This will ensure that the default link target reference finder implementation will work and clicking on the link will send you to the indicated location in the document. For more details about the link target reference finder read the section Configuring a Link target reference finder.

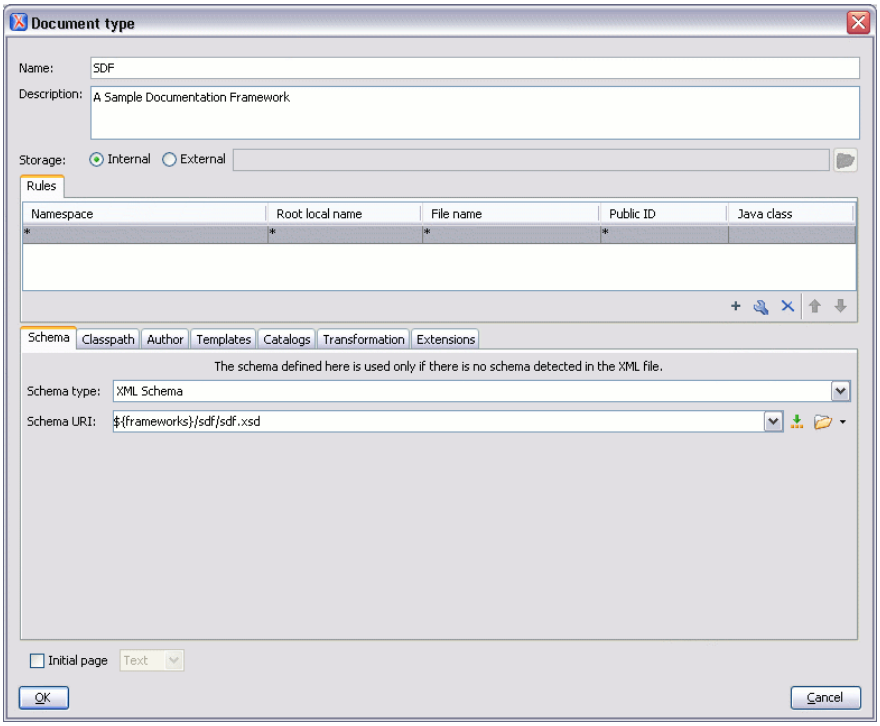
Example 1. IDs as references for links

```
link[linkend]:before{
    display:inline;
    link: "#" attr(linkend);
    content: "Click to open: " attr(linkend);
}
```

Third Step. The Association.

Now that we have the XML Schema and the CSS stylesheet for the documents we intend to edit, we can proceed to create a distributable framework package for our content authors.

Figure 6. The Document Type Dialog



Organizing the Framework Files

First create a new folder called `sdf` (from "Simple Documentation Framework") in `{oXygen_installation_directory}/frameworks`. We will use this folder to store all files related to our documentation framework. Let's organise it a bit, by creating the following folder structure:

```
oxygen
  frameworks
     sdf
       schema
       css
```

Important

The `frameworks` directory is the container where all the oXygen framework customizations are located.

Each subdirectory contains files related to a specific type of XML documents: schemas, catalogs, stylesheets, CSS files, etc.

Distributing a framework means delivering a framework directory.

Important

We assume you have the right to create files and folder inside the oXygen installation directory. If you do not have this right, you will have to install another copy of the program in a folder you have access to, the home directory for instance, or your desktop. You can download the "all platforms" distribution from the oXygen website and extract it in the chosen folder.

To test your framework distribution you will need to copy it in the `frameworks` directory of the newly installed application and start oXygen by running the provided start-up script files.

We should copy the created schema files `abs.xsd` and `sdf.xsd`, `sdf.xsd` being the master schema, to the `schema` directory and the CSS file `sdf.css` to the `css` directory.

Association Rules

We must specify when oXygen should use the files created in the previous section by creating a document type association. Open the Document Type dialog by following the procedure:

1. Open the Options Dialog, and select the Document Types Association option pane.
2. Select the **Developer** user role from the **User role** combo box at the top of the dialog. This is important, because it will allow us to save the document type association in a file on disk, instead of <oXygen/> options.
3. Click on the **New** button.

In the displayed dialog, fill in the following data:

Name — Enter **SDF** - This is the name of the document type.

Description — Enter **Simple Documentation Framework** - This is a short description helping the other users understand the purpose of the Document Type.

Storage — The storage refers to the place where the Document Type settings are stored. Internal means the Document Types are stored in the default <oXygen/> preferences file. Since we want to share the Document Type to other users, we must select External, and choose a file.

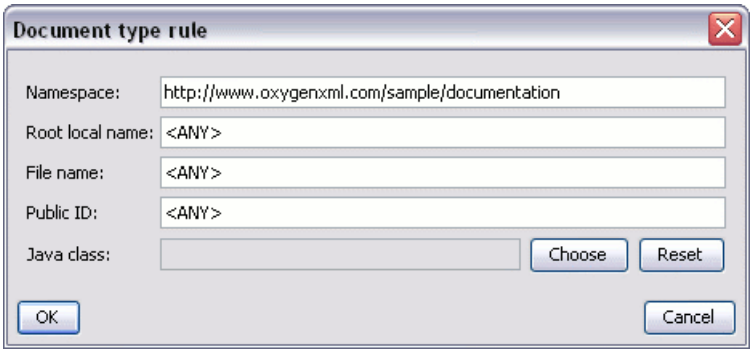
The file must be in the `{oXygen_installation_directory}/frameworks/sdf` directory. A possible location is **/Users/{user_name}/Desktop/oxygen/frameworks/sdf/sdf.framework**. The framework directory structure will be:

```
oxygen
  frameworks
     sdf
       sdf.framework
       schema
         sdf.xsd
       css
         sdf.css
```

Rules — If a document opened in <oXygen/> matches one of the rules defined for the Document Type, then it is activated.

Press the + Add button from the Rules section. Using the newly displayed dialog, we add a new rule that matches documents with the root from the namespace: `http://www.oxygenxml.com/sample/documentation`. The root name, file name or PublicID are not relevant.

Figure 7. Editing a rule



A document matches a rule when it fulfills the conditions imposed by each field of the rule:

Namespace	the namespace of the root element declared in the XML documents of the current document type. A value of ANY_VALUE matches any namespace in an XML document. Value may contain wildcards(*, ?) and editor variables. Multiple values separated by comma(,) are accepted.
Root local name	The local name of the root element of the XML documents of the current document type. A value of ANY_VALUE matches any local name of the root element. Value may contain wildcards(*, ?) and editor variables. Multiple values separated by comma(,) are accepted.
File name	The file name of the XML documents of the current document type. A value of ANY_VALUE matches any file name. Value may contain wildcards(*, ?) and editor variables. Multiple values separated by comma(,) are accepted.
Public ID	The public ID of the XML documents of the current document type (for a document validated against a DTD). A value of ANY_VALUE matches any public ID. Value may contain wildcards(*, ?) and editor variables. Multiple values separated by comma(,) are accepted.
Java class	The full name of a Java class that has access to all root element attributes and the above 4 values in order to decide if the document matches the rule.

Java API: Rules implemented in Java

An alternative to the rule we defined for our association is to write the entire logic in Java.

1. Create a new Java project, in your IDE.

   Create the `lib` directory in the Java project directory and copy there the `oxygen.jar` file from the `{oXygen_installation_directory}/lib`. The `oxygen.jar` contains the Java interfaces we have to implement and the available Author API needed to access its features.

2. Create the class `simple.documentation.framework.CustomRule`. This class must implement the `ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher` interface.

   The interface defines two methods: `matches`, and `getDescription`.

1. The `matches` method is the one that is invoked when the edited document must be checked against the document type association. It takes as arguments the root local name, its namespace, the document location URI, the PublicID and the root element attributes. It must return `true` when the document matches the association.

2. The `getDescription` method returns a description of the rule.

Here is the implementation of these two methods. The implementation of `matches` is just a Java equivalent of the rule we defined earlier.

```
public boolean matches(
  String systemID,
  String rootNamespace,
  String rootLocalName,
  String doctypePublicID,
  Attributes rootAttributes) {

 return "http://www.oxygenxml.com/sample/documentation"
  .equals(rootNamespace);
}

public String getDescription() {
 return "Checks if the current Document Type Association"
 + " is matching the document.";
}
```

The complete source code is found in the Example Files Listings, the Java Files section.

3. Package the compiled class into a *jar* file. Here is an example of an ANT script that packages the `classes` directory content into a *jar* archive named `sdf.jar`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="dist">
    <target name="dist">
   <jar destfile="sdf.jar" basedir="classes">
    <fileset dir="classes">
     <include name="**/*"/>
     </fileset>
   </jar>
    </target>
</project>
```

4. Copy the `sdf.jar` file into the `frameworks/sdf` directory.

5. Add the `sdf.jar` to the Author classpath. To do this select **SDF** Document Type from the **Document Type Association** options page and press the Edit button.

   Select the Classpath tab in the lower part of the dialog.

   Press the + Add button . In the displayed dialog enter the location of the jar file, relative to the <oXygen/> `frameworks` directory. If you are in the process of developing the extension actions you can also specify a path to a directory which holds compiled Java classes.

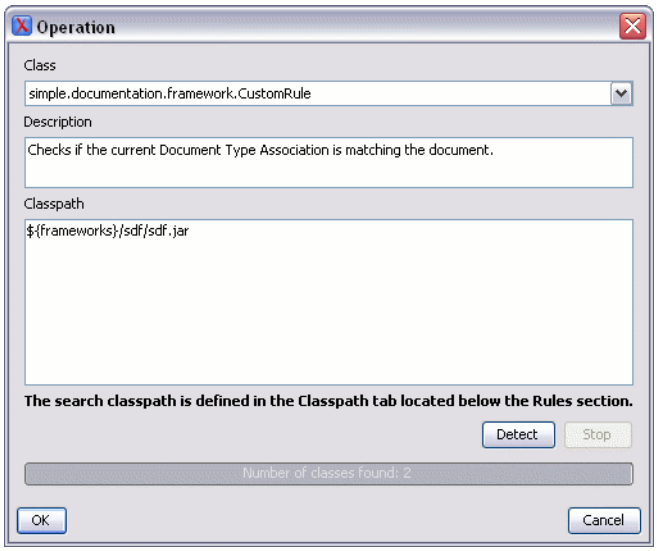
Figure 8. Adding a classpath entry

6. Clear the rules we defined before by using the − Remove button.

   Press the + Add button from the Rules section.

   Press the Choose button that follows the Java class value. The following dialog is displayed:

Figure 9. Selecting a Java association rule.



To test the association, open the `sdf.xml` sample and validate it.

Deciding the initial page

You can decide to impose an initial page for opening files which match the association rules. For example if the files are usually edited in the *Author* page you can set it as the initial page for files matching your rules.

Schema Settings

In the dialog for editing the Document Type properties, in the bottom section there are a series of tabs. The first one refers to the schema that is used for validation of the documents that match the defined association **Rules**.

Important

If the document refers a schema, using for instance a `DOCTYPE` declaration or a `xsi:schemaLocation` attribute, the schema from the document type association will not be used when validating.

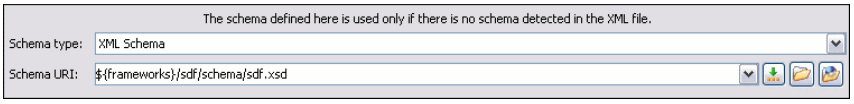
Schema Type — Select from the combo box the value **XML Schema**.

Schema URI — Enter the value `${frameworks}/sdf/schema/sdf.xsd`. We should use the `${frameworks}` editor variable in the schema URI path instead of a full path in order to be valid for different <oXygen/> installations.

Important

The `${frameworks}` variable is expanded at the validation time into the absolute location of the directory containing the frameworks.

Figure 10. The Schema panel



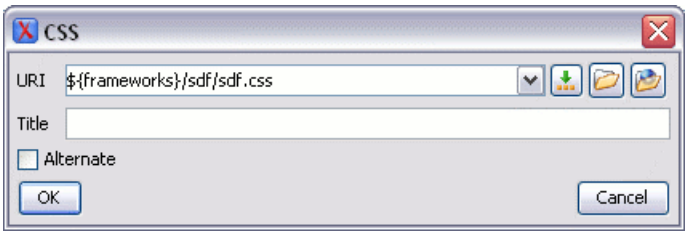
Author CSS Settings

Select the Author tab from the Document Type edit dialog. By clicking on the CSS label in the right part of the tab the list of associated CSS files is shown.

Here you can also specify how should the CSSs defined in the document type be treated when there are CSSs specified in the document(with `xml-stylesheet` processing instructions). The CSSs from the document can either replace the CSSs defined in the document type association or merge with them.

Add the URI of the CSS file `sdf.css` we already defined. We should use the `${frameworks}` editor variable in the file path.

Figure 11. CSS settings dialog



The Title text field refers to a symbolic name for the stylesheet. When adding several stylesheets with different titles to a Document Type association, the content author can select what CSS will be used for editing from the **Author CSS Alternatives** toolbar.

This combo-box from the toolbar is also populated in case your XML document refers CSS files directly using `xml-stylesheet` processing instructions, and the processing instructions define titles for the CSS files.

Note

The CSS settings dialog allows to create a *virtual*`xml-stylesheet` processing instructions. The CSS files defined in the Document Type Association dialog and the `xml-stylesheet` processing instructions from the XML document are processed together, as being all a list of processing instructions.

<oXygen/> Author fully implements the W3C recommendation regarding "Associating Style Sheets with XML documents". For more information see: http://www.w3.org/TR/xml-stylesheet/http://www.w3.org/TR/REC-html40/present/styles.html#h-14.3.2

Testing the Document Type Association

To test the new Document Type create an XML instance that is conforming with our Simple Document Format. We will not specify an XML Schema location directly in the document, using an `xsi:schemaLocation` attribute; <oXygen/> will detect instead its associated document type and use the specified schema.

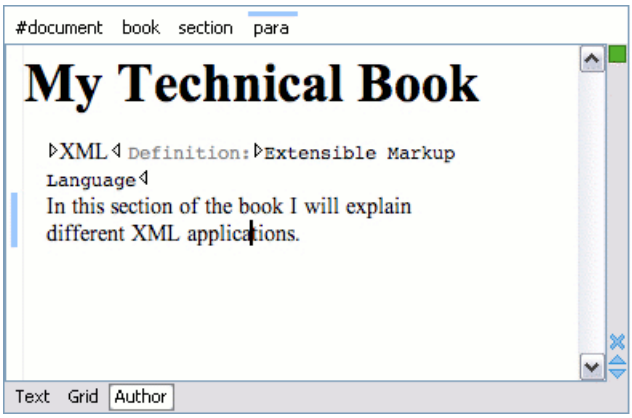
```
<book xmlns="http://www.oxygenxml.com/sample/documentation"
      xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">

    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will
           explain different XML applications.</para>
    </section>
</book>
```

When trying to validate the document there should be no errors. Now modify the `title` to `title2`. Validate again. This time there should be one error:

```
cvc-complex-type.2.4.a: Invalid content was found starting with element
'title2'. One of '{"http://www.oxygenxml.com/sample/documentation":title}'
is expected.
```

Undo the tag name change. Press on the Author button at the bottom of the editing area. <oXygen/> should load the CSS from the document type association and create a layout similar to this:



Packaging and Deploying

Using a file explorer, go to the <oXygen/> `frameworks` directory. Select the `sdf` directory and make an archive from it. Move it to another <oXygen/> installation (eventually on another computer). Extract it in the `frameworks` directory. Start <oXygen/> and test the association as explained above.

If you create multiple document type associations and you have a complex directory structure it might be easy from the deployment point of view to use an <oXygen/> all platforms distribution. Add your framework files to it, repackage it and send it to the content authors.

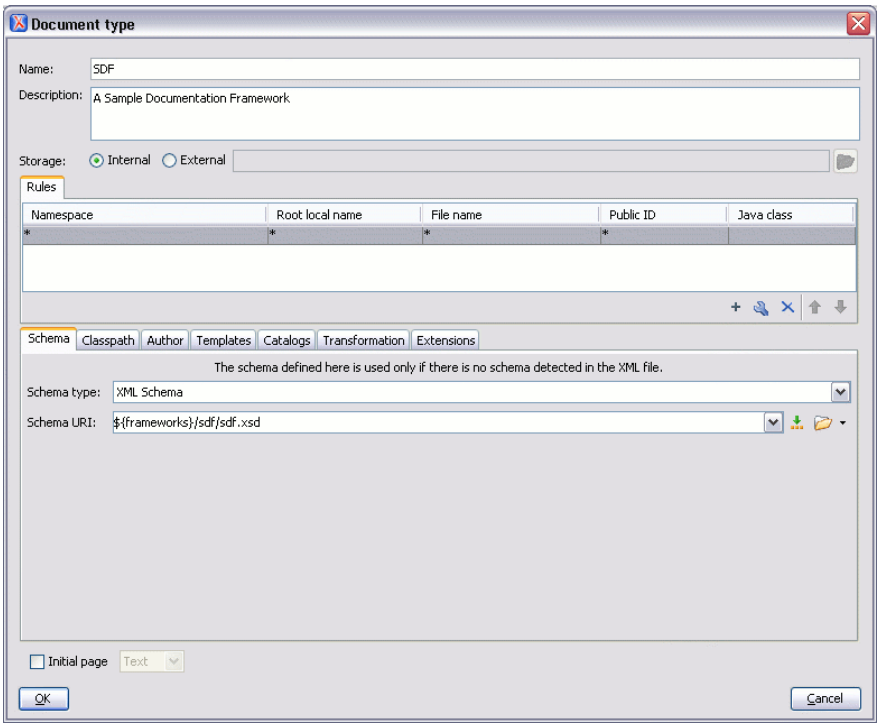
Warning

When deploying your customized sdf directory please make sure that your sdf directory contains the sdf.framework file (that is the file defined as External Storage in Document Type Association dialog shall always be stored inside the sdf directory). If your external storage points somewhere else <oXygen/> will not be able to update the Document Type Association options automatically on the deployed computers.

Author Settings

You can add a new *Document Type Association* or edit the properties of an existing one from the Options+Preferences+Document Type Association option pane. All the changes can be made into the *Document type* edit dialog.

Figure 12. The Document Type Dialog



Configuring Actions, Menus and Toolbars

The <oXygen/> Author toolbars and menus can be changed to provide a productive editing experience for the content authors. You can create a set of actions that are specific to a document type.

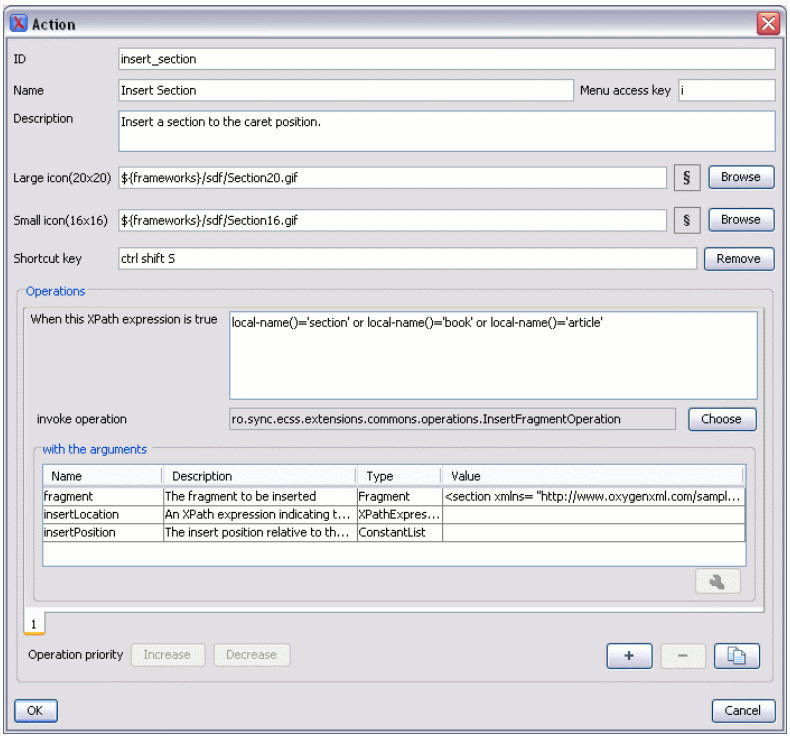
In our example, the `sdf` framework, we created the stylesheet and the validation schema. Now let's add some actions for inserting a `section` and a `table`. To add a new action, follow the procedure:

1. Open the Options Dialog, and select the Document Types Association option pane.
2. In the lower part of the Document Type Association dialog, click on the Author tab, then select the Actions label.
3. To add a new action click on the + Add button.

The Insert Section Action

This paragraph describes how you can define the action for adding a section. We assume the icon files `Section16.gif` for the menu item and `Section20.gif` for the toolbar, are already available. Although we could use the same icon size for both menu and toolbar, usually the icons from the toolbars are larger than the ones placed in the menus. These files should be placed in the `frameworks/sdf` directory.

Figure 13. The Action Edit Dialog



ID — An unique identifier for the action. You can use **insert_section**.

Name — The name of the action. It is displayed as a tooltip when the action is placed in the toolbar, or as the menu item name. Use **Insert section**.

Menu access key — On Windows, the menu items can be accessed using (ALT + letter) combination, when the menu is visible. The letter is visually represented by underlining the first letter from the menu item name having the same value. Since the name is **Insert section**, we can use as a menu access key the letter **s**.

Description — You can add a short description for the action. In our case **Adds a section element** will suffice.

Large icon (20x20) — The path to the file that contains the toolbar image for the action. A good practice is to store the image files inside the framework directory. This way we can use the editor variable `${frameworks}` to make the image file relative to the framework location. Insert **${frameworks}/sdf/Section20.gif**

Note

If the images are bundled in a jar archive together with some Java operations implementation for instance, it might be convenient for you to refer the images not by the file name, but by their relative path location in the class-path.

If the image file `Section20.gif` is located in the directory `images` inside the jar archive, you can refer to it by using **/images/Section20.gif**. The jar file must be added into the Classpath list.

Small icon (16x16) — The path to the file that contains the menu image. Insert **${frameworks}/sdf/Section16.gif**

Shortcut key — A shortcut key combination for triggering the action. To define it, click in the text field and press the desired key combination. We can choose **Ctrl+Shift+s**.

Note

The shortcut is enabled only by adding the action to the main menu of the Author mode which contains all the actions that the author will have in a menu for the current document type.

At this time the action has no functionality added to it. Next we must define how this action operates. An action can have multiple operation modes, each of them activated by the evaluation of an XPath version 2.0 expression.

Note

The XPath expression of an operation mode is evaluated relative to the **current element**. The current element is the one where the caret is positioned. In fact there is hierarchy of elements containing the caret position, but we are considering only the closest one. A simple expression like:

```
title
```

is a relative one and checks if the current element has a "title" child element. To check that the current element is a "section" we can use the expression:

```
local-name()='section'
```

Note

<oXygen/> Author determines the operation to be executed by iterating through the defined operation modes. The first operation whose XPath expression "matched" the current document context gets executed, while the others are being ignored. Make sure you order correctly your operations by placing the ones with more specific XPath selectors before the ones having more generic selectors.

For instance the expression

```
person[@name='Cris' and @age='24']
```

is more specific than

```
person[@name='Cris']
```

The action mode using the first expression must be placed before the one using the second expression in the action modes list.

We decide that we can add sections only if the current element is either a `book`, `article`, or another `section`.

XPath expression — Set the value to:

```
local-name()='section' or local-name()='book' or
 local-name()='article'
```

Invoke operation — A set of built-in operations is available. A complete list is found in the Author Default Operations section. To this set you can add your own Java operation implementations. In our case, we'll use the **InsertFragmentOperation** built-in operation, that inserts an XML fragment at the caret position.

Configure the arguments by setting the following values:

fragment —

```
<section xmlns=
"http://www.oxygenxml.com/sample/documentation">
     <title/>
</section>
```

insertLocation — Leave it empty. This means the location will be the element at the caret position.

insertPosition — Select "Inside".

The Insert Table Action

We will create an action that inserts into the document a table with three rows and three columns. The first row is the table header. Similarly to the insert section action, we will use the **InsertFragmentOperation**.

We assume the icon files  `Table16.gif` for the menu item and  `Table20.gif` for the toolbar, are already available. We place these files in the `frameworks/sdf` directory.

The action properties:

ID — You can use **insert_table**.

Name — Insert **Insert table**.

Menu access key — Enter the **t** letter.

Description — We can use **Adds a section element**.

Toolbar icon — Use **${frameworks}/sdf/Table20.gif**

Menu icon — Insert **${frameworks}/sdf/Table16.gif**

Shortcut key — We can choose **Ctrl+Shift+t**.

Now let's set up the operation the action uses.

XPath expression — Set it to the value

```
true()
```

Note

true() is equivalent with leaving this field empty.

Invoke operation

We'll use **InsertFragmentOperation** built-in operations that inserts an XML fragment at the caret position.

Configure its arguments by setting the values:

fragment

```
<table xmlns=
"http://www.oxygenxml.com/sample/documentation">
  <header><td/><td/><td/></header>
  <tr><td/><td/><td/></tr>
  <tr><td/><td/><td/></tr>
</table>
```

insertLocation

In our example will always add tables at the end of the section that contains the caret position. Use:

```
ancestor::section/*[last()]
```

insertPosition

Select "After".

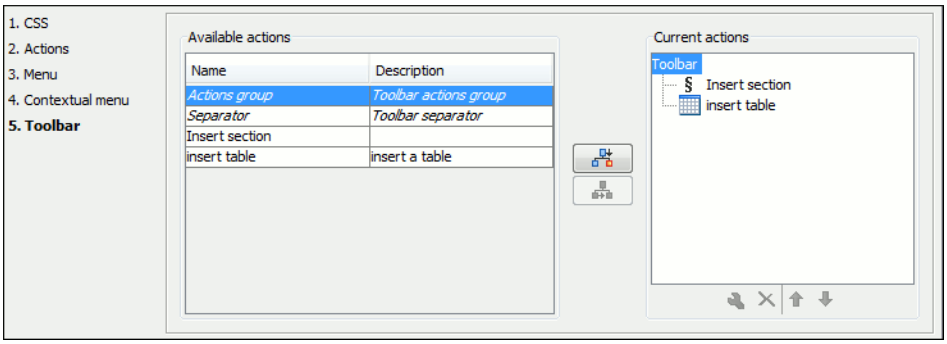
Configuring the Toolbars

Now that we have defined the two actions we can add them to the toolbar. You can configure additional toolbars on which to add your custom actions.

The first thing to check is that the toolbar Author custom actions should be displayed when switching to the **Author** mode: Right click in the application window upper part, in the area that contains the toolbar buttons and check Author custom actions in the displayed menu if it is unchecked.

Open the Document Type edit dialog for the **SDF** framework and select on the Author tab. Next click on the Toolbar label.

Figure 14. Configuring the Toolbar



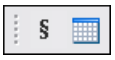
The panel is divided in two sections: the left side contains a list of actions, while the right one contains an action tree, displaying the list of actions added in the toolbar. The special entry called *Separator* allows you to visually separate the actions in the toolbar.

Select the Insert section action in the left and the Toolbar label in the right, then press the Add as child button.

Now select the Insert table action in the left and the Insert section in the right. Press the Add as sibling button.

When opening a **Simple Documentation Framework** test document in Author mode, the toolbar below will be displayed at the top of the editor.

Figure 15. Author Custom Actions Toolbar



Tip

If you have many custom toolbar actions or want to group actions according to their category you can add additional toolbars with custom names and split the actions to better suit your purpose.

Configuring the Main Menu

Defined actions can be grouped into customized menus in the <oXygen/> menu bar. For this open the Document Type dialog for the **SDF** framework and click on the Author tab. Next click on the Menu label.

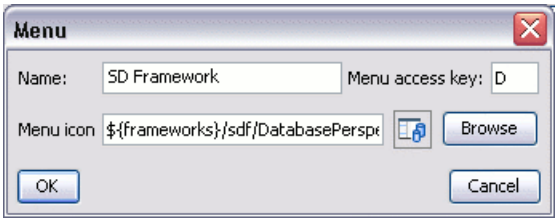
In the left side we have the list of actions and some special entries:

Submenu — Creates a submenu. You can nest an unlimited number of menus.

Separator — Creates a separator into a menu. In this way you can logically separate the menu entries.

In the right side we have the menu tree, having the Menu entry as root. To change its name click on this label to select it, then press the Edit button. Enter **SD Framework** as name, and **D** as menu access key.

Figure 16. Changing the Name of the Menu

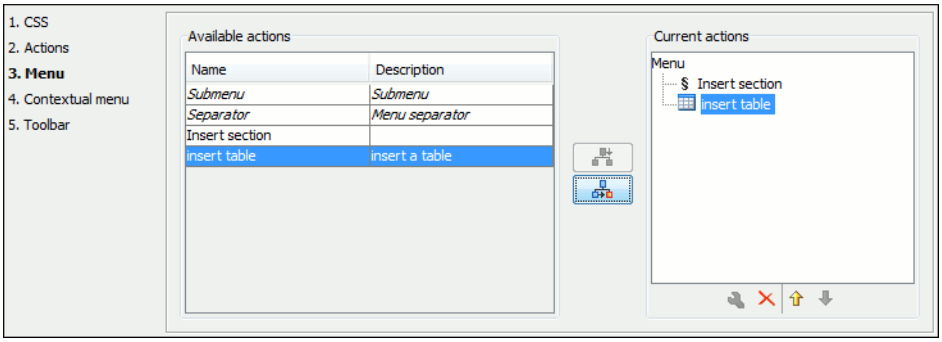


Select the Submenu label in the left an the SD Framework label in the right, then press the Add as child button. Change the submenu name to Table, using the Edit button.

Select the Insert section action in the left and the Table label in the right, then press the Add as sibling button.

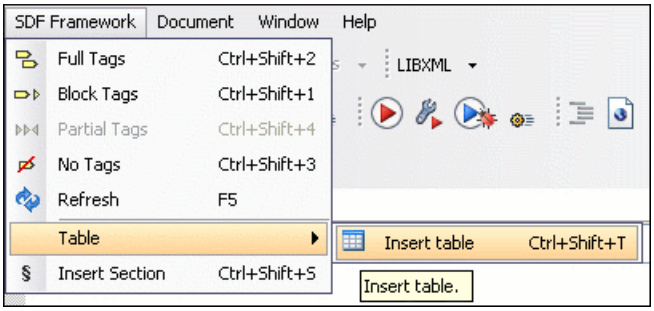
Now select the Insert table action in the left and the Table in the right. Press the Add as child button.

Figure 17. Configuring the Menu



When opening a **Simple Documentation Framework** test document in Author mode, the menu we created is displayed in the editor menu bar, between the Debugger and the Document menus. In the menu we find the Table submenu and the two actions:

Figure 18. Author Menu



Note

The shortcut of an action defined for the current document type is enabled only if the action is added to the main menu. Otherwise the author can run the action only from the toolbar.

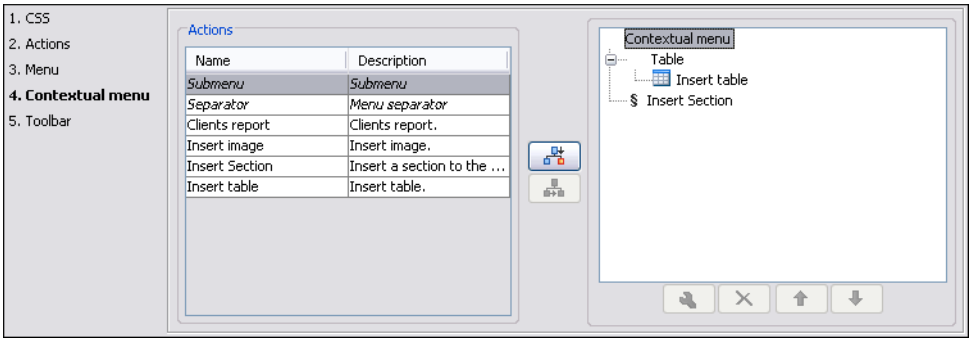
Configuring the Contextual Menu

The contextual menu is shown when you right click (on Mac OS X it is used the combination **ctrl** and mouse click) in the Author editing area. In fact we are configuring the bottom part of the menu, since the top part is reserved for a list of generic actions like Copy, Paste, Undo, etc..

Open the Document Type dialog for the **SDF** framework and click on the Author tab. Next click on the Contextual Menu label.

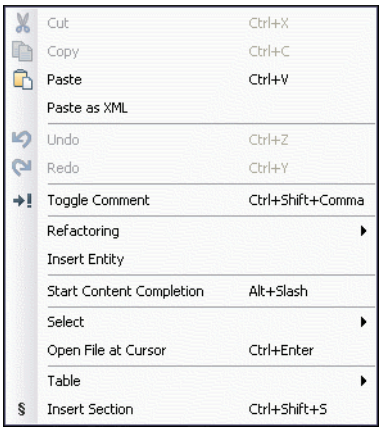
Follow the same steps as explained above in the Configuring the Main Menu, except changing the menu name - the contextual menu has no name.

Figure 19. Configuring the Contextual Menu



To test it, open the test file, and click to open the contextual menu. In the lower part there is shown the Table sub-menu and the Insert section action:

Figure 20. Author Contextual Menu



Author Default Operations

Below are listed all the operations and their arguments.

InsertFragmentOperation — Inserts an XML fragment at the current cursor position. The selection - if there is one, remains unchanged. The fragment will be inserted in the current context of the cursor position. That means that if the current XML document uses some namespace declarations then the inserted fragment must use the same declarations. The inserted fragment will not be copied and pasted to the cursor position, but the namespace declarations of the fragment will be adapted if needed to the existing namespace declarations of the XML document. Examples of namespace adjusting when the fragment is inserted and the descriptions of the arguments are described here.

InsertOrReplaceFragmentOperation — Similar to **InsertFragmentOperation**, except it removes the selected content before inserting the fragment.

InsertOrReplaceTextOperation — Inserts a text. It removes the selected content before inserting the text section.

text — The text section to insert.

SurroundWithFragmentOperation — Surrounds the selected content by a fragment. Since the fragment can have multiple nodes, the surrounded content will be always placed in the first leaf element. If there is no selection, the operation will simply insert the fragment at the caret position. The arguments are described here.

SurroundWithTextOperation — The surround with text operation takes two arguments, two text values that will be inserted before and after the selected content. If there is no selected content, the two sections will be inserted at the caret position. The arguments of the operation are:

- header — The text that will be placed before the selection.
- footer — The test that will be placed after the selection.

The arguments of `InsertFragmentOperation`

fragment — The value for this argument is a text. This is parsed by the <oXygen/> Author as it was already in the document at the caret position. You can use entities references declared in the document and it is namespace aware. The fragment may have multiple roots.

Note

You can use even namespace prefixes that are not declared in the inserted fragment, if they are declared in the document where the insertion is done. For clarity, we recommend always to prefix and declare namespaces in the inserted fragment!

Note

If there are namespace declarations in the fragment that are identical to the in the document insertion context, the namespace declaration attributes are removed from the fragment elements.

Example 2. Prefixes that are not bound explicitly

For instance, the fragment:

```
<x:item id="dty2"/>
&ent;
<x:item id="dty3"/>
```

Can be correctly inserted in the document: ('|' marks the insertion point):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE x:root [
    <!ENTITY ent "entity">
]>

<x:root xmlns:x="nsp">
  |
</x:root>
```

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE x:root [
    <!ENTITY ent "entity">
]>
<x:root xmlns:x="nsp">
    <x:item id="dty2"/>
    &ent;
    <x:item id="dty3"/>
</x:root>
```

Example 3. Default namespaces

If there is a default namespace declared in the document and the document fragment does not declare a namespace, the elements from the fragment are considered to be in **no namespace**.

For instance the fragment:

```
<item id="dty2"/>
<item id="dty3"/>
```

Inserted in the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
|
</root>
```

Gives the result document:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="nsp">
    <item xmlns="" id="dty2"/>
    <item xmlns="" id="dty3"/>
</root>
```

insertLocation — An XPath expression that is relative to the current node. It selects the reference node for the fragment insertion.

insertPosition — One of the three constants: "Inside", "After", or "Before" , showing where the insertion is made relative to the reference node selected by the **insertLocation**. "Inside" has the meaning of the first child of the reference node.

The arguments of `SurroundWithFragmentOperation`

fragment — The XML fragment that will surround the selection.

Example 4. Surrounding with a fragment

Let's consider the fragment:

```
<F>
   <A></A>
   <B>
     <C></C>
   </B>
</F>
```

And the document:

```
<doc>
  <X></X>
  <Y></Y>
  <Z></Z>
<doc>
```

Considering the selected content that is to be surrounded is the sequence of elements `X` and `Y`, then the result is:

```
<doc>
   <F>
      <A>
         <X></X>
         <Y></Y>
      </A>
      <B>
        <C></C>
      </B>
   </F>
  <Z></Z>
<doc>
```

Because the element `A` was the first leaf in the fragment, it received the selected content. The fragment was then inserted in the place of the selection.

Java API - Extending Author Functionality through Java

<oXygen/> Author has a built-in set of operations covering the insertion of text and XML fragments (see the Author Default Operations) and the execution of XPath expressions on the current document edited in Author mode. However, there are situations in which we need to extend this set. For instance if you need to enter an element whose attributes

should be edited by the user through a graphical user interface. Or the users must send the selected element content or even the whole document to a server, for some kind of processing or the content authors must extract pieces of information from a server and insert it directly into the edited XML document. Or you need to apply an XPath expression on the current Author document and process the nodes of the result nodeset.

In the following sections we are presenting the Java programming interface (API) available to the developers. You will need the Oxygen Author SDK [http://www.oxygenxml.com/InstData/Editor/Developer/oxygenAuthorSDK.zip] available on the <oXygen/> website [http://www.oxygenxml.com/developer.html] which includes the source code of the Author operations in the predefined document types and the full documentation in Javadoc format of the public API available for the developer of Author custom actions.

The next Java examples are making use of AWT classes. If you are developing extensions for the <oXygen/> XML Editor plugin for Eclipse you will have to use their SWT counterparts.

We assume you already read the Configuring Actions, Menus, Toolbar section and you are familiar with the <oXygen/> Author customization. You may find the XML schema, CSS and XML sample in the Example Files Listings.

Warning

Make sure the Java classes of your custom Author operations are compiled with the same Java version that is used by <oXygen/> XML Author . Otherwise the classes may not be loaded by the Java virtual machine. For example if you run <oXygen/> XML Author with a Java 1.5 virtual machine but the Java classes of your custom Author operations are compiled with a Java 1.6 virtual machine then the custom operations cannot be loaded and used by the Java 1.5 virtual machine.

Example 1. Step by Step Example. Simple Use of a Dialog from an Author Operation.

Let's start adding functionality for inserting images, in our **Simple Documentation Framework** (shortly SDF). The images are represented by the `image` element. The location of the image file is represented by the value of the `href` attribute. In our Java implementation we will show a dialog with a text field, in which the user can enter a full URL, or he can browse for a local file.

1. Create a new Java project, in your IDE.

   Create the directory `lib` in the Java project directory and copy in it the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` directory. The `oxygen.jar` contains the Java interfaces we have to implement and the API needed to access the Author features.

2. Create the class `simple.documentation.framework.InsertImageOperation`. This class must implement the `ro.sync.ecss.extensions.api.AuthorOperation` interface.

   The interface defines three methods: `doOperation`, `getArguments` and `getDescription`.

   1. The `doOperation` method is invoked when the action is performed either by pressing the toolbar button, selecting the menu item or through the shortcut. It takes as arguments an object of type `AuthorAccess` and a map or argument names and values.

   2. The `getArguments` method is used by <oXygen/> when the action is configured, it returns the list of arguments (name and type) that are accepted by the operation.

   3. The `getDescription` method is also used by <oXygen/> when the operation is configured and its return value describes what the operation does.

   Here is the implementation of these three methods.

```
 /**
 * Performs the operation.
 */
public void doOperation(
               AuthorAccess authorAccess,
               ArgumentsMap arguments)
 throws IllegalArgumentException,
                  AuthorOperationException {

 JFrame oxygenFrame = (JFrame) authorAccess.getParentFrame();
 String href = displayURLDialog(oxygenFrame);
 if (href.length() != 0) {
     // Creates the image XML fragment.
     String imageFragment =
        "<image xmlns='http://www.oxygenxml.com/sample/documentation' href='"
        + href + "'/>";

     // Inserts this fragment at the caret position.
     int caretPosition = authorAccess.getCaretOffset();
     authorAccess.insertXMLFragment(imageFragment, caretPosition);
 }
}

/**
 * Has no arguments.
 *
 * @return null.
 */
public ArgumentDescriptor[] getArguments() {
 return null;
}

/**
 * @return A description of the operation.
 */
public String getDescription() {
 return "Inserts an image element. Asks the user for a URL reference.";
}
```

The complete source code of our operation is found in the Example Files Listings, the Java Files section.

Important

Make sure you always specify the namespace of the inserted fragments.

3. Package the compiled class into a jar file. An example of an ANT script that packages the `classes` directory content into a jar archive named `sdf.jar` is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="project" default="dist">
    <target name="dist">
```

```
    <jar destfile="sdf.jar" basedir="classes">
     <fileset dir="classes">
      <include name="**/*"/>
      </fileset>
    </jar>
     </target>
</project>
```

4. Copy the `sdf.jar` file into the `frameworks/sdf` directory.

5. Add the `sdf.jar` to the Author class path. To do this, Open the options Document Type Dialog, select **SDF** and press the Edit button.

   Select the Classpath tab in the lower part of the dialog.

   Press the + Add button . In the displayed dialog enter the location of the jar file, relative to the <oXygen/> `frameworks` directory:

   Figure 21. Adding a classpath entry

6. Let's create now the action which will use the defined operation. Click on the Actions label.

   We assume the icon files `Image16.gif` for the menu item and `Image20.gif` for the toolbar are already available. Place these files in the `frameworks/sdf` directory.

   Define the action properties:

ID	An unique identifier for the action. Use **insert_image**.
Name	The name of the action. Use **Insert image**.
Menu access key	Use the **i** letter.
Description	Enter the text **Inserts an image**.
Toolbar icon	Enter here: **${frameworks}/sdf/Image20.gif**
Menu icon	Enter here: **${frameworks}/sdf/Image16.gif**
Shortcut key	We will use: **Ctrl+Shift+i**.

   Now let's set up the operation.

   We are adding images only if the current element is a `section`, `book` or `article`.

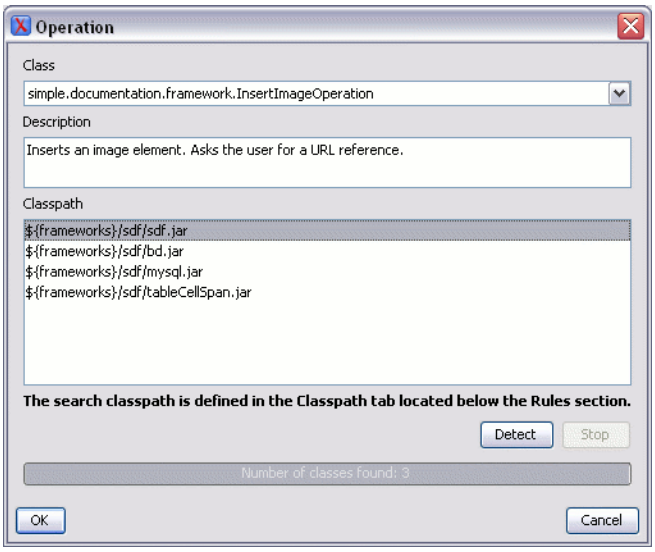
   XPath expression Set the value to:

```
local-name()='section' or local-name='book'
 or local-name='article'
```

Invoke operation

In this case, we'll use our Java operation we defined earlier. Press the Choose button, then select `simple.documentation.framework.InsertImageOperation`.

Figure 22. Selecting the Operation

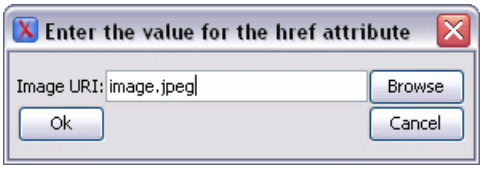


This operation has no arguments.

7. Add the action to the toolbar, using the Toolbar panel.

To test the action, you can open the `sdf.xml` sample, then place the caret inside a `section` between two `para` elements for instance. Press the button associated with the action from the toolbar. In the dialog select an image URL and press Ok. The image is inserted into the document.

Figure 23. Dialog Displayed by the Insert Image Operation



Example 2. Operations with Arguments. Report from Database Operation.

In this example we will create an operation that connects to a relational database and executes an SQL statement. The result should be inserted in the edited XML document as a `table`. To make the operation fully configurable, it will have arguments for the *database connection string*, the *user name*, the *password* and the *SQL expression*.

1. Create a new Java project, in your IDE.

   Create the directory `lib` in the Java project directory and copy in it the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` directory.

2. Create the class `simple.documentation.framework.QueryDatabaseOperation`. This class must implements the `ro.sync.ecss.extensions.api.AuthorOperation` interface.

```
import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class QueryDatabaseOperation implements AuthorOperation{
```

Let's define the arguments of the operation. For each of them we will use a `String` constant representing the argument name:

```
private static final String ARG_JDBC_DRIVER ="jdbc_driver";
private static final String ARG_USER ="user";
private static final String ARG_PASSWORD ="password";
private static final String ARG_SQL ="sql";
private static final String ARG_CONNECTION ="connection";
```

We must describe each of the argument name and type. To do this implement the `getArguments` method which will return an array of argument descriptors:

```
public ArgumentDescriptor[] getArguments() {
  ArgumentDescriptor args[] = new ArgumentDescriptor[] {
    new ArgumentDescriptor(
      ARG_JDBC_DRIVER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the Java class that is the JDBC driver."),
    new ArgumentDescriptor(
      ARG_CONNECTION,
      ArgumentDescriptor.TYPE_STRING,
      "The database URL connection string."),
    new ArgumentDescriptor(
      ARG_USER,
      ArgumentDescriptor.TYPE_STRING,
      "The name of the database user."),
    new ArgumentDescriptor(
      ARG_PASSWORD,
      ArgumentDescriptor.TYPE_STRING,
      "The database password."),
    new ArgumentDescriptor(
      ARG_SQL,
      ArgumentDescriptor.TYPE_STRING,
      "The SQL statement to be executed.")
  };
  return args;
 }
```

These names, types and descriptions will be listed in the Arguments table when the operation is configured.

When the operation is invoked, the implementation of the `doOperation` method extracts the arguments, forwards them to the method that connects to the database and generates the XML fragment. The XML fragment is then inserted at the caret position.

```
public void doOperation(AuthorAccess authorAccess, ArgumentsMap map)
   throws IllegalArgumentException, AuthorOperationException {

  // Collects the arguments.
  String jdbcDriver =
   (String)map.getArgumentValue(ARG_JDBC_DRIVER);
  String connection =
   (String)map.getArgumentValue(ARG_CONNECTION);
  String user =
   (String)map.getArgumentValue(ARG_USER);
  String password =
   (String)map.getArgumentValue(ARG_PASSWORD);
  String sql =
   (String)map.getArgumentValue(ARG_SQL);

  int caretPosition = authorAccess.getCaretOffset();
  try {
   authorAccess.insertXMLFragment(
     getFragment(jdbcDriver, connection, user, password, sql),
     caretPosition);
  } catch (SQLException e) {
   throw new AuthorOperationException(
     "The operation failed due to the following database error: "
     + e.getMessage(), e);
  } catch (ClassNotFoundException e) {
   throw new AuthorOperationException(
     "The JDBC database driver was not found. Tried to load ' "
     + jdbcDriver + "'", e);
  }
 }
```

The `getFragment` method loads the JDBC driver, connects to the database and extracts the data. The result is a `table` element from the `http://www.oxygenxml.com/sample/documentation` namespace. The `header` element contains the names of the SQL columns. All the text from the XML fragment is escaped. This means that the '<' and '&' characters are replaced with the '<' and '&' character entities to ensure the fragment is well-formed.

```
private String getFragment(
  String jdbcDriver,
  String connectionURL,
  String user,
  String password,
  String sql) throws
   SQLException,
   ClassNotFoundException {

      Properties pr = new Properties();
      pr.put("characterEncoding", "UTF8");
      pr.put("useUnicode", "TRUE");
      pr.put("user", user);
      pr.put("password", password);

      // Loads the database driver.
      Class.forName(jdbcDriver);
```

```
        // Opens the connection
        Connection connection =
           DriverManager.getConnection(connectionURL, pr);
        java.sql.Statement statement =
           connection.createStatement();
        ResultSet resultSet =
           statement.executeQuery(sql);

        StringBuffer fragmentBuffer = new StringBuffer();
        fragmentBuffer.append(
          "<table xmlns=" +
          "'http://www.oxygenxml.com/sample/documentation'>");

        //
        // Creates the table header.
        //
        fragmentBuffer.append("<header>");
        ResultSetMetaData metaData = resultSet.getMetaData();
        int columnCount = metaData.getColumnCount();
        for (int i = 1; i <= columnCount; i++) {
            fragmentBuffer.append("<td>");
            fragmentBuffer.append(
              xmlEscape(metaData.getColumnName(i)));
            fragmentBuffer.append("</td>");
        }
        fragmentBuffer.append("</header>");

        //
        // Creates the table content.
        //
        while (resultSet.next()) {
            fragmentBuffer.append("<tr>");
            for (int i = 1; i <= columnCount; i++) {
                fragmentBuffer.append("<td>");
                fragmentBuffer.append(
                  xmlEscape(resultSet.getObject(i)));
                fragmentBuffer.append("</td>");
            }
            fragmentBuffer.append("</tr>");
        }

        fragmentBuffer.append("</table>");

        // Cleanup
        resultSet.close();
        statement.close();
        connection.close();
        return fragmentBuffer.toString();
    }
```

The complete source code of our operation is found in the Example Files Listings, the Java Files section.

3. Package the compiled class into a jar file.

4. Copy the jar file and the JDBC driver files into the `frameworks/sdf` directory.

5. Add the jars to the Author class path. For this, Open the options Document Type Dialog, select **SDF** and press the Edit button.

   Select the Classpath tab in the lower part of the dialog.

6. Click on the Actions label.

   The action properties are:

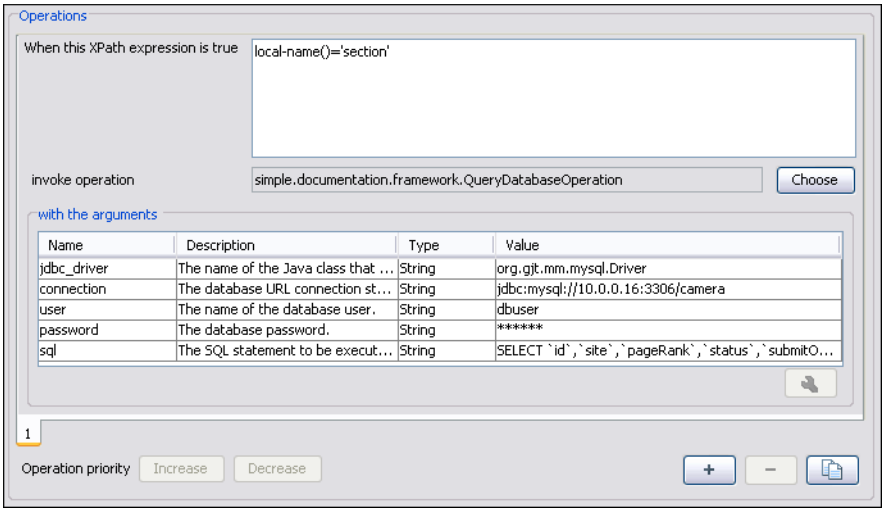
ID	An unique identifier for the action. Use **clients_report**.
Name	The name of the action. Use **Clients Report**.
Menu access key	Use the letter **r**.
Description	Enter the text **Connects to the database and collects the list of clients**.
Toolbar icon	Enter here: **${frameworks}/sdf/TableDB20.gif** We assume the image  `TableDB20.gif` for the toolbar action is already present in the `frameworks/sdf` directory.
Menu icon	Leave empty.
Shortcut key	We will use: **Ctrl+Shift+c**.

   Let's set up the operation. The action will work only if the current element is a `section`.

XPath expression	Set the value to: `local-name()='section'`
Invoke operation	In this case, we'll use our Java operation we defined earlier. Press the Choose button, then select `simple.documentation.framework.QueryDatabaseOperation`. Once selected, the list of arguments is displayed. In the figure below the first argument, *jdbc_driver*, represents the class name of the MySQL JDBC driver. The connection string has the URL syntax : *jdbc://<database_host>:<database_port>/<database_name>*. The SQL expression used in the example is: `SELECT userID, email FROM users` but it can be any valid SELECT expression which can be applied to the database.

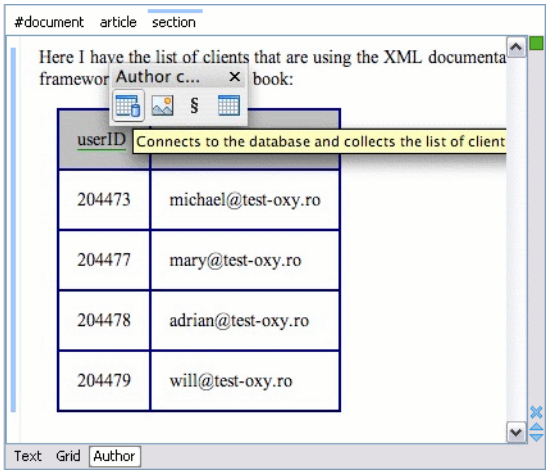
7. Add the action to the toolbar, using the Toolbar panel.

Figure 24. Java Operation Arguments Setup



To test the action you can open the `sdf.xml` sample place the caret inside a `section` between two `para` elements for instance. Press the Create Report button from the toolbar. You can see below the toolbar with the action button and sample table inserted by the Clients Report action.

Figure 25. Table Content Extracted from the Database



Configuring New File Templates

We will create a set of document templates that the content authors will use as starting points for creating new *Simple Document Framework* books and articles.

Each of the Document Type Associations can point to a directory usually named `templates` containing the file templates. All the files that are found here are considered templates for the respective document type. The template name is taken from the name of the file, and the template kind is detected from the file extension.

Create the `templates` directory into the `frameworks/SDF` directory. The directory tree for our documentation framework is now:

```
oxygen
  frameworks
     sdf
       schema
       css
       templates
```

Now let's create in this `templates` directory two files, one for the *book* template and another for the *article* template.

The `Book.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>Book Template Title</title>
    <section>
        <title>Section Title</title>
        <abs:def/>
        <para>This content is copyrighted:</para>
        <table>
            <header>
                <td>Company</td>
                <td>Date</td>
            </header>
            <tr>
                <td/>
                <td/>
            </tr>
        </table>
    </section>
</book>
```

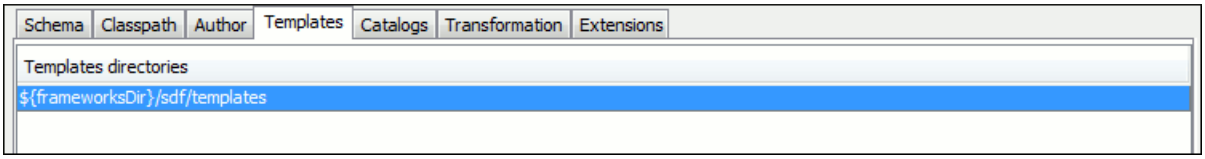
The `Article.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<article
    xmlns="http://www.oxygenxml.com/sample/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <title></title>
    <section>
        <title></title>
        <para></para>
        <para></para>
    </section>
</article>
```

You can also use editor variables in the template files' content and they will be expanded when the files are opened.

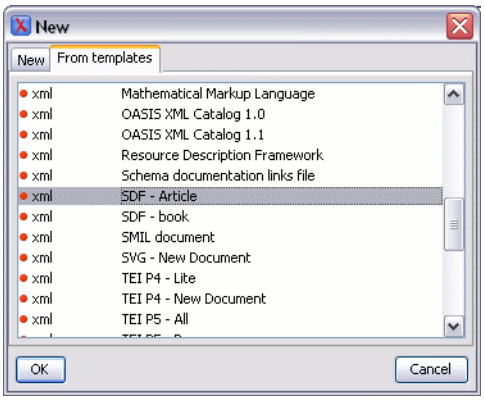
Open the Document Type dialog for the **SDF** framework and click on the Templates tab. Enter in the Templates directory text field the value `${frameworksDir}/sdf/templates`. As we already seen before, it is recommended that all the file references made from a Document Type Association to be relative to the `${frameworksDir}` directory. Binding a Document Type Association to an absolute file (e.g: "C:\some_dir\templates") makes the association difficult to share between users.

Figure 26. Setting the templates directory



To test the templates settings, press the File/New menu item to display the New dialog. The names of the two templates are prefixed with the name of the Document Type Association, in our case **SDF**. Selecting one of them should create a new XML file with the content specified in the template file.

Figure 27. Templates displayed in the New Dialog.



Configuring XML Catalogs

You can add catalog files to your Document Type Association using the Catalogs tab from the Document Type dialog.

Important

<oXygen/> XML Editor collects all the catalog files listed in the installed frameworks. No matter what the Document Type Association matches the edited file, all the catalog mappings are considered when resolving external references.

Important

The catalog files settings are available for all editing modes, not only for the **Author** mode.

In the XML sample file for **SDF** we did not used a `xsi:schemaLocation` attribute, but instead we let the editor use the schema from the association. However there are cases in which we must refer for instance the location of a schema file from a remote web location. In such cases the catalog may be used to map the web location to a local file system entry.

In the following section we will present an use-case for the XML catalogs, by modifying our `sdf.xsd` XML Schema file from the Example Files Listings.

The `sdf.xml` file refers the other file `abs.xsd` through an `import` element:

```
<xs:import namespace=
 "http://www.oxygenxml.com/sample/documentation/abstracts"
 schemaLocation="abs.xsd"/>
```

The `schemaLocation` attribute references the `abs.xsd` file located in the same directory. What if the file was on the web, at the `http://www.oxygenxml.com/SDF/abs.xsd` location for instance? In this case the attribute value will be:

```
<xs:import namespace=
 "http://www.oxygenxml.com/sample/documentation/abstracts"
 schemaLocation="http://www.oxygenxml.com/SDF/abs.xsd"/>
```

There is a problem with this approach. What happens if an Internet connection is not available? How will we check our document for errors if a part of the schema is not available? The answer is to create a catalog file that will help the parser locate the missing piece containing the mapping:

```
http://www.oxygenxml.com/SDF/abs.xsd -> ../local_path/abs.xsd
```

To do this create a new XML file called `catalog.xml` and save it into the `{oXygen_installation_directory}/frameworks/sdf` directory. The content of the file should be:

```
<?xml version="1.0"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
    <system
      systemId="http://www.oxygenxml.com/SDF/abs.xsd"
      uri="schema/abs.xsd"/>
</catalog>
```

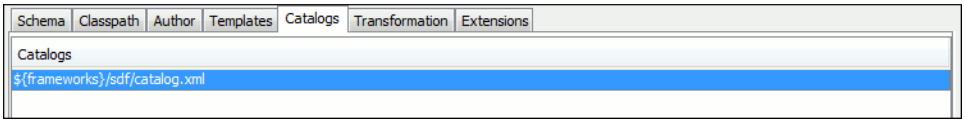
This means that all the references to `http://www.oxygenxml.com/SDF/abs.xsd` must be resolved to the `abs.xsd` file located in the `schema` directory.

Note

The references in the XML catalog files are relative to the directory that contains the catalog.

Save the catalog file and modify the `sdf.xsd` file by changing its `import` element, then add the catalog to the Document Type association. You can do this in the **Catalogs** tab by pressing the New button. Enter `${frameworks}/sdf/catalog.xml` in the displayed dialog.

Figure 28. Adding Catalogs to the Document Type Association



To test the catalog settings, restart <oXygen/> and try to validate a new sample **Simple Documentation Framework** document. There should be no errors.

Configuring Transformation Scenarios

When distributing a framework to the users, it is a good idea to have the transformation scenarios already configured. This would help the content authors publish their work in different formats. Being contained in the **Document Type Association** the scenarios can be distributed along with the actions, menus, toolbars, catalogs, etc.

In the following section we will create a transformation scenario for our framework.

Create the directory `xsl` in the directory `frameworks/sdf`. The directory structure for our documentation framework should be:

```
oxygen
  frameworks
     sdf
       schema
       css
       templates
       xsl
```

Create the `sdf.xsl` file in the `xsl` directory. The complete content of the `sdf.xsl` file is found in the Example Files Listings.

Open the Options/Preferences/Document Type Associations. Open the Document Type dialog for the **SDF** framework then choose the Transformation tab. Click on the New. In the Edit Scenario dialog, fill the following fields:

Name — The name of the transformation scenario. Enter *SDF to HTML*.

XSL URL — `${frameworks}/sdf/xsl/sdf.xsl`

Transformer — Saxon 9B.

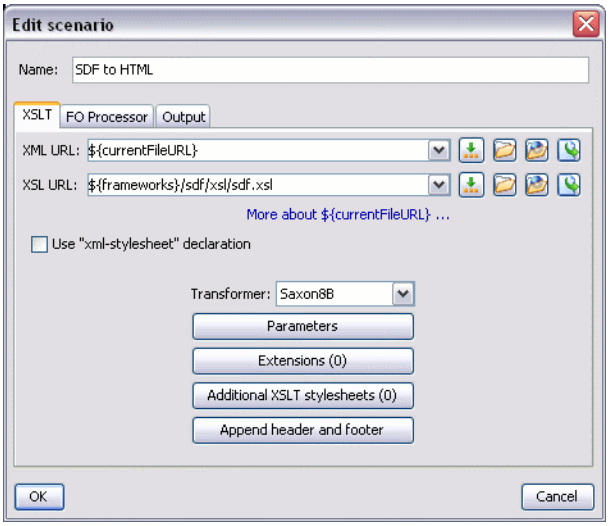
Change to the Output tab. Change the fields:

Save as — `${cfd}/${cfn}.html` This means the transformation output file will have the name of the XML file and the *html* extension and will be placed in the same directory.

Open in browser — Enable this option.

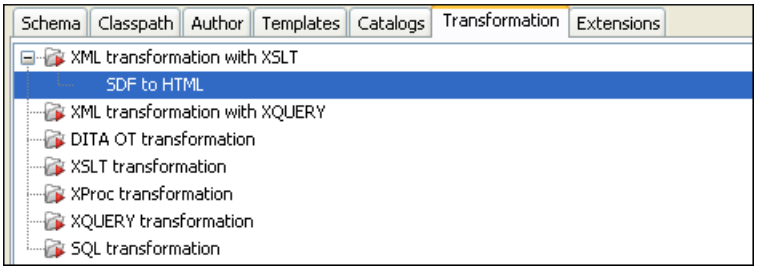
Saved file — Enable this checkbox.

Figure 29. Configuring a transformation scenario



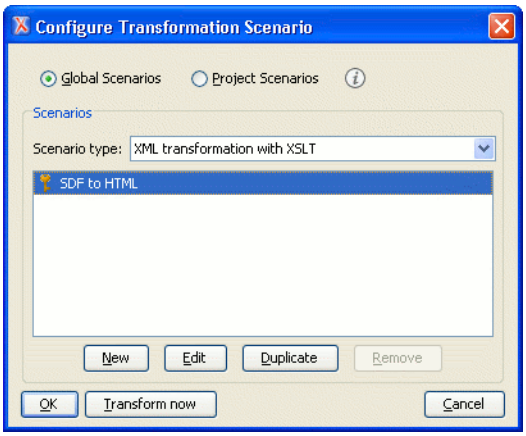
Now the scenario is listed in the Transformation tab:

Figure 30. The transformation tab



To test the transformation scenario we created, open the **SDF** XML sample from the Example Files Listings. Click on the Apply Transformation Scenario button. The Configure Transformation Dialog is displayed. Its scenario list contains the scenario we defined earlier *SDF to HTML*. Click on it then choose Transform now. The HTML file should be saved in the same directory as the XML file and opened in the browser.

Figure 31. Selecting the predefined scenario



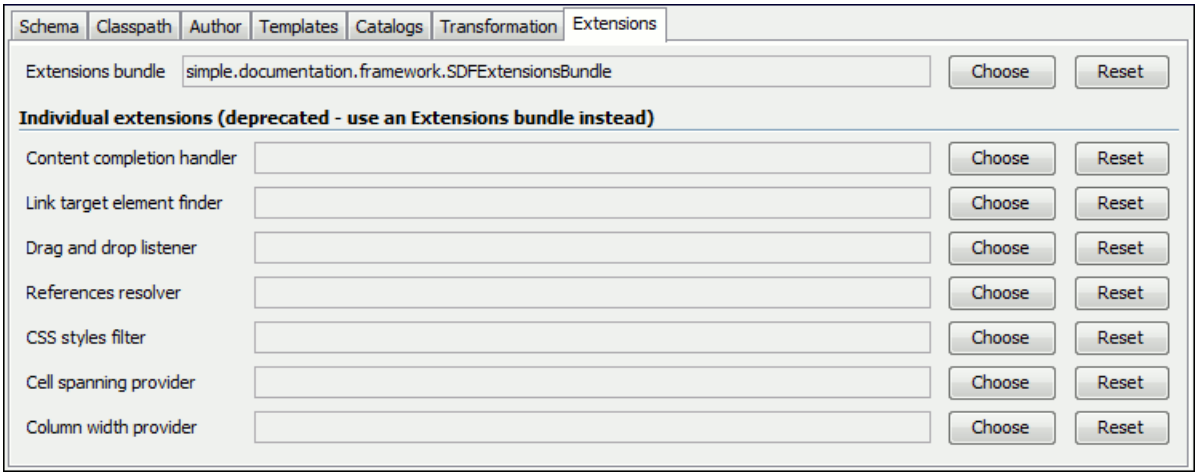
Note

The key symbol indicates that the scenario is read-only. It has this state because the scenario was loaded from a Document Type Association. The content authors can still change parameters and other settings if they are duplicating the scenario and edit the duplicate. In this case the copy of the scenario is created in the user local settings.

Configuring Extensions

You can add extensions to your Document Type Association using the Extensions tab from the Document Type dialog.

Figure 32. Configure extensions for a document type



Configuring an Extensions Bundle

Starting with <oXygen/> 10.3 version a single bundle was introduced acting as a provider for all other extensions. The individual extensions can still be set and if present they take precedence over the single provider, but this practice is being discouraged and the single provider should be used instead.

The extensions bundle is represented by the `ro.sync.ecss.extensions.api.ExtensionsBundle` class. The provided implementation of the `ExtensionsBundle` is instantiated when the rules of the Document Type Association defined for the custom framework match a document opened in the editor. Therefor references to objects which need to be persistent throughout the application running session must not be kept in the bundle because the next detection event can result in creating another `ExtensionsBundle` instance.

1. Create a new Java project, in your IDE.

   Create the `lib` directory in the Java project directory and copy in it the `oxygen.jar` file from the `{oXygen_installation_directory}/lib` directory.

2. Create the class `simple.documentation.framework.SDFExtensionsBundle` which must extend the abstract class `ro.sync.ecss.extensions.api.ExtensionsBundle`.

   ```
   public class SDFExtensionsBundle extends ExtensionsBundle {
   ```

   A **Document Type ID** and a short description should be defined first by implementing the methods `getDocumentTypeID` and `getDescription`. The Document Type ID is used to uniquely identify the current framework. Such an ID must be provided especially if options related to the framework need to be persistently stored and retrieved between sessions.

   ```
   public String getDocumentTypeID() {
        return "Simple.Document.Framework.document.type";
   }

   public String getDescription() {
        return "A custom extensions bundle used for the Simple Document" +
                     "Framework document type";
   }
   ```

In order to be notified about the activation of the custom Author extension in relation with an opened document an `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` should be implemented. The **activation** and **deactivation** events received by this listener should be used to perform custom initializations and to register/remove listeners like `ro.sync.ecss.extensions.api.AuthorListener`, `ro.sync.ecss.extensions.api.AuthorMouseListener` or `ro.sync.ecss.extensions.api.AuthorCaretListener`. The custom author extension state listener should be provided by implementing the method `createAuthorExtensionStateListener`.

```
public AuthorExtensionStateListener createAuthorExtensionStateListener() {
     return new SDFAuthorExtensionStateListener();
}
```

The `AuthorExtensionStateListener` is instantiated and notified about the activation of the framework when the rules of the Document Type Association match a document opened in the Author editor page. The listener is notified about the deactivation when another framework is activated for the same document, the user switches to another page or the editor is closed. A complete description and implementation of an `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` can be found in the Implementing an Author Extension State Listener.

If Schema Aware mode is active in Oxygen, all actions that can generate invalid content will be redirected toward the `AuthorSchemaAwareEditingHandler`. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an `InvalidEditException`. The actions that are forwarded to this handler include typing, delete or paste.

See the Implementing an Author Schema Aware Editing Handler section for more details about this handler.

Customizations of the content completion proposals are permitted by creating a schema manager filter extension. The interface that declares the methods used for content completion proposals filtering is `ro.sync.contentcompletion.xml.SchemaManagerFilter`. The filter can be applied on elements, attributes or on their values. Responsible for creating the content completion filter is the method `createSchemaManagerFilter`. A new `SchemaManagerFilter` will be created each time a document matches the rules defined by the Document Type Association which contains the filter declaration.

```
public SchemaManagerFilter createSchemaManagerFilter() {
     return new SDFSchemaManagerFilter();
}
```

A detailed presentation of the schema manager filter can be found in Configuring a Content completion handler section.

The <oXygen/> Author supports link based navigation between documents and document sections. Therefor, if the document contains elements defined as links to other elements, for example links based on the **id** attributes, the extension should provide the means to find the referred content. To do this an implementation of the `ro.sync.ecss.extensions.api.link.ElementLocatorProvider` interface should be returned by the `createElementLocatorProvider` method. Each time an element pointed by a link needs to be located the method is invoked.

```
public ElementLocatorProvider createElementLocatorProvider() {
     return new DefaultElementLocatorProvider();
}
```

The section that explains how to implement an element locator provider is Configuring a Link target element finder.

The drag and drop functionality can be extended by implementing the `ro.sync.exml.editor.xmleditor.pageauthor.AuthorDnDListener` interface. Relevant methods from the listener are invoked when the mouse is dragged, moved over, or exits the author editor page, when the drop action changes, and when the drop occurs. Each method receives the `DropTargetEvent` containing information about the drag and drop operation. The drag and drop extensions are available on Author page for both <oXygen/> Eclipse plugin and standalone application. The Text page corresponding listener is available only for <oXygen/> Eclipse plugin. The methods corresponding to each implementation are: `createAuthorAWTDndListener`, `createTextSWTDndListener` and `createAuthorSWTDndListener`.

```
public AuthorDnDListener createAuthorAWTDndListener() {
     return new SDFAuthorDndListener();
}
```

For more details about the Author drag and drop listeners see the Configuring a custom Drag and Drop listener section.

Another extension which can be included in the bundle is the reference resolver. In our case the references re represented by the **ref** element and the attribute indicating the referred resource is **location**. To be able to obtain the content of the referred resources we will have to implement a Java extension class which implements the `ro.sync.ecss.extensions.api.AuthorReferenceResolver`. The method responsible for creating the custom references resolver is `createAuthorReferenceResolver`. The method is called each time a document opened in an Author editor page matches the Document Type Association where the extensions bundle is defined. The instantiated references resolver object is kept and used until another extensions bundle corresponding to another Document Type is activated as result of the detection process.

```
public AuthorReferenceResolver createAuthorReferenceResolver() {
     return new ReferencesResolver();
}
```

A more detailed description of the references resolver can be found in the Configuring a References Resolver section.

To be able to dynamically customize the default CSS styles for a certain `AuthorNode` an implementation of the `ro.sync.ecss.extensions.api.StylesFilter` can be provided. The extensions bundle method responsible for creating the `StylesFilter` is `createAuthorStylesFilter`. The method is called each time a document opened in an Author editor page matches the document type association where the extensions bundle is defined. The instantiated filter object is kept and used until another extensions bundle corresponding to another Document Type is activated as result of the detection process.

```
public StylesFilter createAuthorStylesFilter() {
     return new SDFStylesFilter();
}
```

See the Configuring CSS styles filter section for more details about the styles filter extension.

In order to edit data in custom tabular format implementations of the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` and the `ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interfaces should be provided. The two methods from the `ExtensionsBundle` specifying these two extension points are `createAuthorTableCellSpanProvider` and `createAuthorTableColumnWidthProvider`.

```
public AuthorTableCellSpanProvider createAuthorTableCellSpanProvider() {
     return new TableCellSpanProvider();
}
```

```
public AuthorTableColumnWidthProvider
         createAuthorTableColumnWidthProvider() {
     return new TableColumnWidthProvider();
}
```

The two table information providers are not reused for different tables. The methods are called for each table in the document so new instances should be provided every time. Read more about the cell span and column width information providers in Configuring a Table Cell Span Provider and Configuring a Table Column Width Provider sections.

If the functionality related to one of the previous extension point does not need to be modified then the developed `ExtensionsBundle` should not override the corresponding method and leave the default base implementation to return **null**.

3. Package the compiled class into a jar file.
4. Copy the jar file into the `frameworks/sdf` directory.
5. Add the jar file to the Author class path.
6. Register the Java class by clicking on the Extensions tab. Press the Choose button and select from the displayed dialog the name of the class: `SDFExtensionsBundle`.

The complete source code of the `SDFExtensionsBundle` implementation is found in the Example Files Listings, the Java Files section.

Implementing an Author Extension State Listener

The `ro.sync.ecss.extensions.api.AuthorExtensionStateListener` implementation is notified when the Author extension where the listener is defined is activated or deactivated in the Document Type detection process.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorExtensionStateListener;

public class SDFAuthorExtensionStateListener implements
  AuthorExtensionStateListener {
  private AuthorListener sdfAuthorDocumentListener;
  private AuthorMouseListener sdfMouseListener;
  private AuthorCaretListener sdfCaretListener;
  private OptionListener sdfOptionListener;
```

The **activation** event received by this listener when the rules of the Document Type Association match a document opened in the Author editor page, should be used to perform custom initializations and to register listeners like `ro.sync.ecss.extensions.api.AuthorListener`, `ro.sync.ecss.extensions.api.AuthorMouseListener` or `ro.sync.ecss.extensions.api.AuthorCaretListener`.

```
 public void activated(AuthorAccess authorAccess) {
   // Get the value of the option.
   String option = authorAccess.getOptionsStorage().getOption(
               "sdf.custom.option.key", "");
   // Use the option for some initializations...

   // Add an option listener.
   authorAccess.getOptionsStorage().addOptionListener(sdfOptionListener);
```

```
   // Add author document listeners.
   sdfAuthorDocumentListener = new SDFAuthorListener();
   authorAccess.getDocumentController().addAuthorListener(
               sdfAuthorDocumentListener);

   // Add mouse listener.
   sdfMouseListener = new SDFAuthorMouseListener();
   authorAccess.getEditorAccess().addAuthorMouseListener(sdfMouseListener);

   // Add caret listener.
   sdfCaretListener = new SDFAuthorCaretListener();
   authorAccess.getEditorAccess().addAuthorCaretListener(sdfCaretListener);

   // Other custom initializations...

 }
```

The *authorAccess* parameter received by the `activated` method can be used to gain access to Author specific actions and informations related to components like the editor, document, workspace, tables, change tracking a.s.o.

If options specific to the custom developed Author extension need to be stored or retrieved, a reference to the `OptionsStorage` can be obtained by calling the `getOptionsStorage` method from the author access. The same object can be used to register `OptionListener` listeners. An option listener is registered in relation with an option **key** and will be notified about the value changes of that option.

An `AuthorListener` can be used if events related to the Author document modifications are of interest. The listener can be added to the `AuthorDocumentController`. A reference to the document controller is returned by the `getDocumentController` method from the author access. The document controller can also be used to perform operations involving document modifications.

To provide access to Author editor component related functionality and informations, the author access has a reference to the `AuthorEditorAccess` that can be obtained when calling the `getEditorAccess` method. At this level `AuthorMouseListener` and `AuthorCaretListener` can be added which will be notified about mouse and caret events occurring in the Author editor page.

The **deactivation** event is received when another framework is activated for the same document, the user switches to another editor page or the editor is closed. The `deactivate` method is typically used to unregister the listeners previously added on the `activate` method and to perform other actions. For example options related to the deactivated author extension can be saved at this point.

```
 public void deactivated(AuthorAccess authorAccess) {
   // Store the option.
   authorAccess.getOptionsStorage().setOption(
               "sdf.custom.option.key", optionValue);

   // Remove the option listener.
   authorAccess.getOptionsStorage().removeOptionListener(sdfOptionListener);

   // Remove document listeners.
   authorAccess.getDocumentController().removeAuthorListener(
               sdfAuthorDocumentListener);

   // Remove mouse listener.
```

```
   authorAccess.getEditorAccess().removeAuthorMouseListener(sdfMouseListener);

   // Remove caret listener.
   authorAccess.getEditorAccess().removeAuthorCaretListener(sdfCaretListener);

   // Other actions...

 }
```

Implementing an Author Schema Aware Editing Handler

You can implement your own handler for actions like typing, delete or paste by providing an implementation of `ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler`. The Schema Aware Editing must be **On** or **Custom** in order for this handler to be called. The handler can either resolve a specific case, let the default implementation take place or reject the edit entirely by throwing an `InvalidEditException`.

```
package simple.documentation.framework.extensions;

/**
 * Specific editing support for SDF documents. Handles typing and paste events inside sect
 */
public class SDFSchemaAwareEditingHandler implements AuthorSchemaAwareEditingHandler {
```

Typing events can be handled using the `handleTyping` method. For example, the `SDFSchemaAwareEditing-Handler` checks if the schema is not a learned one, was loaded successfully and Smart Paste is active. If these conditions are met, the event will be handled.

```
/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleTyping(int, char
 */
public boolean handleTyping(int offset, char ch, AuthorAccess authorAccess)
throws InvalidEditException {
  boolean handleTyping = false;
  AuthorSchemaManager authorSchemaManager = authorAccess.getDocumentController().getAuthor
  if (!authorSchemaManager.isLearnSchema() &&
      !authorSchemaManager.hasLoadingErrors() &&
      authorSchemaManager.getAuthorSchemaAwareOptions().isEnableSmartTyping()) {
    try {
      AuthorDocumentFragment characterFragment =
        authorAccess.getDocumentController().createNewDocumentTextFragment(String.valueOf(
      handleTyping = handleInsertionEvent(offset, new AuthorDocumentFragment[] {characterF
    } catch (AuthorOperationException e) {
      throw new InvalidEditException(e.getMessage(), "Invalid typing event: " + e.getMessa
    }
  }
  return handleTyping;
}
```

Implementing the `AuthorSchemaAwareEditingHandler` gives the possibility to handle other events like: the keyboard delete event at the given offset (using Delete or Backspace keys), delete element tags, delete selection, join elements or paste fragment.

The complete source code of our implementation is found in the Example Files Listings, the Java Files section.

Configuring a Content completion handler

You can filter or contribute to items offered for content completion by implementing the `ro.sync.contentcompletion.xml.SchemaManagerFilter` interface.

```
import java.util.List;

import ro.sync.contentcompletion.xml.CIAttribute;
import ro.sync.contentcompletion.xml.CIElement;
import ro.sync.contentcompletion.xml.CIValue;
import ro.sync.contentcompletion.xml.Context;
import ro.sync.contentcompletion.xml.SchemaManagerFilter;
import ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatElementsCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatPossibleValuesHasAttributeContext;

public class SDFSchemaManagerFilter implements SchemaManagerFilter {
```

You can implement the various callbacks of the interface either by returning the default values given by <oXygen/> or by contributing to the list of proposals. The filter can be applied on elements, attributes or on their values. Attributes filtering can be implemented using the `filterAttributes` method and changing the default content completion list of `ro.sync.contentcompletion.xml.CIAttribute` for the element provided by the current `ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext` context. For example, the `SDFSchemaManagerFilter` checks if the element from the current context is the `table` element and add the `frame` attribute to the `table` list of attributes.

```
/**
 * Filter attributes of the "table" element.
 */
public List<CIAttribute> filterAttributes(List<CIAttribute> attributes,
    WhatAttributesCanGoHereContext context) {
  // If the element from the current context is the 'table' element add the
  // attribute named 'frame' to the list of default content completion proposals
  if (context != null) {
    ContextElement contextElement = context.getParentElement();
    if ("table".equals(contextElement.getQName())) {
      CIAttribute frameAttribute = new CIAttribute();
      frameAttribute.setName("frame");
      frameAttribute.setRequired(false);
      frameAttribute.setFixed(false);
      frameAttribute.setDefaultValue("void");
      if (attributes == null) {
        attributes = new ArrayList<CIAttribute>();
      }
      attributes.add(frameAttribute);
    }
  }
  return attributes;
}
```

The elements that can be inserted in a specific context can be filtered using the `filterElements` method. The `SDFSchemaManagerFilter` uses this method to replace the `td` child element with the `th` element when `header` is the current context element.

```
public List<CIElement> filterElements(List<CIElement> elements,
    WhatElementsCanGoHereContext context) {
  // If the element from the current context is the 'header' element remove the
  // 'td' element from the list of content completion proposals and add the
  // 'th' element.
  if (context != null) {
    Stack<ContextElement> elementStack = context.getElementStack();
    if (elementStack != null) {
      ContextElement contextElement = context.getElementStack().peek();
      if ("header".equals(contextElement.getQName())) {
        if (elements != null) {
          for (Iterator<CIElement> iterator = elements.iterator(); iterator.hasNext();) {
            CIElement element = iterator.next();
            // Remove the 'td' element
            if ("td".equals(element.getQName())) {
              elements.remove(element);
              break;
            }
          }
        } else {
          elements = new ArrayList<CIElement>();
        }
        // Insert the 'th' element in the list of content completion proposals
        CIElement thElement = new SDFElement();
        thElement.setName("th");
        elements.add(thElement);
      }
    }
  } else {
    // If the given context is null then the given list of content completion elements con
    // global elements.
  }
  return elements;
}
```

The elements or attributes values can be filtered using the `filterElementValues` or `filterAttributeValues` methods.

The complete source code of the `SDFSchemaManagerFilter` implementation is found in the Example Files Listings, the Java Files section.

Configuring a Link target element finder

The link target reference finder represents the support for finding references from links which indicate specific elements inside an XML document. This support will only be available if a schema is associated with the document type.

If you do not define a custom link target reference finder, the `DefaultElementLocatorProvider` implementation will be used by default. The interface which should be implemented for a custom link target reference finder is `ro.sync.ecss.extensions.api.link.ElementLocatorProvider`. As an alternative, the `ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider` implementation can also be extended.

The used `ElementLocatorProvider` will be queried for an `ElementLocator` when a link location must be determined (when user clicks on a link). Then, to find the corresponding (linked) element, the obtained `ElementLocator` will be queried for each element from the document.

The `DefaultElementLocatorProvider` implementation

The `DefaultElementLocatorProvider` implementation offers support for the most common types of links:

- links based on ID attribute values
- XPointer element() scheme

The method `getElementLocator` determines what `ElementLocator` should be used. In the default implementation it checks if the link is an XPointer element() scheme otherwise it assumes it is an ID. A non-null `IDTypeVerifier` will always be provided if a schema is associated with the document type.

The `link` string argument is the "anchor" part of the of the URL which is composed from the value of the link property specified for the link element in the CSS.

```
public ElementLocator getElementLocator(IDTypeVerifier idVerifier,
               String link) {
  ElementLocator elementLocator = null;
  try {
    if(link.startsWith("element(")){
      // xpointer element() scheme
      elementLocator = new XPointerElementLocator(idVerifier, link);
    } else {
      // Locate link element by ID
      elementLocator = new IDElementLocator(idVerifier, link);
    }
  } catch (ElementLocatorException e) {
    logger.warn("Exception when create element locator for link: "
        + link + ". Cause: " + e, e);
  }
  return elementLocator;
}
```

The `XPointerElementLocator` implementation

The `XPointerElementLocator` is an implementation of the abstract class `ro.sync.ecss.extensions.api.link.ElementLocator` for links that have one of the following XPointer element() scheme patterns:

element(`elementID`)	locate the element with the specified id
element(`/1/2/3`)	A child sequence appearing alone identifies an element by means of stepwise navigation, which is directed by a sequence of integers separated by slashes (/); each integer n locates the nth child element of the previously located element.
element(`elementID/3/4`)	A child sequence appearing after a `NCName` identifies an element by means of stepwise navigation, starting from the element located by the given name.

The constructor separates the id/integers which are delimited by slashes(/) into a sequence of identifiers (an XPointer path). It will also check that the link has one of the supported patterns of the XPointer element() scheme.

```
public XPointerElementLocator(IDTypeVerifier idVerifier, String link)
                     throws ElementLocatorException {
```

```
  super(link);
  this.idVerifier = idVerifier;

  link = link.substring("element(".length(), link.length() - 1);

  StringTokenizer stringTokenizer = new StringTokenizer(link, "/", false);
  xpointerPath = new String[stringTokenizer.countTokens()];
  int i = 0;
  while (stringTokenizer.hasMoreTokens()) {
    xpointerPath[i] = stringTokenizer.nextToken();
    boolean invalidFormat = false;

    // Empty xpointer component is not supported
    if(xpointerPath[i].length() == 0){
      invalidFormat = true;
    }

    if(i > 0){
      try {
        Integer.parseInt(xpointerPath[i]);
      } catch (NumberFormatException e) {
        invalidFormat = true;
      }
    }

    if(invalidFormat){
      throw new ElementLocatorException(
        "Only the element() scheme is supported when locating XPointer links."
        + "Supported formats: element(elementID), element(/1/2/3),
              element(elemID/2/3/4).");
    }
    i++;
  }

  if(Character.isDigit(xpointerPath[0].charAt(0))){
    // This is the case when xpointer have the following pattern /1/5/7
    xpointerPathDepth = xpointerPath.length;
  } else {
    // This is the case when xpointer starts with an element ID
    xpointerPathDepth = -1;
    startWithElementID  = true;
  }
}
```

The method `startElement` will be invoked at the beginning of every element in the XML document(even when the element is empty). The arguments it takes are

`uri`	the namespace URI, or the empty string if the element has no namespace URI or if namespace processing is disabled
`localName`	the local name of the element
`qName`	the qualified name of the element

`atts` the attributes attached to the element. If there are no attributes, it will be empty.

The method returns `true` if the processed element is found to be the one indicated by the link.

The `XPointerElementLocator` implementation of the `startElement` will update the depth of the current element and keep the index of the element in its parent. If the `xpointerPath` starts with an element ID then the current element ID is verified to match the specified ID. If this is the case the depth of the XPointer is updated taking account of the depth of the current element.

If the XPointer path depth is the same as the current element depth then the kept indices of the current element path are compared to the indices in the XPointer path. If all of them match then the element has been found.

```
public boolean startElement(String uri, String localName,
        String name, Attr[] atts) {
  boolean linkLocated = false;
  // Increase current element document depth
  startElementDepth ++;

  if (endElementDepth != startElementDepth) {
    // The current element is the first child of the parent
    currentElementIndexStack.push(new Integer(1));
  } else {
    // Another element in the parent element
    currentElementIndexStack.push(new Integer(lastIndexInParent + 1));
  }

  if (startWithElementID) {
    // This the case when xpointer path starts with an element ID.
    String xpointerElement = xpointerPath[0];
    for (int i = 0; i < atts.length; i++) {
      if(xpointerElement.equals(atts[i].getValue())){
        if(idVerifier.hasIDType(
            localName, uri, atts[i].getQName(), atts[i].getNamespace())){
          xpointerPathDepth = startElementDepth + xpointerPath.length - 1;
          break;
        }
      }
    }
  }

  if (xpointerPathDepth == startElementDepth){
    // check if xpointer path matches with the current element path
    linkLocated = true;
    try {
      int xpointerIdx = xpointerPath.length - 1;
      int stackIdx = currentElementIndexStack.size() - 1;
      int stopIdx = startWithElementID ? 1 : 0;
      while (xpointerIdx >= stopIdx && stackIdx >= 0) {
        int xpointerIndex = Integer.parseInt(xpointerPath[xpointerIdx]);
        int currentElementIndex =
          ((Integer)currentElementIndexStack.get(stackIdx)).intValue();
        if(xpointerIndex != currentElementIndex) {
          linkLocated = false;
          break;
```

```
        }

        xpointerIdx--;
        stackIdx--;
      }

    } catch (NumberFormatException e) {
      logger.warn(e,e);
    }
  }
  return linkLocated;
}
```

The method `endElement` will be invoked at the end of every element in the XML document (even when the element is empty).

The `XPointerElementLocator` implementation of the `endElement` updates the depth of the current element path and the index of the element in its parent.

```
public void endElement(String uri, String localName, String name) {
  endElementDepth = startElementDepth;
  startElementDepth --;
  lastIndexInParent = ((Integer)currentElementIndexStack.pop()).intValue();
}
```

The `IDElementLocator` implementation

The `IDElementLocator` is an implementation of the abstract class `ro.sync.ecss.extensions.api.link.ElementLocator` for links that use an **id**.

The constructor only assigns field values and the method `endElement` is empty for this implementation.

The method `startElement` checks each of the element's attribute values and when one matches the link, it considers the element found if one of the following conditions is satisfied:

- the qualified name of the attribute is `xml:id`
- the attribute is of type ID

The type of the attribute is checked with the help of the method `IDTypeVerifier.hasIDType`.

```
public boolean startElement(String uri, String localName,
        String name, Attr[] atts) {
  boolean elementFound = false;
  for (int i = 0; i < atts.length; i++) {
    if (link.equals(atts[i].getValue())) {
      if("xml:id".equals(atts[i].getQName())) {
        // xml:id attribute
        elementFound = true;
      } else {
        // check if attribute has ID type
        String attrLocalName =
          ExtensionUtil.getLocalName(atts[i].getQName());
        String attrUri = atts[i].getNamespace();
        if (idVerifier.hasIDType(localName, uri, attrLocalName, attrUri)) {
```

```
          elementFound = true;
        }
      }
    }
  }

  return elementFound;
}
```

Creating a customized link target reference finder

If you need to create a custom link target reference finder you can do so by following these steps.

Create the class which will implement the `ro.sync.ecss.extensions.api.link.ElementLocatorProvider` interface. As an alternative, your class could extend `ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider`, the default implementation.

As a start point you can use the source code of the `DefaultElementLocatorProvider` implementation which is found in the Example Files Listings, the Java Files section. There you will also find the implementations for `XPointerElementLocator` and `IDElementLocator` .

Configuring a custom Drag and Drop listener

You can add your own drag and drop listener implementation of `ro.sync.ecss.extensions.api.DnDHandler`. You can choose from three interfaces to implement depending on whether you are using the framework with the <oXygen/> Eclipse plugin or the standalone version or if you want to add the handler for the Text or Author pages.

Table 2. Interfaces for the DnD listener

Interface	Description
`ro.sync.exml.editor.xmleditor.pageauthor.AuthorCustomDnDHandler`	Receives callbacks from the <oXygen/> standalone application for Drag And Drop in Author
`com.oxygenxml.editor.editors.author.AuthorDnDListener`	Receives callbacks from the <oXygen/> Eclipse plugin for Drag And Drop in Author
`com.oxygenxml.editor.editors.TextDnDListener`	Receives callbacks from the <oXygen/> Eclipse plugin for Drag And Drop in Text

Configuring a References Resolver

We need to provide a handler for resolving references and obtain the content they refer. In our case the element which has references is **ref** and the attribute indicating the referred resource is **location**. We will have to implement a Java extension class for obtaining the referred resources.

Create the class `simple.documentation.framework.ReferencesResolver`. This class must implement the `ro.sync.ecss.extensions.api.AuthorReferenceResolver` interface.

```
import ro.sync.ecss.extensions.api.AuthorReferenceResolver;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;
import ro.sync.ecss.extensions.api.node.AuthorNode;
```

```
public class ReferencesResolver
      implements AuthorReferenceResolver {
```

The method `hasReferences` verifies if the handler considers the node to have references. It takes as argument an `AuthorNode` that represents the node which will be verified. The method will return `true` if the node is considered to have references. In our case, to be a reference the node must be an element with the name *ref* and it must have an attribute named *location*.

```
public boolean hasReferences(AuthorNode node) {
  boolean hasReferences = false;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      hasReferences = attrValue != null;
    }
  }
  return hasReferences;
}
```

The method `getDisplayName` returns the display name of the node that contains the expanded referred content. It takes as argument an `AuthorNode` that represents the node for which the display name is needed. The referred content engine will ask this `AuthorReferenceResolver` implementation what is the display name for each node which is considered a reference. In our case the display name is the value of the *location* attribute from the *ref* element.

```
public String getDisplayName(AuthorNode node) {
  String displayName = "ref-fragment";
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        displayName = attrValue.getValue();
      }
    }
  }
  return displayName;
}
```

The method `resolveReference` resolves the reference of the node and returns a `SAXSource` with the parser and the parser's input source. It takes as arguments an `AuthorNode` that represents the node for which the reference needs resolving, the `systemID` of the node, the `AuthorAccess` with access methods to the Author data model and a SAX `EntityResolver` which resolves resources that are already opened in another editor or resolve resources through the XML catalog. In our implementation we need to resolve the reference relative to the `systemID`, and create a parser and an input source over the resolved reference.

```
public SAXSource resolveReference(
    AuthorNode node,
    String systemID,
    AuthorAccess authorAccess,
    EntityResolver  entityResolver) {
  SAXSource saxSource = null;

  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
```

```
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(new URL(systemID),
              authorAccess.correctURL(attrStringVal));

          InputSource inputSource = entityResolver.resolveEntity(null,
              absoluteUrl.toString());
          if(inputSource == null) {
            inputSource = new InputSource(absoluteUrl.toString());
          }

          XMLReader xmlReader = authorAccess.newNonValidatingXMLReader();
          xmlReader.setEntityResolver(entityResolver);

          saxSource = new SAXSource(xmlReader, inputSource);
        } catch (MalformedURLException e) {
          logger.error(e, e);
        } catch (SAXException e) {
          logger.error(e, e);
        } catch (IOException e) {
          logger.error(e, e);
        }
      }
    }
  }

  return saxSource;
}
```

The method `getReferenceUniqueID`should return an unique identifier for the node reference. The unique identifier is used to avoid resolving the references recursively. It takes as argument an `AuthorNode` that represents the node with the reference. In our implementation the unique identifier is the value of the *location* attribute from the *ref* element.

```
public String getDisplayName(AuthorNode node) {
  String displayName = "ref-fragment";
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        displayName = attrValue.getValue();
      }
    }
  }
  return displayName;
}
```

The method `getReferenceSystemID`should return the `systemID` of the referred content. It takes as arguments an `AuthorNode` that represents the node with the reference and the `AuthorAccess` with access methods to the

Author data model. In our implementation we use the value of the *location* attribute from the *ref* element and resolve it relatively to the XML base URL of the node.

```
public String getReferenceSystemID(AuthorNode node,
                                   AuthorAccess authorAccess) {
  String systemID = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(node.getXMLBaseURL(),
              authorAccess.correctURL(attrStringVal));
          systemID = absoluteUrl.toString();
        } catch (MalformedURLException e) {
          logger.error(e, e);
        }
      }
    }
  }
  return systemID;
}
```

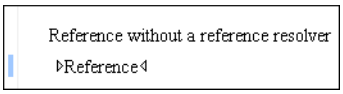
The complete source code of our implementation is found in the Example Files Listings, the Java Files section.

In the listing below, the XML document contains the **ref** element:

```
<ref location="referred.xml">Reference</ref>
```

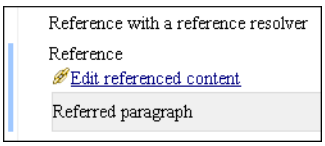
When no reference resolver is specified, the reference has the following layout:

Figure 33. Reference with no specified reference resolver



When the above implementation is configured, the reference has the expected layout:

Figure 34. Reference with reference resolver



Configuring CSS Styles Filter

You can modify the CSS styles for each `ro.sync.ecss.extensions.api.node.AuthorNode` rendered in the Author page using an implementation of `ro.sync.ecss.extensions.api.StylesFilter` You can implement the various callbacks of the interface either by returning the default value given by <oXygen/> or by contributing to the value. The received styles `ro.sync.ecss.css.Styles` can be processed and values can be

overwritten with your own. For example you can override the `KEY_BACKGROUND_COLOR` style to return your own implementation of `ro.sync.exml.view.graphics.Color` or override the `KEY_FONT` style to return your own implementation of `ro.sync.exml.view.graphics.Font`.

For instance in our simple document example the filter can change the value of the `KEY_FONT` property for the `table` element:

```
package simple.documentation.framework;

import ro.sync.ecss.css.Styles;
import ro.sync.ecss.extensions.api.StylesFilter;
import ro.sync.ecss.extensions.api.node.AuthorNode;
import ro.sync.exml.view.graphics.Font;

public class SDFStylesFilter implements StylesFilter {

     public Styles filter(Styles styles, AuthorNode authorNode) {
         if (AuthorNode.NODE_TYPE_ELEMENT == authorNode.getType()
           && "table".equals(authorNode.getName())) {
           styles.setProperty(Styles.KEY_FONT, new Font(null, Font.BOLD, 12));
         }
         return styles;
     }
}
```

Configuring a Table Column Width Provider

In our documentation framework the `table` element and the table columns can have specified widths. In order for these widths to be considered by <oXygen/> Author we need to provide the means to determine them. As explained in the Styling the Table Element section which describes the CSS properties needed for defining a table, if we use the table element attribute **width** <oXygen/> can determine the table width automatically. In our example the table has `col` elements with **width** attributes that are not recognized by default. We will need to implement a Java extension class for determining the column widths.

Create the class `simple.documentation.framework.TableColumnWidthProvider`. This class must implement the `ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider` interface.

```
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperationException;
import ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider;
import ro.sync.ecss.extensions.api.WidthRepresentation;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableColumnWidthProvider
      implements AuthorTableColumnWidthProvider {
```

The method `init` is taking as argument the `AuthorElement` that represents the XML `table` element. In our case the column widths are specified in `col` elements from the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
 public void init(AuthorElement tableElement) {
  this.tableElement = tableElement;
  AuthorElement[] colChildren = tableElement.getElementsByLocalName("customcol");
```

```
  if (colChildren != null && colChildren.length > 0) {
   for (int i = 0; i < colChildren.length; i++) {
    AuthorElement colChild = colChildren[i];
    if (i == 0) {
     colsStartOffset = colChild.getStartOffset();
    }
    if (i == colChildren.length - 1) {
     colsEndOffset = colChild.getEndOffset();
    }
    // Determine the 'width' for this col.
    AttrValue colWidthAttribute = colChild.getAttribute("width");
    String colWidth = null;
    if (colWidthAttribute != null) {
     colWidth = colWidthAttribute.getValue();
     // Add WidthRepresentation objects for the columns this 'customcol' specification
     // spans over.
     colWidthSpecs.add(new WidthRepresentation(colWidth, true));
    }
   }
  }
 }
```

The method `isTableAcceptingWidth` should check if the table cells are `td`.

```
public boolean isTableAcceptingWidth(String tableCellsTagName) {
  return "td".equals(tableCellsTagName);
}
```

The method `isTableAndColumnsResizable` should check if the table cells are `td`. This method determines if the table and columns can be resized by dragging with the mouse the edge of a column.

```
public boolean isTableAndColumnsResizable(String tableCellsTagName) {
  return "td".equals(tableCellsTagName);
}
```

The methods `getTableWidth` and `getCellWidth` are used for determining the table width and the column width. The table layout engine will ask this `AuthorTableColumnWidthProvider` implementation what is the table width for each table element and the cell width for each cell element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and just parses the value of the **width** attribute. The methods must return `null` for the tables/cells that do not have a specified width.

```
 public WidthRepresentation getTableWidth(String tableCellsTagName) {
  WidthRepresentation toReturn = null;
  if (tableElement != null && "td".equals(tableCellsTagName)) {
   AttrValue widthAttr = tableElement.getAttribute("width");
   if (widthAttr != null) {
    String width = widthAttr.getValue();
    if (width != null) {
     toReturn = new WidthRepresentation(width, true);
    }
   }
  }
  return toReturn;
 }
```

```
public List<WidthRepresentation> getCellWidth(AuthorElement cellElement, int colNumberSta
  int colSpan) {
 List<WidthRepresentation> toReturn = null;
 int size = colWidthSpecs.size();
 if (size >= colNumberStart && size >= colNumberStart + colSpan) {
  toReturn = new ArrayList<WidthRepresentation>(colSpan);
  for (int i = colNumberStart; i < colNumberStart + colSpan; i ++) {
   // Add the column widths
   toReturn.add(colWidthSpecs.get(i));
  }
 }
 return toReturn;
}
```

The methods `commitTableWidthModification` and `commitColumnWidthModifications` are used for committing changes made to the width of the table or its columns when using the mouse drag gestures.

```
public void commitTableWidthModification(AuthorDocumentController authorDocumentControlle
  int newTableWidth, String tableCellsTagName) throws AuthorOperationException {
 if ("td".equals(tableCellsTagName)) {
  if (newTableWidth > 0) {
   if (tableElement != null) {
    String newWidth = String.valueOf(newTableWidth);

    authorDocumentController.setAttribute(
      "width",
      new AttrValue(newWidth),
      tableElement);
   } else {
    throw new AuthorOperationException("Cannot find the element representing the table.")
   }
  }
 }
}

public void commitColumnWidthModifications(AuthorDocumentController authorDocumentControll
  WidthRepresentation[] colWidths, String tableCellsTagName) throws AuthorOperationExcept
 if ("td".equals(tableCellsTagName)) {
  if (colWidths != null && tableElement != null) {
   if (colsStartOffset >= 0 && colsEndOffset >= 0 && colsStartOffset < colsEndOffset) {
    authorDocumentController.delete(colsStartOffset,
      colsEndOffset);
   }
   String xmlFragment = createXMLFragment(colWidths);
   int offset = -1;
   AuthorElement[] header = tableElement.getElementsByLocalName("header");
   if (header != null && header.length > 0) {
    // Insert the cols elements before the 'header' element
    offset = header[0].getStartOffset();
   }
   if (offset == -1) {
    throw new AuthorOperationException("No valid offset to insert the columns width speci
   }
   authorDocumentController.insertXMLFragment(xmlFragment, offset);
```

```
   }
  }
 }

 private String createXMLFragment(WidthRepresentation[] widthRepresentations) {
  StringBuffer fragment = new StringBuffer();
  String ns = tableElement.getNamespace();
  for (int i = 0; i < widthRepresentations.length; i++) {
   WidthRepresentation width = widthRepresentations[i];
   fragment.append("<customcol");
   String strRepresentation = width.getWidthRepresentation();
   if (strRepresentation != null) {
    fragment.append(" width=\"" + width.getWidthRepresentation() + "\"");
   }
   if (ns != null && ns.length() > 0) {
    fragment.append(" xmlns=\"" + ns + "\"");
   }
   fragment.append("/>");
  }
  return fragment.toString();
 }
```

The following three methods are used to determine what type of column width specifications the table column width provider support. In our case all types of specifications are allowed:

```
 public boolean isAcceptingFixedColumnWidths(String tableCellsTagName) {
  return true;
 }

 public boolean isAcceptingPercentageColumnWidths(String tableCellsTagName) {
  return true;
 }

 public boolean isAcceptingProportionalColumnWidths(String tableCellsTagName) {
  return true;
 }
```

The complete source code of our implementation is found in the Example Files Listings, the Java Files section.

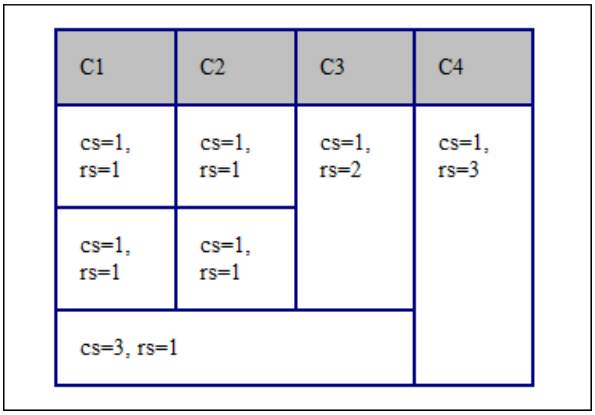
In the listing below, the XML document contains the table element:

```
<table width="300">
    <customcol width="50.0px"/>
    <customcol width="1*"/>
    <customcol width="2*"/>
    <customcol width="20%"/>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
```

```
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
        <td row_span="2">cs=1, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

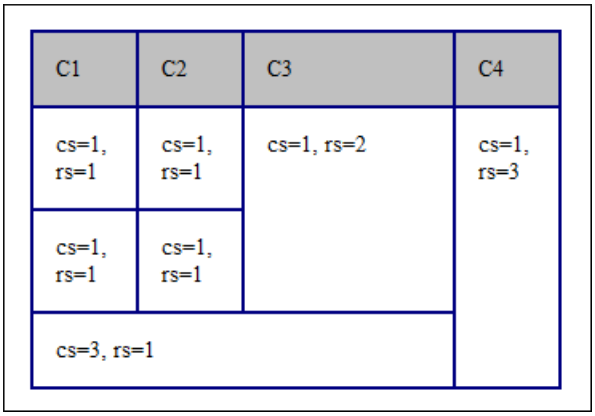
When no table column width provider is specified, the table has the following layout:

Figure 35. Table layout when no column width provider is specified



When the above implementation is configured, the table has the correct layout:

Figure 36. Columns with custom widths



Configuring a Table Cell Span Provider

In our documentation framework the `table` element can have cells that span over multiple columns and rows. As explained in the Styling the Table Element section which describes the CSS properties needed for defining a table, we

need to indicate <oXygen/> Author a method to determine the cell spanning. If we use the cell element attributes **rowspan** and **colspan** or **rows** and **cols**, <oXygen/> can determine the cell spanning automatically. In our example the `td` element uses the attributes **row_span** and **column_span** that are not recognized by default. We will need to implement a Java extension class for defining the cell spanning.

Create the class `simple.documentation.framework.TableCellSpanProvider`. This class must implement the `ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider` interface.

```
import ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;

public class TableCellSpanProvider
      implements AuthorTableCellSpanProvider {
```

The method `init` is taking as argument the `AuthorElement` that represents the XML `table` element. In our case the cell span is specified for each of the cells so we leave this method empty. However there are cases like the table CALS model when the cell spanning is specified in the `table` element. In such cases you must collect the span information by analyzing the `table` element.

```
public void init(AuthorElement table) {
}
```

The method `getColSpan` is taking as argument the table cell. The table layout engine will ask this `AuthorTableSpanSupport` implementation what is the column span and the row span for each XML element from the table that was marked as cell in the CSS using the property `display:table-cell`. The implementation is simple and just parses the value of **column_span** attribute. The method must return `null` for all the cells that do not change the span specification.

```
public Integer getColSpan(AuthorElement cell) {
  Integer colSpan = null;

  AttrValue attrValue = cell.getAttribute("column_span");
  if(attrValue != null) {
    // The attribute was found.
    String cs = attrValue.getValue();
    if(cs != null) {
      try {
        colSpan = new Integer(cs);
      } catch (NumberFormatException ex) {
        // The attribute value was not a number.
      }
    }
  }
  return colSpan;
}
```

The row span is determined in a similar manner:

```
public Integer getRowSpan(AuthorElement cell) {
  Integer rowSpan = null;

  AttrValue attrValue = cell.getAttribute("row_span");
  if(attrValue != null) {
    // The attribute was found.
```

```
    String rs = attrValue.getValue();
    if(rs != null) {
      try {
        rowSpan = new Integer(rs);
      } catch (NumberFormatException ex) {
        // The attribute value was not a number.
      }
    }
  }
  return rowSpan;
}
```

The method `hasColumnSpecifications` always returns `true` considering column specifications always available.

```
public boolean hasColumnSpecifications(AuthorElement tableElement) {
  return true;
}
```

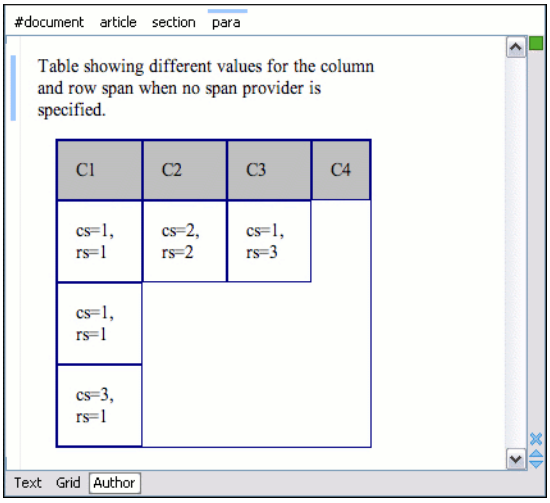
The complete source code of our implementation is found in the Example Files Listings, the Java Files section.

In the listing below, the XML document contains the table element:

```
<table>
    <header>
        <td>C1</td>
        <td>C2</td>
        <td>C3</td>
        <td>C4</td>
    </header>
    <tr>
        <td>cs=1, rs=1</td>
        <td column_span="2" row_span="2">cs=2, rs=2</td>
        <td row_span="3">cs=1, rs=3</td>
    </tr>
    <tr>
        <td>cs=1, rs=1</td>
    </tr>
    <tr>
        <td column_span="3">cs=3, rs=1</td>
    </tr>
</table>
```

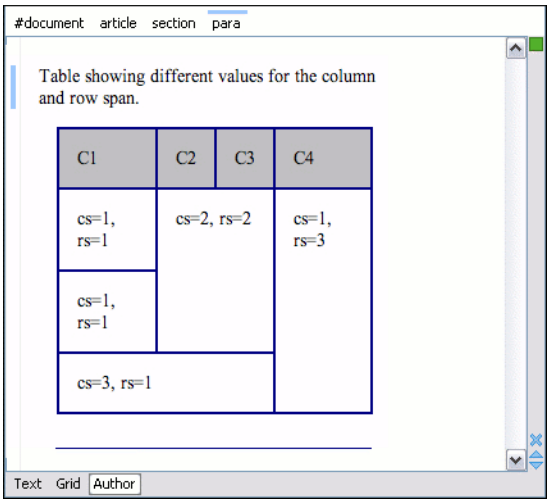
When no table cell span provider is specified, the table has the following layout:

Figure 37. Table layout when no cell span provider is specified



When the above implementation is configured, the table has the correct layout:

Figure 38. Cells spanning multiple rows and columns.



Configuring an Unique Attributes Recognizer

The `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` interface can be implemented if you want to provide for your framework the following features:

Automatic ID generation

You can automatically generate unique IDs for newly inserted elements. Implementations are already available for the DITA and Docbook frameworks. The following methods can be implemented to accomplish this:

```
/**
 * Assign unique IDs between a start
 * and an end offset in the document
 * @param startOffset Start offset
 * @param endOffset End offset
 */
```

```
void assignUniqueIDs(int startOffset, int endOffset);

/**
 * @return true if auto
 */
boolean isAutoIDGenerationActive();
```

Avoiding copying unique attributes when "Split" is called inside an element

You can split the current block element by pressing the "Enter" key and then choosing "Split". This is a very useful way to create new paragraphs. All attributes are by default copied on the new element but if those attributes are IDs you sometimes want to avoid creating validation errors in the editor. Implementing the following method, you can decide whether an attribute should be copied or not during the split:

```
/**
 * Check if the attribute specified by QName can
 * be considered as a valid attribute to copy
 * when the element is split.
 *
 * @param attrQName The attribute qualified name
 * @param element The element
 * @return true if the attribute should be copied
 * when Split is performed.
 */
boolean copyAttributeOnSplit(String attrQName,
            AuthorElement element);
```

Tip

The `ro.sync.ecss.extensions.commons.id.DefaultUniqueAttributesRecognizer` class is an implementation of the interface which can be extended by your customization to provide easy assignation of IDs in your framework. You can also check out the DITA and Docbook implementations of `ro.sync.ecss.extensions.api.UniqueAttributesRecognizer` to see how they were implemented and connected to the extensions bundle.

Customizing the default CSS of a document type

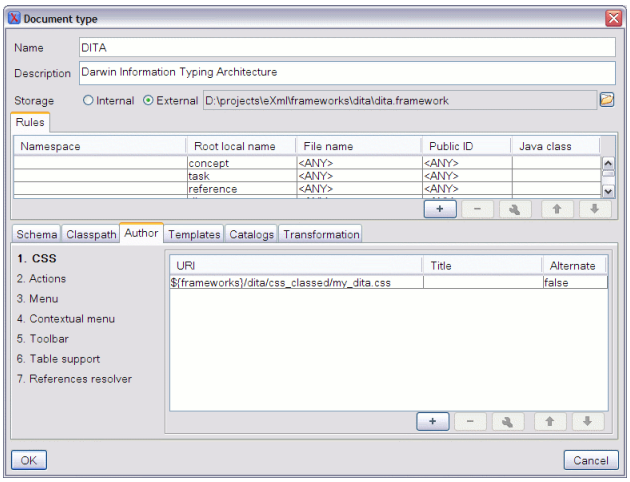
The easiest way of customizing the default CSS stylesheet of a document type is to create a new CSS stylesheet in the same folder as the customized one, import the customized CSS stylesheet and set the new stylesheet as the default CSS of the document type. For example let us customize the default CSS for DITA documents by changing the background color of the *task* and *topic* elements to red. First we create a new CSS stylesheet called *my_dita.css* in the folder *${frameworks}/dita/css_classed* where the default stylesheet called *dita.css* is located. *${frameworks}* is the subfolder *frameworks* of the Oxygen XML Editor. The new stylesheet *my_dita.css* contains:

```
@import "dita.css";

task, topic{
    background-color:red;
}
```

To set the new stylesheet as the default CSS stylesheet for DITA documents first open the Document Type Association preferences panel from menu Options → Preferences+Document Type Association Select the DITA document type and start editing it by pressing the Edit button. The user role must be set to *Developer* otherwise a warning is displayed and a duplicate copy of the DITA document type is created and edited. This check makes sure that regular content authors who just edit the content of XML documents do not accidentally modify the document type. In the Author tab of the document type edit dialog change the URI of the default CSS stylesheet from *${frameworks}/dita/css_classed/dita.css* to *${frameworks}/dita/css_classed/my_dita.css*.

Figure 39. Set the location of the default CSS stylesheet



Press OK in all the dialogs to validate the changes. Now you can start editing DITA documents based on the new CSS stylesheet. You can edit the new CSS stylesheet itself at any time and see the effects on rendering DITA XML documents in the Author mode by running the *Refresh* action available on the Author toolbar and on the DITA menu.

Document type sharing

A document type can be shared between authors in two ways:

- save the document type at global level in the Document Type Association panel and distribute a zip file that includes all the files of the document type (CSS stylesheets, jar files with custom actions, etc). Each user will unzip the zip file in a subdirectory of the ${frameworks} directory and will restart the application for adding the new document type to the list of the Document Type Association panel
- save the document type at project level in the Document Type Association panel and distribute both the Oxygen project file and the files of the document type (CSS stylesheets, jar files with custom actions, etc). Each user will copy the files of the document type in the subdirectory of the ${frameworks} directory that corresponds to the document type and will load the Oxygen project file in the *Project* view.

CSS support in <oXygen/> Author

CSS 2.1 features

Supported selectors

The following CSS level 2.1 selectors are supported by the <oXygen/> Author:

Table 3. Supported CSS 2.1 selectors

Expression	Name	Description/Example
*	Universal selector	Matches any element
E	Type selector	Matches any E element (i.e an element with the local name E)
E F	Descendant selector	Matches any F element that is a descendant of an E element.
E > F	Child selectors	Matches any F element that is a child of an element E.
E:first-child	The :first-child pseudo-class	Matches element E when E is the first child of its parent.
E:lang(c)	The :lang() pseudo-class	Matches element of type E if it is in (human) language c (the document language specifies how language is determined).
E + F	Adjacent selector	Matches any F element immediately preceded by a sibling element E.
E[foo]	Attribute selector	Matches any E element with the "foo" attribute set (whatever the value).
E[foo="warning"]	Attribute selector	Matches any E element whose "foo" attribute value is exactly equal to "warning".
E[foo~="warning"]	Attribute selector	Matches any E element whose "foo" attribute value is a list of space-separated values, one of which is exactly equal to "warning".
E[lang|="en"]	Attribute selector	Matches any E element whose "lang" attribute has a hyphen-separated list of values beginning (from the left) with "en".
E:before and E:after	Pseudo elements	The ':before' and ':after' pseudo-elements can be used to insert generated content before or after an element's content.

Unsupported selectors

The following CSS level 2.1 selectors are **not supported** by the <oXygen/> Author:

Table 4. Unsupported CSS 2.1 selectors

Expression	Name	Description/Example
E#myid	ID selectors	Matches any E element with ID equal to "myid".
E:link, E:visited	The link pseudo-class	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited).
E:active, E:hover, E:focus	The dynamic pseudo-classes	Matches E during certain user actions.
E:first-line	The :first-line pseudo-class	The :first-line pseudo-element applies special styles to the contents of the first formatted line of a paragraph.
E:first-letter	The :first-letter pseudo-class	The :first-letter pseudo-element must select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line. The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects.

Properties Support Table

All the properties belonging to the *aural* and *paged* categories are **not supported** in <oXygen/> Author. The properties from the table below belong to the *visual* category.

Table 5. CSS Level 2.1 Properties and their support in <oXygen/> Author

Name	Supported Values	Not Supported Values
'background-attachment'		ALL
'background-color'	<color> | inherit	transparent
'background-image'		ALL
'background-position'		ALL
'background-repeat'		ALL
'background'		ALL
'border-collapse'		ALL
'border-color'	<color> | inherit	transparent
'border-spacing'		ALL
'border-style'	<border-style> | inherit	
'border-top' 'border-right' 'border-bottom' 'border-left'	[<border-width> || <border-style> || 'border-top-color'] | inherit	
'border-top-color' 'border-right-color' 'border-bottom-color' 'border-left-color'	<color> | inherit	transparent
'border-top-style' 'border-right-style' 'border-bottom-style' 'border-left-style'	<border-style> | inherit	
'border-top-width' 'border-right-width' 'border-bottom-width' 'border-left-width'	<border-width> | inherit	
'border-width'	<border-width> | inherit	
'border'	[<border-width> || <border-style> || 'border-top-color'] | inherit	
'bottom'		ALL
'caption-side'		ALL
'clear'		ALL
'clip'		ALL
'color'	<color> | inherit	
'content'	normal | none | [<string> | <uri> | <counter> | attr(<identifier>) | open-quote | close-quote]+ | inherit	no-open-quote | no-close-quote
'counter-increment'	[<identifier> <integer> ?]+ | none | inherit	
'counter-reset'	[<identifier> <integer> ?]+ | none | inherit	
'cursor'		ALL
'direction'	ltr	rtl | inherit
'display'	inline | block | list-item | table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit	run-in | inline-block | inline-table - considered block
'empty-cells'	show | hide | inherit	
'float'		ALL
'font-family'	[[<family-name> | <generic-family>] [, <family-name> | <generic-family>]*] | inherit	

Name	Supported Values	Not Supported Values
'font-size'	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	
'font-style'	normal | italic | oblique | inherit	
'font-variant'		ALL
'font-weight'	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	
'font'	[['font-style' || 'font-weight']? 'font-size' [/ 'line-height']? 'font-family'] | inherit	'font-variant' 'line-height' caption | icon | menu | message-box | small-caption | status-bar
'height'		ALL
'left'		ALL
'letter-spacing'		ALL
'line-height'	normal | <number> | <length> | <percentage> | inherit	
'list-style-image'		ALL
'list-style-position'		ALL
'list-style-type'	disc | circle | square | decimal | lower-roman | upper-roman | lower-latin | upper-latin | lower-alpha | upper-alpha | none | inherit	lower-greek | armenian | georgian
'list-style'	['list-style-type'] | inherit	'list-style-position' || 'list-style-image'
'margin-right' 'margin-left'	<margin-width> | inherit	
'margin-top' 'margin-bottom'	<margin-width> | inherit	
'margin'	<margin-width> | inherit	
'max-height'		ALL
'max-width'	<length> | <percentage> | none | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'min-height'		ALL
'min-width'	<length> | <percentage> | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'outline-color'		ALL
'outline-style'		ALL
'outline-width'		ALL
'outline'		ALL
'overflow'		ALL
'padding-top' 'padding-right' 'padding-bottom' 'padding-left'	<padding-width> | inherit	
'padding'	<padding-width> | inherit	
'position'		ALL

Name	Supported Values	Not Supported Values
'quotes'		ALL
'right'		ALL
'table-layout'	auto	fixed | inherit
'text-align'	left | right | center | inherit	justify
'text-decoration'	none | [underline || overline || line-through] | inherit	blink
'text-indent'		ALL
'text-transform'		ALL
'top'		ALL
'unicode-bidi'		ALL
'vertical-align'	baseline | sub | super | top | text-top | middle | bottom | text-bottom | inherit	<percentage> | <length>
'visibility'	visible | hidden | inherit	collapse
'white-space'	normal | pre | nowrap | pre-wrap | pre-line	
'width'	<length> | <percentage> | auto | inherit - supported for block-level and replaced elements, e.g. images, tables, table cells.	
'word-spacing'		ALL
'z-index'		ALL

<oXygen/> CSS Extensions

Media Type oxygen

The style sheets can specify how a document is to be presented on different media: on the screen, on paper, speech synthesiser, etc. You can specify that some of the features of your CSS stylesheet should be taken into account only in the <oXygen/> Author and ignored in the rest. This can be accomplished by using the media type oxygen.

For instance using the following CSS:

```
b{
 font-weight:bold;
 display:inline;
}

@media oxygen{
 b{
  text-decoration:underline;
 }
}
```

would make a text bold if the document was opened in a web browser who does not recognize `@media oxygen` and bold and underlined in <oXygen/> Author.

You can use this media type to group specific <oXygen/> CSS features and also to hide them when opening the documents with other viewers.

Supported Features from CSS Level 3

Namespace Selectors

In the current CSS 2.1 standard the element selectors are ignoring the namespaces of the elements they are matching. Only the local name of the elements are considered in the selector matching process.

<oXygen/> Author uses a different approach similar to the CSS Level 3 specification. If the element name from the CSS selector is not preceded by a namespace prefix it is considered to match an element with the same local name as the selector value and ANY namespace, otherwise the element must match both the local name and the namespace.

In CSS up to version 2.1 the name tokens from the selectors are matching all elements from ANY namespace that have the same local name. Example:

```
<x:b xmlns:x="ns_x"/>
<y:b xmlns:y="ns_y"/>
```

Are both matched by the rule:

```
b {font-weight:bold}
```

Starting with CSS Level 3 you can create selectors that are namespace aware.

Example 5. Defining both prefixed namespaces and the default namespace

Given the namespace declarations:

```
@namespace sync "http://sync.example.org";
@namespace "http://example.com/foo";
```

In a context where the default namespace applies:

sync|A represents the name A in the `http://sync.example.org` namespace.

|B represents the name B that belongs to NO NAMESPACE.

*|C represents the name C in ANY namespace, including NO NAMESPACE.

D represents the name D in the `http://example.com/foo` namespace.

Example 6. Defining only prefixed namespaces

Given the namespace declaration:

```
@namespace sync "http://sync.example.org";
```

Then:

sync|A represents the name A in the `http://sync.example.org` namespace.

|B represents the name B that belongs to NO NAMESPACE.

*|C represents the name C in ANY namespace, including NO NAMESPACE.

D represents the name D in ANY namespace, including NO NAMESPACE.

The `attr()` function: Properties Values Collected from the Edited Document.

In CSS Level 2.1 you may collect attribute values and use them as content *only* for the pseudo elements. For instance the :before pseudo-element can be used to insert some content before an element. This is valid in CSS 2.1:

```
title:before{
  content: "Title id=(" attr(id) ")";
}
```

If the `title` element from the XML document is:

```
<title id="title12">My title.</title>
```

Then the title will be displayed as:

`Title id=(title12) My title.`

In <oXygen/> Author the use of `attr()` function is not available only for the `content` property but for any other property. This is similar to the CSS Level 3 working draft: http://www.w3.org/TR/2006/WD-css3-values-20060919/#functional. The arguments of the function are:

`attr(`*`attribute_name, attribute_type, default_value`*`);`

`attribute_name` ;
`attribute_type` ;
`default_value` ;

attribute_name — The name of the attribute. This argument is required.

attribute_type — The type of the attribute. This argument is optional. If it is missing the type of the argument is considered `string`. This argument indicates what is the meaning of the attribute value and helps to perform conversions of this value. <oXygen/> Author accepts one of the following types:

- color — The value represents a color. The attribute may specify a color in different formats. <oXygen/> Author supports colors specified either by name: `red`, `blue`, `green`, etc. or as an RGB hexadecimal value `#FFEEFF`.
- url — The value is an URL pointing to a media object. <oXygen/> Author supports only images. The attribute value can be a complete URL, or a relative one to the XML document. Please note that this URL is also resolved through the catalog resolver.
- integer — The value must be interpreted as an integer.
- number — The value must be interpreted as a float number.
- length — The value must be interpreted as an integer.
- percentage — The value must be interpreted relative to another value (length, size) expressed in percents.
- em — The value must be interpreted as a size. 1 em is equal to the *font-size* of the relevant font.

	ex	The value must be interpreted as a size. 1 ex is equal to the *height* of the **x** character of the relevant font.
	px	The value must be interpreted as a size expressed in pixels relative to the viewing device.
	mm	The value must be interpreted as a size expressed in millimeters.
	cm	The value must be interpreted as a size expressed in centimeters.
	in	The value must be interpreted as a size expressed in inches. 1 inch is equal to 2.54 centimeters.
	pt	The value must be interpreted as a size expressed in points. The points used by CSS2 are equal to 1/72th of an inch.
	pc	The value must be interpreted as a size expressed in picas. 1 pica is equal to 12 points.
default_value	This argument specifies a value that is used by default if the attribute value is missing. This argument is optional.	

Example 7. Usage samples for the attr() function

Consider the following XML instance:

```
<sample>
    <para bg_color="#AAAAFF">Blue paragraph.</para>
    <para bg_color="red">Red paragraph.</para>
    <para bg_color="red" font_size="2">Red paragraph with large font.</para>
    <para bg_color="#00AA00" font_size="0.8" space="4">
        Green paragraph with small font and margin.</para>
</sample>
```

The `para` elements have `bg_color` attributes with RGB color values like `#AAAAFF`. We can use the `attr()` function to change the elements appearance in the editor based on the value of this attribute:

```
background-color:attr(bg_color, color);
```

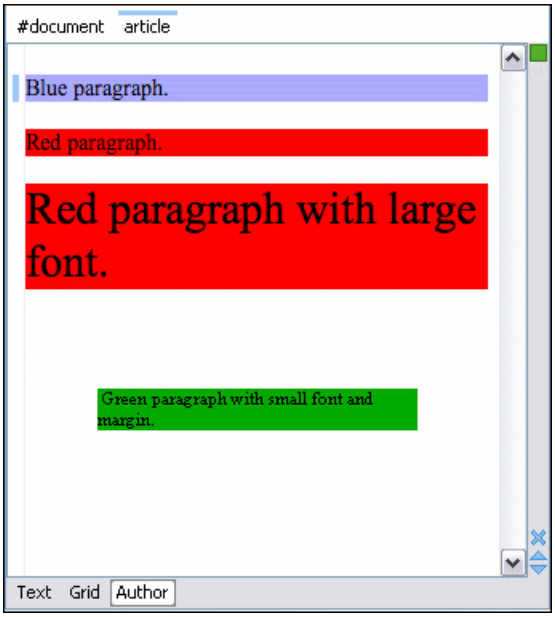
The attribute `font_size` represents the font size in *em* units. We can use this value to change the style of the element:

```
font-size:attr(font_size, em);
```

The complete CSS rule is:

```
para{
 display:block;
 background-color:attr(bg_color, color);
 font-size:attr(font_size, em);
 margin:attr(space, em);
}
```

The document is rendered as:



Additional Custom Selectors

Oxygen Author provides support for selecting additional types of nodes. These custom selectors apply to: *document*, *doctype sections*, *processing-instructions*, *comments*, *CDATA sections*, and *entities*. In order for the custom selectors

to work in your CSS files you will have to declare the Author extensions namespace at the beginning of the stylesheet documents:

```
@namespace oxy url('http://www.oxygenxml.com/extensions/author');
```

Example rules:

- *document*

```
oxy|document {
    display:block;
}
```

- *doctype sections*

```
oxy|doctype {
    display:block;
    color:blue;
    background-color:transparent;
}
```

- *processing-instructions*

```
oxy|processing-instruction {
    display:block;
    color:purple;
    background-color:transparent;
}
```

- *comments*

```
oxy|comment {
    display:block;
    color:green;
    background-color:transparent;
}
```

- *CDATA sections*

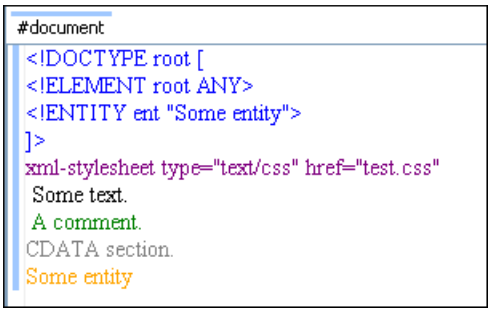
```
oxy|cdata{
    display:block;
    color:gray;
    background-color:transparent;
}
```

- *entities*

```
oxy|entity {
    display:morph;
    editable:false;
    color:orange;
```

```
    background-color:transparent;
}
```

A sample document rendered using these rules:



Additional Properties

Folding elements: `foldable` and `not-foldable-child` properties

<oXygen/> Author allows you to declare some elements to be *foldable* (collapsible). This is especially useful when working with large documents organized in logical blocks, editing a large DocBook article or book for instance.

To define the element whose content can be folded by the user, you must use the property: `foldable:true;`.

When collapsing an element, it is useful to keep some of its content visible, like a short description of the collapsed region. The property `not-foldable-child` is used to identify the child elements that are kept visible. It accepts as value an element name or a list of comma separated element names. If the element is marked as foldable (`foldable:true;`) but it doesn't have the property `not-foldable-child` or none of the specified non-foldable children exists then the element will still be foldable. In this case the element that will be kept visible when folded will be the before pseudo element.

Note

Both `foldable` and `not-foldable-child` are non standard properties and are recognized only by <oXygen/> Author.

Example 8. Folding DocBook Elements

All the elements below can have a `title` child element and are considered to be logical sections. We mark them as being foldable leaving the `title` element visible.

```
set,
book,
part,
reference,
chapter,
preface,
article,
sect1,
sect2,
sect3,
sect4,
section,
appendix,
figure,
example,
table {
    foldable:true;
    not-foldable-child: title;
}
```

Link elements

<oXygen/> Author allows you to declare some elements to be *links*. This is especially useful when working with many documents which refer each other. The links allow for an easy way to get from one document to another. Clicking on the link marker will open the referred resource in an editor.

To define the element which should be considered a link, you must use the property `link` on the before or after pseudo element. The value of the property indicates the location of the linked resource. Since links are usually indicated by the value of an attribute in most cases it will have the a value similar to `attr(href)`

Note

`link` is a non standard property and is recognized only by <oXygen/> Author.

Example 9. Docbook Link Elements

All the elements below are defined to be links on the before pseudo element and their value is defined by the value of an attribute.

```
*[href]:before{
    link:attr(href);
    content: "Click " attr(href) " for opening" ;
}

ulink[url]:before{
    link:attr(url);
    content: "Click to open: " attr(url);
}

olink[targetdoc]:before{
    link: attr(targetdoc);
    content: "Click to open: " attr(targetdoc);
}
```

Display Tag Markers

<oXygen/> Author allows you to choose whether tag markers of an element should never be presented or the current Display mode should be respected. This is especially useful when working with `:before` and `:after` pseudo elements in which case the element range is already visually defined so the tag markers are redundant.

The property is named `display-tags`. Its possible values are :

- *none* Tags markers must not be presented regardless of the current Display mode.
- *default* The tag markers will be created depending on the current Display mode.
- *inherit* The value of the property is inherited from an ancestor element.

```
display-tags
    Value: none | default | inherit
    Initial: default
    Applies to: all nodes(comments, elements, CDATA, etc)
    Inherited: false
    Media: all
```

Note

`display-tags` is a non standard property and is recognized only by <oXygen/> Author.

Example 10. Docbook Para elements

In this example the para element from Docbook is using an `:before` and `:after` element so we don't want its tag markers to be visible.

```
para:before{
    content: "{";
}

para:after{
    content: "}";
}

para{
    display-tags: none;
    display:block;
    margin: 0.5em 0;
}
```

<oXygen/> Custom CSS functions

In <oXygen/> Author there are implemented a few <oXygen/> specific custom CSS functions. Imbricated custom functions are also supported.

Example 11. Imbricated functions

The result of the functions below will be the local name of the current node with the first letter capitalized.

```
capitalize(local-name())
```

The `local-name()` function

This function evaluates the local name of the current node. It does not have any arguments

The `name()` function

This function evaluates the qualified name of the current node. It does not have any arguments

The `url()` function

This function evaluates the URL of a location relative to the CSS file location and appends each of the relative path components to the final location.

```
url(location, loc_1, loc_2);(...);

location ;
loc_1 ;
loc_2 ;
```

location	The location as string. If not absolute, will be solved relative to the CSS file URL.
loc_1 ... loc_n	Relative location path components as string. (optional)

The `base-uri()` function

This function evaluates the base URL in the context of the current node. It does not have any arguments and takes into account the `xml:base` context of the current node. See the XML Base specification [http://www.w3.org/TR/xmlbase/] for more details.

The `parent-url()` function

This function evaluates the parent URL of an URL received as string.

```
parent-url(url);

url ;
```

url The url as string.

The `capitalize()` function

This function capitalizes the first letter of the text received as argument.

```
capitalize(text);

text ;
```

text The text for which the first letter will be capitalized.

The `uppercase()` function

This function transforms to upper case the text received as argument.

```
uppercase(text);

text ;
```

text The text to be capitalized.

The `lowercase()` function

This function transforms to lower case the text received as argument.

```
lowercase(text);

text ;
```

text The text to be lower cased.

The `concat()` function

This function concatenates the received string arguments.

```
concat(str_1, str_2);(...);

str_1 ;
str_2 ;
```

str_1 ... str_n The string arguments to be concatenated.

The `replace()` function

This function has two signatures:

- `replace(`*`text`*`, `*`target`*`, `*`replacement`*`);`

  `text` ;
  `target` ;
  `replacement` ;

  This function replaces each substring of the text that matches the literal target string with the specified literal replacement string.

text	The text in which the replace will occur.
target	The target string to be replaced.
replacement	The string replacement.

- `replace(`*`text`*`, `*`target`*`, `*`replacement`*`, `*`isRegExp`*`);`

  `text` ;
  `target` ;
  `replacement` ;
  `isRegExp` ;

  This function replaces each substring of the text that matches the target string with the specified replacement string.

`text`	The text in which the replace will occur.
`target`	The target string to be replaced.
`replacement`	The string replacement.
`isRegExp`	If *true* the target and replacement arguments are considered regular expressions in PERL syntax, if *false* they are considered literal strings.

The `unparsed-entity-uri()` function

This function returns the uri value of an unparsed entity name.

`unparsed-entity-uri(`*`unparsedEntityName`*`);`

`unparsedEntityName` ;

`unparsedEntityName`	The name of an unparsed entity defined in the DTD.

This function can be useful to display images which are referred with unparsed entity names.

Example 12. CSS for displaying the image in Author for an *`imagedata`* with *`entityref`* to an unparsed entity

```
imagedata[entityref]{
content: url(unparsed-entity-uri(attr(entityref)));
}
```

The `attributes()` function

This function concatenates the attributes for an element and returns the serialization.

```
attributes();
```

Example 13. attributes()

For the following XML fragment:`<element att1="x" xmlns:a="2" x="&quot;"/>` the `attributes()` function will return `att1="x" xmlns:a="2" x="""`.

Example Files Listings

The Simple Documentation Framework Files

XML Schema files

`sdf.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oxygenxml.com/sample/documentation"
    xmlns:doc="http://www.oxygenxml.com/sample/documentation"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts"
    elementFormDefault="qualified">

    <xs:import
        namespace="http://www.oxygenxml.com/sample/documentation/abstracts"
        schemaLocation="abs.xsd"/>

    <xs:element name="book" type="doc:sectionType"/>
    <xs:element name="article" type="doc:sectionType"/>
    <xs:element name="section" type="doc:sectionType"/>

    <xs:complexType name="sectionType">
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element ref="abs:def" minOccurs="0"/>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="doc:section"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:choice maxOccurs="unbounded">
                    <xs:element ref="doc:para"/>
                    <xs:element ref="doc:ref"/>
                    <xs:element ref="doc:image"/>
                    <xs:element ref="doc:table"/>
                </xs:choice>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
```

```
<xs:element name="para" type="doc:paragraphType"/>

<xs:complexType name="paragraphType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b"/>
        <xs:element name="i"/>
        <xs:element name="link"/>
    </xs:choice>
</xs:complexType>

<xs:element name="ref">
    <xs:complexType>
        <xs:attribute name="location" type="xs:anyURI"
            use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="image">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI"
            use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="table">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="customcol" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="width" type="xs:string"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="header">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td"
                            maxOccurs="unbounded"
                            type="doc:paragraphType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="tr" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="td"
                            type="doc:tdType"
                            maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="width" type="xs:string"/>
    </xs:complexType>
```

```
    </xs:element>


    <xs:complexType name="tdType">
        <xs:complexContent>
            <xs:extension base="doc:paragraphType">
                <xs:attribute name="row_span"
                    type="xs:integer"/>
                <xs:attribute name="column_span"
                    type="xs:integer"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

abs.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=
 "http://www.oxygenxml.com/sample/documentation/abstracts">
    <xs:element name="def" type="xs:string"/>
</xs:schema>
```

CSS Files

sdf.css

```
/* Element from another namespace */
@namespace abs "http://www.oxygenxml.com/sample/documentation/abstracts";

abs|def{
    font-family:monospace;
    font-size:smaller;
}
abs|def:before{
    content:"Definition:";
    color:gray;
}

/* Vertical flow */
book,
section,
para,
title,
image,
ref {
    display:block;
}

/* Horizontal flow */
b,i {
    display:inline;
```

```
}

section{
    margin-left:1em;
    margin-top:1em;
}

section{
    foldable:true;
    not-foldable-child: title;
}

link[href]:before{
    display:inline;
    link:attr(href);
    content: "Click to open: " attr(href);
}

/* Title rendering*/
title{
    font-size: 2.4em;
    font-weight:bold;
}

* * title{
    font-size: 2.0em;
}
* * * title{
    font-size: 1.6em;
}
* * * * title{
    font-size: 1.2em;
}

book,
article{
    counter-reset:sect;
}
book > section,
article > section{
    counter-increment:sect;
}
book > section > title:before,
article > section > title:before{
    content: "Section: " counter(sect) " ";
}

/* Inlines rendering*/
b {
    font-weight:bold;
}

i {
    font-style:italic;
```

```
}

/*Table rendering */
table{
    display:table;
    border:1px solid navy;
    margin:1em;
    max-width:1000px;
    min-width:150px;
}

table[width]{
  width:attr(width, length);
}

tr, header{
    display:table-row;
}

header{
    background-color: silver;
    color:inherit
}

td{
  display:table-cell;
  border:1px solid navy;
  padding:1em;
}

image{
    display:block;
    content: attr(href, url);
    margin-left:2em;
}
```

XML Files

sdf_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://www.oxygenxml.com/sample/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:abs="http://www.oxygenxml.com/sample/documentation/abstracts">
    <title>My Technical Book</title>
    <section>
        <title>XML</title>
        <abs:def>Extensible Markup Language</abs:def>
        <para>In this section of the book I will explain
            different XML applications.</para>
    </section>
    <section>
        <title>Accessing XML data.</title>
```

```
<section>
    <title>XSLT</title>
    <abs:def>Extensible stylesheet language
        transformation (XSLT) is a language for
        transforming XML documents into other XML
        documents.</abs:def>
    <para>A list of XSL elements and what they do..</para>
    <table>
        <header>
            <td>XSLT Elements</td>
            <td>Description</td>
        </header>
        <tr>
            <td>
                <b>xsl:stylesheet</b>
            </td>
            <td>The <i>xsl:stylesheet</i> element is
                always the top-level element of an
                XSL stylesheet. The name
                    <i>xsl:transform</i> may be used
                as a synonym.</td>
        </tr>
        <tr>
            <td>
                <b>xsl:template</b>
            </td>
            <td>The <i>xsl:template</i> element has
                an optional mode attribute. If this
                is present, the template will only
                be matched when the same mode is
                used in the invoking
                    <i>xsl:apply-templates</i>
                element.</td>
        </tr>
        <tr>
            <td>
                <b>for-each</b>
            </td>
            <td>The xsl:for-each element causes
                iteration over the nodes selected by
                a node-set expression.</td>
        </tr>
        <tr>
            <td column_span="2">End of the list</td>
        </tr>
    </table>
</section>
<section>
    <title>XPath</title>
    <abs:def>XPath (XML Path Language) is a terse
        (non-XML) syntax for addressing portions of
        an XML document. </abs:def>
    <para>Some of the XPath functions.</para>
    <table>
```

```
                <header>
                    <td>Function</td>
                    <td>Description</td>
                </header>
                <tr>
                    <td>format-number</td>
                    <td>The <i>format-number</i> function
                        converts its first argument to a
                        string using the format pattern
                        string specified by the second
                        argument and the decimal-format
                        named by the third argument, or the
                        default decimal-format, if there is
                        no third argument</td>
                </tr>
                <tr>
                    <td>current</td>
                    <td>The <i>current</i> function returns
                        a node-set that has the current node
                        as its only member.</td>
                </tr>
                <tr>
                    <td>generate-id</td>
                    <td>The <i>generate-id</i> function
                        returns a string that uniquely
                        identifies the node in the argument
                        node-set that is first in document
                        order.</td>
                </tr>
            </table>
        </section>
    </section>
    <section>
        <title>Documentation frameworks</title>
        <para>One of the most important documentation
            frameworks is Docbook.</para>
        <image
            href="http://www.xmlhack.com/images/docbook.gif"/>
        <para>The other is the topic oriented DITA, promoted
            by OASIS.</para>
        <image
href="http://www.oasis-open.org/images/standards/oasis_standard.jpg"
        />
    </section>
</book>
```

XSL Files

sdf.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
```

```
xpath-default-namespace=
"http://www.oxygenxml.com/sample/documentation">

<xsl:template match="/">
    <html><xsl:apply-templates/></html>
</xsl:template>

<xsl:template match="section">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="image">
    <img src="{@href}"/>
</xsl:template>

<xsl:template match="para">
    <p>
        <xsl:apply-templates/>
    </p>
</xsl:template>

<xsl:template match="abs:def"
    xmlns:abs=
    "http://www.oxygenxml.com/sample/documentation/abstracts">
    <p>
        <u><xsl:apply-templates/></u>
    </p>
</xsl:template>

<xsl:template match="title">
    <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="b">
    <b><xsl:apply-templates/></b>
</xsl:template>

<xsl:template match="i">
    <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="table">
    <table frame="box" border="1px">
        <xsl:apply-templates/>
    </table>
</xsl:template>

<xsl:template match="header">
    <tr>
        <xsl:apply-templates/>
    </tr>
</xsl:template>

<xsl:template match="tr">
```

```
        <tr>
            <xsl:apply-templates/>
        </tr>
    </xsl:template>

    <xsl:template match="td">
        <td>
            <xsl:apply-templates/>
        </td>
    </xsl:template>

    <xsl:template match="header/header/td">
        <th>
            <xsl:apply-templates/>
        </th>
    </xsl:template>

</xsl:stylesheet>
```

Java Files

InsertImageOperation.java

```
package simple.documentation.framework;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.net.MalformedURLException;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.filechooser.FileFilter;

import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class InsertImageOperation implements AuthorOperation {

       //
```

```
// Implementing the Author Operation Interface.
//

/**
 * Performs the operation.
 */
public void doOperation(AuthorAccess authorAccess,
  ArgumentsMap arguments)
  throws IllegalArgumentException,
      AuthorOperationException {

 JFrame oxygenFrame = (JFrame) authorAccess.getParentFrame();
 String href = displayURLDialog(oxygenFrame);
 if (href.length() != 0) {
     // Creates the image XML fragment.
     String imageFragment =
         "<image xmlns='http://www.oxygenxml.com/sample/documentation'" +
                " href='" + href + "'/>";

  // Inserts this fragment at the caret position.
  int caretPosition = authorAccess.getCaretOffset();
  authorAccess.insertXMLFragment(imageFragment, caretPosition);
 }
}

/**
 * Has no arguments.
 *
 * @return null.
 */
public ArgumentDescriptor[] getArguments() {
 return null;
}

/**
 * @return A description of the operation.
 */
public String getDescription() {
 return "Inserts an image element. Asks the" +
" user for a URL reference.";
}

//
// End of interface implementation.
//

//
// Auxiliary methods.
//

/**
 * Displays the URL dialog.
 *
 * @param parentFrame The parent frame for
```

```
* the dialog.
 * @return The selected URL string value,
* or the empty string if the user canceled
* the URL selection.
 */
private String displayURLDialog(JFrame parentFrame) {

 final JDialog dlg = new JDialog(parentFrame,
"Enter the value for the href attribute", true);
 JPanel mainContent = new JPanel(new GridBagLayout());

 // The text field.
 GridBagConstraints cstr = new GridBagConstraints();
 cstr.gridx = 0;
 cstr.gridy = 0;
 cstr.weightx = 0;
 cstr.gridwidth = 1;
 cstr.fill = GridBagConstraints.HORIZONTAL;
 mainContent.add(new JLabel("Image URI:"), cstr);

 cstr.gridx = 1;
 cstr.weightx = 1;
 final JTextField urlField = new JTextField();
 urlField.setColumns(15);
 mainContent.add(urlField, cstr);

 // Add the "Browse button."
 cstr.gridx = 2;
 cstr.weightx = 0;
 JButton browseButton = new JButton("Browse");
 browseButton.addActionListener(new ActionListener() {

  /**
   * Shows a file chooser dialog.
   */
  public void actionPerformed(ActionEvent e) {
   JFileChooser fileChooser = new JFileChooser();

   fileChooser.setMultiSelectionEnabled(false);
   // Accepts only the image files.
   fileChooser.setFileFilter(new FileFilter() {
    public String getDescription() {
     return "Image files";
    }

    public boolean accept(File f) {
     String fileName = f.getName();
     return f.isFile() &&
       ( fileName.endsWith(".jpeg")
       || fileName.endsWith(".jpg")
       || fileName.endsWith(".gif")
       || fileName.endsWith(".png")
       || fileName.endsWith(".svg"));
    }
```

```
   });
   if (fileChooser.showOpenDialog(dlg)
        == JFileChooser.APPROVE_OPTION) {
    File file = fileChooser.getSelectedFile();
    try {
     // Set the file into the text field.
     urlField.setText(file.toURL().toString());
    } catch (MalformedURLException ex) {
     // This should not happen.
     ex.printStackTrace();
    }
   }
  }
 });
 mainContent.add(browseButton, cstr);

 // Add the "Ok" button to the layout.
 cstr.gridx = 0;
 cstr.gridy = 1;
 cstr.weightx = 0;
 JButton okButton = new JButton("Ok");
 okButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
   dlg.setVisible(false);
  }
 });
 mainContent.add(okButton, cstr);
 mainContent.setBorder(
 BorderFactory.createEmptyBorder(10, 5, 10, 5));

 // Add the "Cancel" button to the layout.
 cstr.gridx = 2;
 JButton cancelButton = new JButton("Cancel");
 cancelButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
   urlField.setText("");
   dlg.setVisible(false);
  }
 });
 mainContent.add(cancelButton, cstr);

 // When the user closes the dialog
// from the window decoration,
 // assume "Cancel" action.
 dlg.addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent e) {
   urlField.setText("");
  }
 });

 dlg.getContentPane().add(mainContent);
 dlg.pack();
 dlg.setLocationRelativeTo(parentFrame);
 dlg.setVisible(true);
```

```
 return urlField.getText();
}

/**
 * Test method.
 *
 * @param args The arguments are ignored.
 */
public static void main(String[] args) {
 InsertImageOperation operation =
      new InsertImageOperation();
 System.out.println("Choosen URL: " +
   operation.displayURLDialog(new JFrame()));
}
}
```

QueryDatabaseOperation.java

```
package simple.documentation.framework;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;

import ro.sync.ecss.extensions.api.ArgumentDescriptor;
import ro.sync.ecss.extensions.api.ArgumentsMap;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperation;
import ro.sync.ecss.extensions.api.AuthorOperationException;

public class QueryDatabaseOperation implements AuthorOperation{

  private static String ARG_JDBC_DRIVER ="jdbc_driver";
  private static String ARG_USER ="user";
  private static String ARG_PASSWORD ="password";
  private static String ARG_SQL ="sql";
  private static String ARG_CONNECTION ="connection";

  /**
   * @return The array of arguments the developer must specify when
   * configuring the action.
   */
  public ArgumentDescriptor[] getArguments() {
    ArgumentDescriptor args[] = new ArgumentDescriptor[] {
        new ArgumentDescriptor(
            ARG_JDBC_DRIVER,
            ArgumentDescriptor.TYPE_STRING,
            "The name of the Java class that is the JDBC driver."),
        new ArgumentDescriptor(
            ARG_CONNECTION,
            ArgumentDescriptor.TYPE_STRING,
```

```
            "The database URL connection string."),
        new ArgumentDescriptor(
            ARG_USER,
            ArgumentDescriptor.TYPE_STRING,
            "The name of the database user."),
        new ArgumentDescriptor(
            ARG_PASSWORD,
            ArgumentDescriptor.TYPE_STRING,
            "The database password."),
        new ArgumentDescriptor(
            ARG_SQL,
            ArgumentDescriptor.TYPE_STRING,
            "The SQL statement to be executed.")
    };
    return args;
  }

  /**
   * @return The operation description.
   */
  public String getDescription() {
    return "Executes a database query and puts the result in a table.";
  }

  public void doOperation(AuthorAccess authorAccess, ArgumentsMap map)
      throws IllegalArgumentException, AuthorOperationException {

    // Collects the arguments.
    String jdbcDriver =
      (String)map.getArgumentValue(ARG_JDBC_DRIVER);
    String connection =
      (String)map.getArgumentValue(ARG_CONNECTION);
    String user =
      (String)map.getArgumentValue(ARG_USER);
    String password =
      (String)map.getArgumentValue(ARG_PASSWORD);
    String sql =
      (String)map.getArgumentValue(ARG_SQL);

    int caretPosition = authorAccess.getCaretOffset();
    try {
      authorAccess.insertXMLFragment(
          getFragment(jdbcDriver, connection, user, password, sql),
          caretPosition);
    } catch (SQLException e) {
      throw new AuthorOperationException(
          "The operation failed due to the following database error: " +
     e.getMessage(), e);
    } catch (ClassNotFoundException e) {
      throw new AuthorOperationException(
          "The JDBC database driver was not found. Tried to load ' " +
     jdbcDriver + "'", e);
    }
  }
```

```
/**
 * Creates a connection to the database, executes
 * the SQL statement and creates an XML fragment
 * containing a table element that wraps the data
 * from the result set.
 *
 *
 * @param jdbcDriver The class name of the JDBC driver.
 * @param connectionURL The connection URL.
 * @param user The database user.
 * @param password The password.
 * @param sql The SQL statement.
 * @return The string containing the XML fragment.
 *
 * @throws SQLException thrown when there is a
 * problem accessing the database or there are
 * erors in the SQL expression.
 * @throws ClassNotFoundException when the JDBC
 * driver class could not be loaded.
 */
private String getFragment(
      String jdbcDriver,
      String connectionURL,
      String user,
      String password,
      String sql) throws
        SQLException,
        ClassNotFoundException {

    Properties pr = new Properties();
    pr.put("characterEncoding", "UTF8");
    pr.put("useUnicode", "TRUE");
    pr.put("user", user);
    pr.put("password", password);

    // Loads the database driver.
    Class.forName(jdbcDriver);

    // Opens the connection
    Connection connection =
      DriverManager.getConnection(connectionURL, pr);
    java.sql.Statement statement =
      connection.createStatement();
    ResultSet resultSet =
      statement.executeQuery(sql);

    StringBuffer fragmentBuffer = new StringBuffer();
    fragmentBuffer.append(
        "<table xmlns='http://www.oxygenxml.com/sample/documentation'>");

    //
    // Creates the table header.
    //
```

```
      fragmentBuffer.append("<header>");
      ResultSetMetaData metaData = resultSet.getMetaData();
      int columnCount = metaData.getColumnCount();
      for (int i = 1; i <= columnCount; i++) {
          fragmentBuffer.append("<td>");
          fragmentBuffer.append(
              xmlEscape(metaData.getColumnName(i)));
          fragmentBuffer.append("</td>");
        }
      fragmentBuffer.append("</header>");

      //
      // Creates the table content.
      //
      while (resultSet.next()) {
          fragmentBuffer.append("<tr>");
          for (int i = 1; i <= columnCount; i++) {
              fragmentBuffer.append("<td>");
              fragmentBuffer.append(
                  xmlEscape(resultSet.getObject(i)));
              fragmentBuffer.append("</td>");
          }
          fragmentBuffer.append("</tr>");
        }

      fragmentBuffer.append("</table>");

      // Cleanup
      resultSet.close();
      statement.close();
      connection.close();
      return fragmentBuffer.toString();
  }

  /**
   * Some of the values from the database table
   * may contain characters that must be escaped
   * in XML, to ensure the fragment is well formed.
   *
   * @param object The object from the database.
   * @return The escaped string representation.
   */
  private String xmlEscape(Object object) {
    String str = String.valueOf(object);
    return str.
      replaceAll("&", "&amp;").
      replaceAll("<", "&lt;");
  }
}
```

SDFExtensionsBundle.java

```
package simple.documentation.framework;
```

```
import ro.sync.contentcompletion.xml.SchemaManagerFilter;
import ro.sync.ecss.extensions.api.AttributesValueEditor;
import ro.sync.ecss.extensions.api.AuthorExtensionStateListener;
import ro.sync.ecss.extensions.api.AuthorReferenceResolver;
import ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler;
import ro.sync.ecss.extensions.api.AuthorTableCellSpanProvider;
import ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider;
import ro.sync.ecss.extensions.api.ExtensionsBundle;
import ro.sync.ecss.extensions.api.StylesFilter;
import ro.sync.ecss.extensions.api.link.ElementLocatorProvider;
import ro.sync.ecss.extensions.commons.DefaultElementLocatorProvider;
import simple.documentation.framework.extensions.SDFAttributesValueEditor;
import simple.documentation.framework.extensions.SDFAuthorExtensionStateListener;
import simple.documentation.framework.extensions.SDFReferencesResolver;
import simple.documentation.framework.extensions.SDFSchemaAwareEditingHandler;
import simple.documentation.framework.extensions.SDFSchemaManagerFilter;
import simple.documentation.framework.extensions.SDFStylesFilter;
import simple.documentation.framework.extensions.TableCellSpanProvider;
import simple.documentation.framework.extensions.TableColumnWidthProvider;

/**
 * Simple Document Framework extension bundle.
 *
 */
public class SDFExtensionsBundle extends ExtensionsBundle {
  /**
   * Editor for attributes values.
   */
  public AttributesValueEditor createAttributesValueEditor(boolean arg0) {
    return new SDFAttributesValueEditor();
  }

  /**
   * Simple documentation framework state listener.
   */
  public AuthorExtensionStateListener createAuthorExtensionStateListener() {
    return new SDFAuthorExtensionStateListener();
  }

  /**
   * Filter for content completion proposals from the schema manager.
   */
  public SchemaManagerFilter createSchemaManagerFilter() {
    return new SDFSchemaManagerFilter();
  }

  /**
   * Default element locator.
   */
  public ElementLocatorProvider createElementLocatorProvider() {
    return new DefaultElementLocatorProvider();
  }

  /**
```

```
   * Expand content references.
   */
  public AuthorReferenceResolver createAuthorReferenceResolver() {
    return new SDFReferencesResolver();
  }

  /**
   * CSS styles filtering.
   */
  public StylesFilter createAuthorStylesFilter() {
    return new SDFStylesFilter();
  }

  /**
   * Provider for table cell span informations.
   */
  public AuthorTableCellSpanProvider createAuthorTableCellSpanProvider() {
    return new TableCellSpanProvider();
  }

  /**
   * Table column width provider responsible of handling modifications regarding
   * table width and column widths.
   */
  public AuthorTableColumnWidthProvider createAuthorTableColumnWidthProvider() {
    return new TableColumnWidthProvider();
  }

  /**
   * Editing support for SDF documents responsible of handling typing and paste events ins
   */
  public AuthorSchemaAwareEditingHandler getAuthorSchemaAwareEditingHandler() {
    return new SDFSchemaAwareEditingHandler();
  }

  /**
   * The unique identifier of the Document Type.
   * This identifier will be used to store custom SDF options.
   */
  public String getDocumentTypeID() {
    return "Simple.Document.Framework.document.type";
  }

  /**
   * Bundle description.
   */
  public String getDescription() {
    return "A custom extensions bundle used for the Simple Document Framework";
  }
}
```

SDFSchemaManagerFilter.java

```
package simple.documentation.framework;

import java.util.Iterator;
import java.util.List;

import ro.sync.contentcompletion.xml.CIAttribute;
import ro.sync.contentcompletion.xml.CIElement;
import ro.sync.contentcompletion.xml.CIValue;
import ro.sync.contentcompletion.xml.Context;
import ro.sync.contentcompletion.xml.ContextElement;
import ro.sync.contentcompletion.xml.SchemaManagerFilter;
import ro.sync.contentcompletion.xml.WhatAttributesCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatElementsCanGoHereContext;
import ro.sync.contentcompletion.xml.WhatPossibleValuesHasAttributeContext;

public class SDFSchemaManagerFilter implements SchemaManagerFilter {

 @Override
 public List<CIValue> filterAttributeValues(List<CIValue> attributeValues,
       WhatPossibleValuesHasAttributeContext context) {
      return attributeValues;
 }

 @Override
 public List<CIAttribute> filterAttributes(List<CIAttribute> attributes,
       WhatAttributesCanGoHereContext context) {
   // If the element from the current context is the 'table' element add the
   // attribute named 'frame' to the list of default content
      // completion proposals
   ContextElement contextElement = context.getParentElement();
   if ("table".equals(contextElement.getQName())) {
     CIAttribute frameAttribute = new CIAttribute();
     frameAttribute.setName("frame");
     frameAttribute.setRequired(false);
     frameAttribute.setFixed(false);
     frameAttribute.setDefaultValue("void");
     attributes.add(frameAttribute);
   }
   return attributes;
 }

 @Override
 public List<CIValue> filterElementValues(List<CIValue> elementValues,
       Context context) {
      return elementValues;
 }

 @Override
 public List<CIElement> filterElements(List<CIElement> elements,
       WhatElementsCanGoHereContext context) {
```

```
   // If the element from the current context is the 'header'
      // element remove the 'td' element from the list of content
      // completion proposals and add the 'th' element.
   ContextElement contextElement = context.getElementStack().peek();
   if ("header".equals(contextElement.getQName())) {
     for (Iterator<CIElement> iterator = elements.iterator();
                   iterator.hasNext();) {
       CIElement element = iterator.next();
        // Remove the 'td' element
       if ("td".equals(element.getQName())) {
          elements.remove(element);
          break;
        }
      }
     // Insert the 'th' element in the list of content completion proposals
     CIElement thElement = new SDFElement();
     thElement.setName("th");
     elements.add(thElement);
   }
    return elements;
 }

 @Override
 public String getDescription() {
  return null;
 }
}
```

SDFSchemaAwareEditingHandler.java

```
package simple.documentation.framework.extensions;

import java.util.List;

import javax.swing.text.BadLocationException;

import ro.sync.contentcompletion.xml.ContextElement;
import ro.sync.contentcompletion.xml.WhatElementsCanGoHereContext;
import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorOperationException;
import ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler;
import ro.sync.ecss.extensions.api.AuthorSchemaManager;
import ro.sync.ecss.extensions.api.InvalidEditException;
import ro.sync.ecss.extensions.api.node.AuthorDocumentFragment;
import ro.sync.ecss.extensions.api.node.AuthorElement;
import ro.sync.ecss.extensions.api.node.AuthorNode;

/**
 * Specific editing support for SDF documents. Handles typing and paste events inside sect
 */
public class SDFSchemaAwareEditingHandler implements AuthorSchemaAwareEditingHandler {

  private static final String SDF_NAMESPACE = "http://www.oxygenxml.com/sample/documentati
  /**
```

```
 * SDF table element name.
 */
private static final String SDF_TABLE = "table";
/**
 * SDF table row name.
 */
private static final String SDF_TABLE_ROW = "tr";
/**
 * SDF table cell name.
 */
private static final String SDF_TABLE_CELL = "td";
/**
 * SDF section element name.
 */
private static final String SECTION = "section";
/**
 * SDF para element name.
 */
protected static final String PARA = "para";
/**
 * SDF title element name.
 */
protected static final String TITLE = "title";

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleDelete(int, in
 */
public boolean handleDelete(int offset, int deleteType, AuthorAccess authorAccess, boole
throws InvalidEditException {
  // Not handled.
  return false;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleDeleteElementT
 */
public boolean handleDeleteElementTags(AuthorNode nodeToUnwrap, AuthorAccess authorAcces
throws InvalidEditException {
  // Not handled.
  return false;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleDeleteSelectio
 */
public boolean handleDeleteSelection(int selectionStart, int selectionEnd, int generated
    AuthorAccess authorAccess) throws InvalidEditException {
  // Not handled.
  return false;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleJoinElements(r
 */
```

```
public boolean handleJoinElements(AuthorNode targetNode, List<AuthorNode> nodesToJoin, A
throws InvalidEditException {
  // Not handled.
  return false;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handlePasteFragment(
 */
public boolean handlePasteFragment(int offset, AuthorDocumentFragment[] fragmentsToInser
    AuthorAccess authorAccess) throws InvalidEditException {
  boolean handleInsertionEvent = false;
  AuthorSchemaManager authorSchemaManager = authorAccess.getDocumentController().getAuth
  if (!authorSchemaManager.isLearnSchema() &&
      !authorSchemaManager.hasLoadingErrors() &&
      authorSchemaManager.getAuthorSchemaAwareOptions().isEnableSmartPaste()) {
    handleInsertionEvent = handleInsertionEvent(offset, fragmentsToInsert, authorAccess)
  }
  return handleInsertionEvent;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorSchemaAwareEditingHandler#handleTyping(int, ch
 */
public boolean handleTyping(int offset, char ch, AuthorAccess authorAccess)
throws InvalidEditException {
  boolean handleTyping = false;
  AuthorSchemaManager authorSchemaManager = authorAccess.getDocumentController().getAuth
  if (!authorSchemaManager.isLearnSchema() &&
      !authorSchemaManager.hasLoadingErrors() &&
      authorSchemaManager.getAuthorSchemaAwareOptions().isEnableSmartTyping()) {
    try {
      AuthorDocumentFragment characterFragment =
        authorAccess.getDocumentController().createNewDocumentTextFragment(String.valueO
      handleTyping = handleInsertionEvent(offset, new AuthorDocumentFragment[] {characte
    } catch (AuthorOperationException e) {
      throw new InvalidEditException(e.getMessage(), "Invalid typing event: " + e.getMes
    }
  }
  return handleTyping;
}

/**
 * Handle an insertion event (either typing or paste).
 *
 * @param offset Offset where the insertion event occurred.
 * @param fragmentsToInsert Fragments that must be inserted at the given offset.
 * @param authorAccess Author access.
 * @return <code>true</code> if the event was handled, <code>false</code> otherwise.
 *
 * @throws InvalidEditException The event was rejected because it is invalid.
 */
private boolean handleInsertionEvent(
    int offset,
```

```
    AuthorDocumentFragment[] fragmentsToInsert,
    AuthorAccess authorAccess) throws InvalidEditException {
  AuthorSchemaManager authorSchemaManager = authorAccess.getDocumentController().getAuth
  boolean handleEvent = false;
  try {
    AuthorNode nodeAtInsertionOffset = authorAccess.getDocumentController().getNodeAtOff
    if (isElementWithNameAndNamespace(nodeAtInsertionOffset, SDF_TABLE)) {
      // Check if the fragment is allowed as it is.
      boolean canInsertFragments = authorSchemaManager.canInsertDocumentFragments(
          fragmentsToInsert,
          offset,
          AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS);
      if (!canInsertFragments) {
        handleEvent = handleInvalidInsertionEventInTable(
            offset,
            fragmentsToInsert,
            authorAccess,
            authorSchemaManager);
      }
    } else if(isElementWithNameAndNamespace(nodeAtInsertionOffset, SECTION)) {
      // Check if the fragment is allowed as it is.
      boolean canInsertFragments = authorSchemaManager.canInsertDocumentFragments(
          fragmentsToInsert,
          offset,
          AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS);
      if (!canInsertFragments) {
        // Insertion in 'section' element
        handleEvent = handleInvalidInsertionEventInSect(
            offset,
            fragmentsToInsert,
            authorAccess,
            authorSchemaManager);
      }
    }
  } catch (BadLocationException e) {
    throw new InvalidEditException(e.getMessage(), "Invalid typing event: " + e.getMessa
  } catch (AuthorOperationException e) {
    throw new InvalidEditException(e.getMessage(), "Invalid typing event: " + e.getMessa
  }
  return handleEvent;
}

/**
 * @return <code>true</code> if the given node is an element with the given local name a
 */
protected boolean isElementWithNameAndNamespace(AuthorNode node, String elementLocalName
  boolean result = false;
  if(node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    result = elementLocalName.equals(element.getLocalName()) && element.getNamespace().e
  }
  return result;
}
```

```
/**
 * Try to handle invalid insertion events in a SDF 'table'.
 * A row element will be inserted with a new cell in which the fragments will be inserte
 *
 * @param offset Offset where the insertion event occurred.
 * @param fragmentsToInsert Fragments that must be inserted at the given offset.
 * @param authorAccess Author access.
 * @return <code>true</code> if the event was handled, <code>false</code> otherwise.
 */
private boolean handleInvalidInsertionEventInTable(
    int offset,
    AuthorDocumentFragment[] fragmentsToInsert,
    AuthorAccess authorAccess,
    AuthorSchemaManager authorSchemaManager) throws BadLocationException, AuthorOperatio
  boolean handleEvent = false;
  // Typing/paste inside a SDF table. We will try to wrap the fragment into a new cell a
  WhatElementsCanGoHereContext context = authorSchemaManager.createWhatElementsCanGoHere
  StringBuilder xmlFragment = new StringBuilder("<");
  xmlFragment.append(SDF_TABLE_ROW);
  if (SDF_NAMESPACE != null && SDF_NAMESPACE.length() != 0) {
    xmlFragment.append(" xmlns=\"").append(SDF_NAMESPACE).append("\"");
  }
  xmlFragment.append("/>");

  // Check if a row can be inserted at the current offset.
  boolean canInsertRow = authorSchemaManager.canInsertDocumentFragments(
      new AuthorDocumentFragment[] {authorAccess.getDocumentController().createNewDocume
      context,
      AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS);

  // Derive the context by adding a new row element with a cell.
  if (canInsertRow) {
    pushContextElement(context, SDF_TABLE_ROW);
    pushContextElement(context, SDF_TABLE_CELL);

    // Test if fragments can be inserted in the new context.
    if (authorSchemaManager.canInsertDocumentFragments(
        fragmentsToInsert,
        context,
        AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS)) {

      // Insert a new row with a cell.
      xmlFragment = new StringBuilder("<");
      xmlFragment.append(SDF_TABLE_ROW);

      if (SDF_NAMESPACE != null && SDF_NAMESPACE.length() != 0) {
        xmlFragment.append(" xmlns=\"").append(SDF_NAMESPACE).append("\"");
      }
      xmlFragment.append("><");
      xmlFragment.append(SDF_TABLE_CELL);
      xmlFragment.append("/></");
      xmlFragment.append(SDF_TABLE_ROW);
      xmlFragment.append(">");
      authorAccess.getDocumentController().insertXMLFragment(xmlFragment.toString(), off
```

```
      // Get the newly inserted cell.
      AuthorNode newCell = authorAccess.getDocumentController().getNodeAtOffset(offset +
      for (int i = 0; i < fragmentsToInsert.length; i++) {
        authorAccess.getDocumentController().insertFragment(newCell.getEndOffset(), frag
      }

      handleEvent = true;
    }
  }
  return handleEvent;
}

/**
 * Derive the given context by adding the specified element.
 */
protected void pushContextElement(WhatElementsCanGoHereContext context, String elementNa
  ContextElement contextElement = new ContextElement();
  contextElement.setQName(elementName);
  contextElement.setNamespace(SDF_NAMESPACE);
  context.pushContextElement(contextElement, null);
}

/**
 * Try to handle invalid insertion events in 'section'.
 * The solution is to insert the <code>fragmentsToInsert</code> into a 'title' element i
 * into a 'para' element if the sect already contains a 'title'.
 *
 * @param offset Offset where the insertion event occurred.
 * @param fragmentsToInsert Fragments that must be inserted at the given offset.
 * @param authorAccess Author access.
 * @return <code>true</code> if the event was handled, <code>false</code> otherwise.
 */
private boolean handleInvalidInsertionEventInSect(int offset, AuthorDocumentFragment[] f
    AuthorSchemaManager authorSchemaManager) throws BadLocationException, AuthorOperatio
  boolean handleEvent = false;
  // Typing/paste inside an section.
  AuthorElement sectionElement = (AuthorElement) authorAccess.getDocumentController().ge

  if (sectionElement.getStartOffset() + 1 == sectionElement.getEndOffset()) {
    // Empty section element
    WhatElementsCanGoHereContext context = authorSchemaManager.createWhatElementsCanGoHe
    // Derive the context by adding a title.
    pushContextElement(context, TITLE);

    // Test if fragments can be inserted in 'title' element
    if (authorSchemaManager.canInsertDocumentFragments(
        fragmentsToInsert,
        context,
        AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS)) {
      // Create a title structure and insert fragments inside
      StringBuilder xmlFragment = new StringBuilder("<").append(TITLE);
      if (SDF_NAMESPACE != null && SDF_NAMESPACE.length() != 0) {
        xmlFragment.append(" xmlns=\"").append(SDF_NAMESPACE).append("\"");
```

```
        }
        xmlFragment.append(">").append("</").append(TITLE).append(">");
        // Insert title
        authorAccess.getDocumentController().insertXMLFragment(xmlFragment.toString(), off

        // Insert fragments
        AuthorNode newParaNode = authorAccess.getDocumentController().getNodeAtOffset(offs
        for (int i = 0; i < fragmentsToInsert.length; i++) {
          authorAccess.getDocumentController().insertFragment(newParaNode.getEndOffset(),
        }
        handleEvent = true;
      }
    } else {
      // Check if there is just a title.
      List<AuthorNode> contentNodes = sectionElement.getContentNodes();
      if (contentNodes.size() == 1) {
        AuthorNode child = contentNodes.get(0);
        boolean isTitleChild = isElementWithNameAndNamespace(child, TITLE);
        if (isTitleChild && child.getEndOffset() < offset) {
          // We are after the title.

          // Empty sect element
          WhatElementsCanGoHereContext context = authorSchemaManager.createWhatElementsCan
          // Derive the context by adding a para
          pushContextElement(context, PARA);

          // Test if fragments can be inserted in 'para' element
          if (authorSchemaManager.canInsertDocumentFragments(
              fragmentsToInsert,
              context,
              AuthorSchemaManager.VALIDATION_MODE_STRICT_FIRST_CHILD_LAX_OTHERS)) {
            // Create a para structure and insert fragments inside
            StringBuilder xmlFragment = new StringBuilder("<").append(PARA);
            if (SDF_NAMESPACE != null && SDF_NAMESPACE.length() != 0) {
              xmlFragment.append(" xmlns=\"").append(SDF_NAMESPACE).append("\"");
            }
            xmlFragment.append(">").append("</").append(PARA).append(">");
            // Insert para
            authorAccess.getDocumentController().insertXMLFragment(xmlFragment.toString(),

            // Insert fragments
            AuthorNode newParaNode = authorAccess.getDocumentController().getNodeAtOffset(
            for (int i = 0; i < fragmentsToInsert.length; i++) {
              authorAccess.getDocumentController().insertFragment(newParaNode.getEndOffset
            }
            handleEvent = true;
          }
        }
      }
    }
    return handleEvent;
  }
}
```

TableCellSpanProvider.java

```
package simple.documentation.framework;

public class TableCellSpanProvider
    implements AuthorTableCellSpanProvider {


  /**
   * Extracts the integer specifing what is the width
   * (in columns)  of the cell
   * representing in the table layout the cell element.
   */
  public Integer getColSpan(AuthorElement cell) {
    Integer colSpan = null;

    AttrValue attrValue = cell.getAttribute("column_span");
    if(attrValue != null) {
      // The attribute was found.
      String cs = attrValue.getValue();
      if(cs != null) {
        try {
          colSpan = new Integer(cs);
        } catch (NumberFormatException ex) {
          // The attribute value was not a number.
        }
      }
    }
    return colSpan;
  }

  /**
   * Extracts the integer specifing what is the
   * height (in rows) of the cell
   * representing in the table layout the cell element.
   */
  public Integer getRowSpan(AuthorElement cell) {
    Integer rowSpan = null;

    AttrValue attrValue = cell.getAttribute("row_span");
    if(attrValue != null) {
      // The attribute was found.
      String rs = attrValue.getValue();
      if(rs != null) {
        try {
          rowSpan = new Integer(rs);
        } catch (NumberFormatException ex) {
          // The attribute value was not a number.
        }
      }
    }
    return rowSpan;
  }
```

```
  /**
   * @return true considering the column specifications always available.
   */
  public boolean hasColumnSpecifications(AuthorElement tableElement) {
    return true;
  }

  /**
   * Ignored. We do not extract data from the
   * <code>table</code> element.
   */
  public void init(AuthorElement table) {
  }

  public String getDescription() {
    return
      "Implementation for the Simple Documentation Framework table layout.";
  }
}
```

TableColumnWidthProvider.java

```
package simple.documentation.framework.extensions;
import java.util.ArrayList;
import java.util.List;

import ro.sync.ecss.extensions.api.AuthorDocumentController;
import ro.sync.ecss.extensions.api.AuthorOperationException;
import ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider;
import ro.sync.ecss.extensions.api.WidthRepresentation;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;

/**
 *
 * Simple Documentation Framework table column width provider.
 *
 */
public class TableColumnWidthProvider implements AuthorTableColumnWidthProvider {

/**
 * Cols start offset
 */
private int colsStartOffset;

/**
 * Cols end offset
 */
private int colsEndOffset;

/**
 * Column widths specifications
 */
```

```
private List<WidthRepresentation> colWidthSpecs = new ArrayList<WidthRepresentation>();

/**
 * The table element
 */
private AuthorElement tableElement;

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#commitColumnWidthModifi
 *  ro.sync.ecss.extensions.api.AuthorDocumentController, ro.sync.ecss.extensions.api.Widt
 *  java.lang.String)
 */
public void commitColumnWidthModifications(AuthorDocumentController authorDocumentControll
        WidthRepresentation[] colWidths, String tableCellsTagName) throws AuthorOperationE
    if ("td".equals(tableCellsTagName)) {
        if (colWidths != null && tableElement != null) {
            if (colsStartOffset >= 0 && colsEndOffset >= 0 &&
                    colsStartOffset < colsEndOffset) {
                authorDocumentController.delete(colsStartOffset, colsEndOffset);
            }
            String xmlFragment = createXMLFragment(colWidths);
            int offset = -1;
            AuthorElement[] header = tableElement.getElementsByLocalName("header");
            if (header != null && header.length > 0) {
                // Insert the cols elements before the 'header' element
                offset = header[0].getStartOffset();
            }
            if (offset == -1) {
              throw new AuthorOperationException("No valid offset to insert the columns wi
            }
            authorDocumentController.insertXMLFragment(xmlFragment, offset);
        }
    }
}

/**
 * Creates the XML fragment representing the column specifications.
 *
 * @param widthRepresentations
 * @return The XML fragment as a string.
 */
private String createXMLFragment(WidthRepresentation[] widthRepresentations) {
 StringBuffer fragment = new StringBuffer();
 String ns = tableElement.getNamespace();
 for (int i = 0; i < widthRepresentations.length; i++) {
  WidthRepresentation width = widthRepresentations[i];
  fragment.append("<customcol");
  String strRepresentation = width.getWidthRepresentation();
  if (strRepresentation != null) {
   fragment.append(" width=\"" + width.getWidthRepresentation() + "\"");
  }
  if (ns != null && ns.length() > 0) {
   fragment.append(" xmlns=\"" + ns + "\"");
  }
```

```
  fragment.append("/>");
 }
 return fragment.toString();
}

/**
  * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#commitTableWidthModifi
  *    ro.sync.ecss.extensions.api.AuthorDocumentController, int, java.lang.String)
 */
public void commitTableWidthModification(AuthorDocumentController authorDocumentController
        int newTableWidth, String tableCellsTagName) throws AuthorOperationException {
    if ("td".equals(tableCellsTagName)) {
        if (newTableWidth > 0) {
            if (tableElement != null) {
                String newWidth = String.valueOf(newTableWidth);
                authorDocumentController.setAttribute(
                    "width",
                    new AttrValue(newWidth),
                    tableElement);
            } else {
                throw new AuthorOperationException("Cannot find the element representing t
            }
        }
    }
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#getCellWidth(
 *     ro.sync.ecss.extensions.api.node.AuthorElement, int, int)
 */
public List<WidthRepresentation> getCellWidth(AuthorElement cellElement, int colNumberStar
  int colSpan) {
 List<WidthRepresentation> toReturn = null;
 int size = colWidthSpecs.size();
 if (size >= colNumberStart && size >= colNumberStart + colSpan) {
  toReturn = new ArrayList<WidthRepresentation>(colSpan);
  for (int i = colNumberStart; i < colNumberStart + colSpan; i ++) {
   // Add the column widths
   toReturn.add(colWidthSpecs.get(i));
  }
 }
 return toReturn;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#getTableWidth(java.lang
 */
public WidthRepresentation getTableWidth(String tableCellsTagName) {
    WidthRepresentation toReturn = null;
    if (tableElement != null && "td".equals(tableCellsTagName)) {
        AttrValue widthAttr = tableElement.getAttribute("width");
        if (widthAttr != null) {
            String width = widthAttr.getValue();
            if (width != null) {
```

```
                toReturn = new WidthRepresentation(width, true);
            }
        }
    }
    return toReturn;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#init(
 *     ro.sync.ecss.extensions.api.node.AuthorElement)
 */
public void init(AuthorElement tableElement) {
    this.tableElement = tableElement;
    AuthorElement[] colChildren = tableElement.getElementsByLocalName("customcol");
    if (colChildren != null && colChildren.length > 0) {
        for (int i = 0; i < colChildren.length; i++) {
            AuthorElement colChild = colChildren[i];
            if (i == 0) {
                colsStartOffset = colChild.getStartOffset();
            }
            if (i == colChildren.length - 1) {
                colsEndOffset = colChild.getEndOffset();
            }
            // Determine the 'width' for this col.
            AttrValue colWidthAttribute = colChild.getAttribute("width");
            String colWidth = null;
            if (colWidthAttribute != null) {
                colWidth = colWidthAttribute.getValue();
                // Add WidthRepresentation objects for the columns this 'customcol' specif
                // spans over.
                colWidthSpecs.add(new WidthRepresentation(colWidth, true));
            }
        }
    }
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#isAcceptingFixedColumnW
 *    java.lang.String)
 */
public boolean isAcceptingFixedColumnWidths(String tableCellsTagName) {
 return true;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#isAcceptingPercentageCo
 *    java.lang.String)
 */
public boolean isAcceptingPercentageColumnWidths(String tableCellsTagName) {
 return true;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#isAcceptingProportional
```

```
 *    java.lang.String)
 */
public boolean isAcceptingProportionalColumnWidths(String tableCellsTagName) {
 return true;
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#isTableAcceptingWidth(
 *    java.lang.String)
 */
public boolean isTableAcceptingWidth(String tableCellsTagName) {
 return "td".equals(tableCellsTagName);
}

/**
 * @see ro.sync.ecss.extensions.api.AuthorTableColumnWidthProvider#isTableAndColumnsResiza
 *     java.lang.String)
 */
public boolean isTableAndColumnsResizable(String tableCellsTagName) {
 return "td".equals(tableCellsTagName);
}

/**
 * @see ro.sync.ecss.extensions.api.Extension#getDescription()
 */
public String getDescription() {
 return "Implementation for the Simple Documentation Framework table layout.";
}
}
```

ReferencesResolver.java

```
package simple.documentation.framework;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.transform.sax.SAXSource;

import org.apache.log4j.Logger;
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;

import ro.sync.ecss.extensions.api.AuthorAccess;
import ro.sync.ecss.extensions.api.AuthorReferenceResolver;
import ro.sync.ecss.extensions.api.node.AttrValue;
import ro.sync.ecss.extensions.api.node.AuthorElement;
import ro.sync.ecss.extensions.api.node.AuthorNode;

/**
 * Resolver for content referred by elements named 'ref' with a
```

```
 *    'location' attribute.
 */
public class ReferencesResolver implements AuthorReferenceResolver {

  /**
   * Logger for logging.
   */
  private static Logger logger = Logger.getLogger(
          ReferencesResolver.class.getName());

  /**
   * Verifies if the handler considers the node to have references.
   *
   * @param node The node to be analyzed.
   * @return <code>true</code> if it is has references.
   */
  public boolean hasReferences(AuthorNode node) {
    boolean hasReferences = false;
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
      AuthorElement element = (AuthorElement) node;
      if ("ref".equals(element.getLocalName())) {
        AttrValue attrValue = element.getAttribute("location");
        hasReferences = attrValue != null;
      }
    }
    return hasReferences;
  }

  /**
   * Returns the name of the node that contains the expanded referred content.
   *
   * @param node The node that contains references.
   * @return The display name of the node.
   */
  public String getDisplayName(AuthorNode node) {
    String displayName = "ref-fragment";
    if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
      AuthorElement element = (AuthorElement) node;
      if ("ref".equals(element.getLocalName())) {
        AttrValue attrValue = element.getAttribute("location");
        if (attrValue != null) {
          displayName = attrValue.getValue();
        }
      }
    }
    return displayName;
  }

  /**
   * Resolve the references of the node.
   *
   * The returning SAXSource will be used for creating the referred content
   * using the parser and source inside it.
   *
```

```
 * @param node        The clone of the node.
 * @param systemID  The system ID of the node with references.
 * @param authorAccess The author access implementation.
 * @param entityResolver The entity resolver that can be used to resolve:
 *
 * <ul>
 *  <li>Resources that are already opened in editor.
 *  For this case the InputSource will contains the editor content.</li>
 *  <li>Resources resolved through XML catalog.</li>
 * </ul>
 *
 * @return The SAX source including the parser and the parser's input source.
 */
public SAXSource resolveReference(
    AuthorNode node,
    String systemID,
    AuthorAccess authorAccess,
    EntityResolver  entityResolver) {
  SAXSource saxSource = null;

  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(new URL(systemID),
                  authorAccess.correctURL(attrStringVal));

          InputSource inputSource = entityResolver.resolveEntity(null,
                  absoluteUrl.toString());
          if(inputSource == null) {
            inputSource = new InputSource(absoluteUrl.toString());
          }

          XMLReader xmlReader = authorAccess.newNonValidatingXMLReader();
          xmlReader.setEntityResolver(entityResolver);

          saxSource = new SAXSource(xmlReader, inputSource);
        } catch (MalformedURLException e) {
          logger.error(e, e);
        } catch (SAXException e) {
          logger.error(e, e);
        } catch (IOException e) {
          logger.error(e, e);
        }
      }
    }
  }

  return saxSource;
}
```

```
/**
 * Get an unique identifier for the node reference.
 *
 * The unique identifier is used to avoid resolving the references
 *     recursively.
 *
 * @param node The node that has reference.
 * @return  An unique identifier for the reference node.
 */
public String getReferenceUniqueID(AuthorNode node) {
  String id = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        id = attrValue.getValue();
      }
    }
  }
  return id;
}

/**
 * Return the systemID of the referred content.
 *
 * @param node The reference node.
 * @param authorAccess The author access.
 *
 * @return The systemID of the referred content.
 */
public String getReferenceSystemID(AuthorNode node,
        AuthorAccess authorAccess) {
  String systemID = null;
  if (node.getType() == AuthorNode.NODE_TYPE_ELEMENT) {
    AuthorElement element = (AuthorElement) node;
    if ("ref".equals(element.getLocalName())) {
      AttrValue attrValue = element.getAttribute("location");
      if (attrValue != null) {
        String attrStringVal = attrValue.getValue();
        try {
          URL absoluteUrl = new URL(node.getXMLBaseURL(),
              authorAccess.correctURL(attrStringVal));
          systemID = absoluteUrl.toString();
        } catch (MalformedURLException e) {
          logger.error(e, e);
        }
      }
    }
  }
  return systemID;
}

/**
```

```
   * Verifies if the references of the given node must be refreshed
   * when the attribute with the specified name has changed.
   *
   * @param node The node with the references.
   * @param attributeName The name of the changed attribute.
   * @return <code>true</code> if the references must be refreshed.
   */
  public boolean isReferenceChanged(AuthorNode node, String attributeName) {
    return "location".equals(attributeName);
  }

  /**
   * @return The description of the author extension.
   */
  public String getDescription() {
    return "Resolves the 'ref' references";
  }
}
```

CustomRule.java

```
package simple.documentation.framework;

import org.xml.sax.Attributes;

import ro.sync.ecss.extensions.api.DocumentTypeCustomRuleMatcher;

public class CustomRule implements
                DocumentTypeCustomRuleMatcher {
  /**
   * Checks if the root namespace is the one
   * of our documentation framework.
   */
  public boolean matches(
      String systemID,
      String rootNamespace,
      String rootLocalName,
      String doctypePublicID,
      Attributes rootAttributes) {

    return
      "http://www.oxygenxml.com/sample/documentation".equals(rootNamespace);
  }

  public String getDescription() {
    return
  "Checks if the current Document Type Association is matching the document.";
  }
}
```

DefaultElementLocatorProvider.java

```
package ro.sync.ecss.extensions.commons;
```

```
import org.apache.log4j.Logger;

import ro.sync.ecss.extensions.api.link.ElementLocator;
import ro.sync.ecss.extensions.api.link.ElementLocatorException;
import ro.sync.ecss.extensions.api.link.ElementLocatorProvider;
import ro.sync.ecss.extensions.api.link.IDTypeVerifier;

/**
 * Default implementation for locating elements based on a given link.
 * Depending on the link structure the following cases are covered:
 * - xinclude element scheme :  element(/1/2) see
 *        http://www.w3.org/TR/2003/REC-xptr-element-20030325/
 * - ID based links : the link represents the value of an attribute of type ID
 */
public class DefaultElementLocatorProvider implements ElementLocatorProvider {
  /** * Logger for logging. */
  private static Logger logger = Logger.getLogger(
          DefaultElementLocatorProvider.class.getName());
  /**
   * @see ro.sync.ecss.extensions.api.link.ElementLocatorProvider#
   *     getElementLocator(ro.sync.ecss.extensions.api.link.IDTypeVerifier,
   *     java.lang.String)
   */
  public ElementLocator getElementLocator(IDTypeVerifier idVerifier,
          String link) {
    ElementLocator elementLocator = null;
    try {
      if(link.startsWith("element(")){
        // xpointer element() scheme
        elementLocator = new XPointerElementLocator(idVerifier, link);
      } else {
        // Locate link element by ID
        elementLocator = new IDElementLocator(idVerifier, link);
      }
    } catch (ElementLocatorException e) {
      logger.warn("Exception when create element locator for link: "
          + link + ". Cause: " + e, e);
    }
    return elementLocator;
  }

  /**
   * @see ro.sync.ecss.extensions.api.Extension#getDescription()
   */
  public String getDescription() {
    return
    "Default implementation for locating elements based on a given link. \n" +
    "The following cases are covered: xinclude element scheme "
        + "and ID based links.";
  }
}
```

XPointerElementLocator.java

```
package ro.sync.ecss.extensions.commons;

import java.util.Stack;
import java.util.StringTokenizer;

import org.apache.log4j.Logger;

import ro.sync.ecss.extensions.api.link.Attr;
import ro.sync.ecss.extensions.api.link.ElementLocator;
import ro.sync.ecss.extensions.api.link.ElementLocatorException;
import ro.sync.ecss.extensions.api.link.IDTypeVerifier;

/**
 * Element locator for links that have the one of the following pattern:
 * <ul>
 *   <li>element(elementID) - locate the element with the same id</li>
 *   <li>element(/1/2/5) - A child sequence appearing alone identifies an
 *   element by means of stepwise navigation, which is directed by a
 *   sequence of integers separated by slashes (/); each integer n locates
 *   the nth child element of the previously located element. </li>
 *   <li>element(elementID/3/4) - A child sequence appearing after an
 *   NCName identifies an element by means of stepwise navigation,
 *   starting from the element located by the given name.</li>
 * </ul>
 *
 */
public class XPointerElementLocator extends ElementLocator {

  /**
   * Logger for logging.
   */
  private static Logger logger = Logger.getLogger(
          XPointerElementLocator.class.getName());

  /**
   * Verifies if a given attribute is of a type ID.
   */
  private IDTypeVerifier idVerifier;

  /**
   * XPointer path, the path to locate the linked element.
   */
  private String[] xpointerPath;

  /**
   * The stack with indexes in parent of the current iterated elements.
   */
  private Stack currentElementIndexStack = new Stack();

  /**
   * The number of elements in xpointer path.
```

```
 */
private int xpointerPathDepth;

/**
 * If true then the XPointer path starts with an element ID.
 */
private boolean startWithElementID = false;

/**
 * The depth of the current element in document, incremented in startElement.
 */
private int startElementDepth = 0;

/**
 * Depth in document in the last endElement event.
 */
private int endElementDepth = 0;

/**
 * The index in parent of the previous iterated element. Set in endElement().
 */
private int lastIndexInParent;

/**
 * Constructor.
 *
 * @param idVerifier Verifies if an given attribute is of type ID.
 * @param link The link that gives the element position.
 * @throws ElementLocatorException  When the link format is not supported.
 **/
public XPointerElementLocator(IDTypeVerifier idVerifier, String link)
        throws ElementLocatorException {
  super(link);
  this.idVerifier = idVerifier;

  link = link.substring("element(".length(), link.length() - 1);

  StringTokenizer stringTokenizer = new StringTokenizer(link, "/", false);
  xpointerPath = new String[stringTokenizer.countTokens()];
  int i = 0;
  while (stringTokenizer.hasMoreTokens()) {
    xpointerPath[i] = stringTokenizer.nextToken();
    boolean invalidFormat = false;

    // Empty xpointer component is not supported
    if(xpointerPath[i].length() == 0){
      invalidFormat = true;
    }

    if(i > 0){
      try {
        Integer.parseInt(xpointerPath[i]);
      } catch (NumberFormatException e) {
        invalidFormat = true;
```

```
      }
    }

    if(invalidFormat){
      throw new ElementLocatorException(
      "Only the element() scheme is supported when locating XPointer links."
      + "Supported formats: element(elementID), element(/1/2/3),
          element(elemID/2/3/4).");
    }
    i++;
  }

  if(Character.isDigit(xpointerPath[0].charAt(0))){
    // This is the case when xpointer have the following pattern /1/5/7
    xpointerPathDepth = xpointerPath.length;
  } else {
    // This is the case when xpointer starts with an element ID
    xpointerPathDepth = -1;
    startWithElementID  = true;
  }
}

/**
 * @see ro.sync.ecss.extensions.api.link.ElementLocator#endElement(
 *     java.lang.String, java.lang.String, java.lang.String)
 */
public void endElement(String uri, String localName, String name) {
  endElementDepth = startElementDepth;
  startElementDepth --;
  lastIndexInParent = ((Integer)currentElementIndexStack.pop()).intValue();
}

/**
 * @see ro.sync.ecss.extensions.api.link.ElementLocator#startElement(
 *    java.lang.String, java.lang.String, java.lang.String,
 *    ro.sync.ecss.extensions.api.link.Attr[])
 */
public boolean startElement(String uri, String localName,
        String name, Attr[] atts) {
  boolean linkLocated = false;
  // Increase current element document depth
  startElementDepth  ++;

  if (endElementDepth != startElementDepth) {
    // The current element is the first child of the parent
    currentElementIndexStack.push(new Integer(1));
  } else {
    // Another element in the parent element
    currentElementIndexStack.push(new Integer(lastIndexInParent + 1));
  }

  if (startWithElementID) {
    // This the case when xpointer path starts with an element ID.
    String xpointerElement = xpointerPath[0];
```

```
      for (int i = 0; i < atts.length; i++) {
        if(xpointerElement.equals(atts[i].getValue())){
          if(idVerifier.hasIDType(
              localName, uri, atts[i].getQName(), atts[i].getNamespace())){
            xpointerPathDepth = startElementDepth + xpointerPath.length - 1;
            break;
          }
        }
      }

    if(xpointerPathDepth == startElementDepth){
      // check if xpointer path matches with the current element path
      linkLocated = true;
      try {
        int xpointerIdx = xpointerPath.length - 1;
        int stackIdx = currentElementIndexStack.size() - 1;
        int stopIdx = startWithElementID ? 1 : 0;
        while (xpointerIdx >= stopIdx && stackIdx >= 0) {
          int xpointerIndex = Integer.parseInt(xpointerPath[xpointerIdx]);
          int currentElementIndex = ((Integer)currentElementIndexStack.
                  get(stackIdx)).intValue();
          if(xpointerIndex != currentElementIndex) {
            linkLocated = false;
            break;
          }

          xpointerIdx--;
          stackIdx--;
        }

      } catch (NumberFormatException e) {
        logger.warn(e,e);
      }
    }
    return linkLocated;
  }
}
```

IDElementLocator.java

```
package ro.sync.ecss.extensions.commons;

import ro.sync.ecss.extensions.api.link.Attr;
import ro.sync.ecss.extensions.api.link.ElementLocator;
import ro.sync.ecss.extensions.api.link.ExtensionUtil;
import ro.sync.ecss.extensions.api.link.IDTypeVerifier;

/**
 * Implementation of an ElementLocator that treats the link as the value of an
 *     attribute with the type ID.
 */
public class IDElementLocator extends ElementLocator {
```

```
  /**
   * Class able to tell if a given attribute is of type ID.
   */
  private IDTypeVerifier idVerifier;

  /**
   * Constructor.
   *
   * @param idVerifier It tells us if an attribute is of type ID.
   * @param link The link used to identify an element.
   */
  public IDElementLocator(IDTypeVerifier idVerifier, String link) {
    super(link);
    this.idVerifier = idVerifier;
  }

  /**
   * @see ro.sync.ecss.extensions.api.link.ElementLocator#endElement(
   *         java.lang.String, java.lang.String, java.lang.String)
   */
  public void endElement(String uri, String localName, String name) {
    // Nothing to do.
  }

  /**
   * @see ro.sync.ecss.extensions.api.link.ElementLocator#startElement(
   *    java.lang.String, java.lang.String, java.lang.String,
   *    ro.sync.ecss.extensions.api.link.Attr[])
   */
  public boolean startElement(String uri, String localName,
          String name, Attr[] atts) {
    boolean elementFound = false;
    for (int i = 0; i < atts.length; i++) {
      if (link.equals(atts[i].getValue())) {
        if("xml:id".equals(atts[i].getQName())) {
          // xml:id attribute
          elementFound = true;
        } else {
          // check if attribute has ID type
          String attrLocalName =
            ExtensionUtil.getLocalName(atts[i].getQName());
          String attrUri = atts[i].getNamespace();
          if (idVerifier.hasIDType(localName, uri, attrLocalName, attrUri)) {
            elementFound = true;
          }
        }
      }
    }

    return elementFound;
  }
}
```